

Higher-Order Rewriting

Femke van Raamsdonk

Division of Mathematics and Computer Science
Faculty of Sciences, Vrije Universiteit
De Boelelaan 1081a, 1081 HV Amsterdam
The Netherlands
femke@cs.vu.nl

CWI

P.O. Box 94079, 1090 GB Amsterdam

Note: This paper will appear in the proceedings of the 10th international conference on rewriting techniques and applications (RTA '99). ©Springer Verlag.

1 Introduction

The wish to consider rewriting systems with bound variables emerges naturally. The various equations with bound variables that are present in both logic and mathematics give rise to rewrite rules as soon as they are oriented. The β -axiom of lambda-calculus is oriented as $(\lambda x.M)N \rightarrow M[x := N]$. The so-obtained rewriting system was used to provide consistency proofs. Another well-known equation with bound variables is the axiom for μ -recursion. Its usual orientation gives rise to the rewrite rule $\mu x.M \rightarrow M[x := \mu x.M]$. Equivalences in logic may contain bound variables, like in $\neg\exists x.P(x) \leftrightarrow \forall x.\neg P(x)$. Also these equivalences can be turned into rewrite rules. Moreover, rules for proof normalisation may contain bound variables. Many equations occurring in mathematics contain bound variables, for instance if derivatives or integrals are present, like in $\int f(x) + g(x)dx = \int f(x)dx + \int g(x)dx$. Some aspects of the theory of a set of equations can be studied by considering the equations as rewrite rules, as is also done for first-order equations without bound variables. Further, functional programming languages may contain specifications of functions that take functions as arguments, like for instance the function `map` that takes as argument a function and a list, and that applies the function to every element of the list.

This explains the need for a unifying theory of higher-order rewriting, where the rewrite rules may contain bound variables.

The present paper aims at providing readers familiar with first-order term rewriting some intuitions of higher-order rewriting. The important feature of higher-order rewriting is that besides all first-order rewrite rules also rules with bound variables, like the ones for β and μ , can be expressed. I focus on two subjects in the theory of rewriting, namely confluence and termination, and discuss how some results concerning these subjects can be generalised from the first-order to the higher-order case. It is certainly not the intention to present this generalisation in a completely formal way; I just try to explain what difficulties arise due to the presence of bound variables, and how they can be overcome.

Very generally speaking, the additional combinatorial complexity of higher-order rewriting compared to first-order rewriting is caused by the possibility of nesting. For instance in the higher-order term rewriting system $\{f(x.z(x)) \rightarrow z(z(x)), g(z) \rightarrow h(z)\}$ we have a rewrite step $f(x.g(x)) \rightarrow g(g(x))$ in which the redex $g(x)$ is duplicated and the two residuals are nested. One of the consequences of this phenomenon is that invariants used in proofs should be closed under substitution. This theme plays a rôle throughout the paper.

A systematic study of rewriting systems with bound variables starts with the work by Klop, who introduces in [16] the class of combinatory reduction systems. Another impulse for the study of rewriting with bound variables originates in the work by Nipkow, who defines in [22] the class of higher-order rewrite systems. Combinatory reduction systems form a generalisation of the class of contraction schemes defined in [1], and can more generally be seen as standing in a tradition where extensions of lambda-calculus are studied. Examples of such extensions are lambda-calculus extended with infinitely many δ -rules that test for equality of closed normal forms, defined by Church (see [3]), lambda-calculus with rules for surjective pairing, and the class of $\lambda(a)$ -reductions which consists of lambda-calculus with constants and rewrite rules for these constants [10]. Higher-order rewrite systems were introduced with the aim to study the meta-theory of systems like Isabelle and λ -Prolog.

The presentation of higher-order rewriting systems in this paper is mainly based on [25, 32], which builds on [16, 22]. However, I would like to stress that the actual format of higher-order rewriting is not of the utmost importance here, since the paper is informal in nature. Moreover, the essence of concepts and proofs does not depend on the details of the chosen format.

For various purposes other classes of rewriting systems with bound variables are introduced in the literature. To mention a two of them: expression reduction systems [15] are similar to combinatory reduction systems, but have been introduced independently, and interaction systems [2] form a subclass of combinatory reduction systems that is introduced for the study of optimality. It is not hard to adapt the presentation of these two classes of systems, and of the one of combinatory reduction systems and higher-order rewrite systems, to the presentation of higher-order rewriting as chosen for this paper.

Although the theory of higher-order rewriting is not as widely developed as the one of lambda-calculus or first-order rewriting, it is nevertheless by no means possible to give a complete overview in the present paper. Readers interested in the theory of equational reasoning and narrowing for higher-order rewriting are referred to [23, 30] and the literature mentioned there. Further, results concerning confluence and termination can be found in detail in [16, 25, 19, 29, 32].

Acknowledgements. I am grateful to Jan Willem Klop and Vincent van Oostrom for discussions that in the course of the last years always have been, and still are, a source of inspiration. I wish to thank Jean-Pierre Jouannaud, Aart Middeldorp, Jaco van de Pol, and Roel de Vrijer for helpful discussions and comments. The diagrams are designed using the package Xy-pic of K.R. Rose.

2 Higher-Order Rewriting

We assume a set of *base types*. A *simple type*, usually shortly called a type, is either a base type or an expression of the form $A \rightarrow B$, with A and B types. For every type A a set consisting of infinitely many *variables* of type A is assumed. Variables are written as x, y, z, \dots , if necessary decorated with their types.

A *signature* is a set of function symbols with a unique type. Function symbols are written as f, g, h, a, b, \dots ; sometimes a more suggestive notation is used.

A *preterm* of type A is a simply typed lambda-term of type A , inductively defined as follows, where $s : A$ denotes that s is a preterm of type A :

1. variables of type A and function symbols of type A are preterms of type A ,
2. if x is a variable of type A , and $s : B$, then $x.s : A \rightarrow B$,
3. if $s : A \rightarrow B$ and $t : A$, then $st : B$.

Preterms are denoted by s, t, r, \dots . Note that abstraction is written as $x.s$ instead of $\lambda x.s$. In the preterm $x.s$, occurrences of the variables x in s are bound. A variable occurrence that is not bound is said to be free. A preterm without bound variables is closed. Preterms are considered up to the equivalence relation generated by the renaming of bound variables, or α -conversion.

In the remainder of this paper, all preterms are supposed to be in long- η -normal form (also written as $\bar{\eta}$ -normal form). That means that every subterm has the maximal number of arguments according to its type. Note that every type can be written as $A_1 \rightarrow \dots \rightarrow A_n \rightarrow B$ with B a base type. In an $\bar{\eta}$ -normal form, a function symbol of type $A_1 \rightarrow \dots \rightarrow A_n \rightarrow B$ occurs always in a subterm of the form $f s_1 \dots s_n$ with $s_1 : A_1, \dots, s_n : A_n$, and similarly for variables. This permits to adopt the functional notation $f(s_1, \dots, s_n)$ instead of the applicative notation $f s_1 \dots s_n$, and similarly for variables; this is often done in the sequel. A sequence of expressions e is usually abbreviated as e .

We work modulo the equivalence generated by the β -reduction rule, which is given as $(x.s)t \rightarrow_\beta s[x := t]$, with s and t arbitrary preterms. Here $s[x := t]$ denotes the result of substituting t for every free occurrence of x in s , renaming variables if necessary in order to avoid unintended capture of variables.

We make use of the facts that β -reduction is confluent and terminating on simply typed lambda-calculus, and that the set of $\bar{\eta}$ -normal forms is closed under β -reduction. Every β -equivalence class of preterms contains a unique β -normal form, which is used as a representative. Such a representative is called a *term*. Terms are the objects that are rewritten in a higher-order rewriting system.

For the definition of a rewrite rule we first need to introduce the notion of a rule-pattern, which is a slight adaptation of the notion of pattern introduced in [21]. A *rule-pattern* is a term of the form $\mathbf{x}.f(\mathbf{s})$ such that, first, every $x_i \in \mathbf{x}$ occurs free in $f(\mathbf{s})$, and second, every $x_i \in \mathbf{x}$ occurs only in subterms of the form $x_i(y_1, \dots, y_m)$ with y_1, \dots, y_m the $\bar{\eta}$ -normal forms of different bound variables not among \mathbf{x} . The terms $z.f(z)$ with z of some base type, and $z.f(x.z(x))$ with z of type $A \rightarrow B$ with B a base type, are examples of rule-patterns. The term $z.f(z(x))$ is not a rule-pattern because x is not bound, the term $z.f(x.z(x, x))$ is

not a rule-pattern because the arguments of z are not different bound variables, and the term $z.z$ is not a rule-pattern because there is no occurrence of a function symbol after the outermost abstractions.

A *rewrite rule* of a higher-order rewriting system is defined as a pair of closed terms of the form $z.l \rightarrow z.r$, with $z.l$ a rule-pattern. In this paper, a higher-order rewriting system is usually described by giving the set of its rewrite rules.

The next thing to define is the rewrite relation of a higher-order rewriting system. To that end we need to introduce the notion of context. We use the symbol \square^A , or simply \square , to denote a variable of type A that is supposed to be free. A *context* of type A is a term with one occurrence of \square^A . The preterm obtained by replacing \square^A by a term s of type A is denoted by $C[s]$.

Now the *rewrite relation* of a higher-order rewriting system, denoted by \rightarrow , is the relation on terms that is defined as follows: we have $s \rightarrow t$ if there is a context C of type A , and a rewrite rule $l \rightarrow r$ with l and r of type A , such that s is the β -normal form of $C[l]$ and t is the β -normal form of $C[r]$. That is, such a rewrite step can be decomposed as

$$s \stackrel{!}{\beta} \leftarrow C[l] \rightarrow C[r] \rightarrow \stackrel{!}{\beta} t$$

where $\rightarrow \stackrel{!}{\beta}$ denotes β -reduction to β -normal form. The requirement that the left-hand side of a rewrite rule must be a pattern makes the rewrite relation decidable [21].

In first-order term rewriting, a rewrite step is usually defined as $C[l\sigma] \rightarrow C[r\sigma]$, with $l \rightarrow r$ a rewrite rule, C a context, and σ an assignment. In higher-order rewriting, simply typed lambda-calculus with β -reduction is used to assign values to free variables in rewrite rules. Therefore lambda-calculus is called the *substitution calculus* of higher-order rewriting as defined here. It is possible to consider other calculi as substitution calculus. As a matter of fact, the substitution calculus of combinatory reduction systems and of expression reduction systems is not simply typed lambda-calculus with reduction to β -normal form, but untyped lambda-calculus with complete developments.

The rewrite rule $f(z) \rightarrow g(z)$, in the usual format of first-order term rewriting, induces for instance a rewrite step $f(a) \rightarrow g(a)$. This rewrite rule is now written as $z.f(z) \rightarrow z.g(z)$, and the rewrite step $f(a) \rightarrow g(a)$ is obtained as follows: $f(a) \stackrel{!}{\beta} \leftarrow (z.f(z)) a \rightarrow (z.g(z)) a \rightarrow \stackrel{!}{\beta} g(a)$.

The signature of lambda-calculus consists of the two function symbols $\text{app} : \text{term} \rightarrow \text{term} \rightarrow \text{term}$ and $\text{abs} : (\text{term} \rightarrow \text{term}) \rightarrow \text{term}$. Here term is the only base type; intuitively speaking it stands for the set of lambda-terms. The rewrite rules are given as follows:

$$\begin{aligned} z.z'. \text{app}(\text{abs}(x.z(x)), z') &\rightarrow_{\text{b}} z.z'. z(z') \\ z. \text{abs}(x.\text{app}(z, x)) &\rightarrow_{\text{e}} z. z \end{aligned}$$

The rewrite step $\text{app}(\text{abs}(x.x), y) \rightarrow_{\text{beta}} y$ is obtained as follows:

$$\text{app}(\text{abs}(x.x), y) \stackrel{!}{\beta} \leftarrow (z.z'. \text{app}(\text{abs}(x.z(x)), z'))(x.x)y \rightarrow_{\text{b}} (z.z'. z(z'))(x.x)y \rightarrow \stackrel{!}{\beta} y.$$

Usually we adopt the convention that the outermost abstractions of rewrite rules are not written, that is, we write $l \rightarrow r$ instead of $z.l \rightarrow z.r$.

3 Confluence

A rewriting system is *confluent* if every two divergent rewrite sequences can be joined; that is, whenever $t' \leftarrow s \rightarrow t$ there exists a term r with $t' \rightarrow r \leftarrow t$. Confluence is an important property because it guarantees that normal forms are unique. Moreover it provides a method to prove consistency. In this section we discuss some results concerning confluence of higher-order term rewriting.

3.1 Orthogonality

A first general confluence result for first-order term rewriting systems is the one stating that *orthogonal* term rewriting systems are confluent [33]. Orthogonality of a rewriting system roughly speaking expresses that rewrite steps are independent. In the setting of first-order term rewriting, it is usually formalised by two requirements imposed on the rewrite rules. These requirements arise from an analysis of the different ways in which contraction of a redex can destroy another redex that is present in the same term.

First, a redex u can be destroyed by contraction of a redex u' if two subterms in u must be identical and contraction of u' changes one of them. For example in the step $f(a, a) \rightarrow f(b, a)$ in the term rewriting system $\{a \rightarrow b, f(x, x) \rightarrow b\}$, the redex $f(a, a)$ is destroyed by contraction of the leftmost redex a . This kind of interference may cause a rewriting system to be non-confluent: in the term rewriting system $\{f(x, x) \rightarrow a, f(x, g(x)) \rightarrow b, c \rightarrow g(c)\}$ given in [11], we have both $f(c, c) \rightarrow a$ and $f(c, c) \rightarrow b$. Another example, given in [16], is the term rewriting system $\{d(x, x) \rightarrow e, c(x) \rightarrow d(x, c(x)), a \rightarrow c(a)\}$. We have $c(a) \rightarrow e$ and $c(a) \rightarrow c(e)$. The source of non-confluence as illustrated here is ruled out by requiring the term rewriting system to be *left-linear*, which means that variables are not allowed to occur more than once in the left-hand side of a rewrite rule.

Second, a redex u can be destroyed by contraction of a redex u' if that contraction changes one or more symbols in the pattern of u . Consider for instance the term rewriting system $\{a \rightarrow b, f(a) \rightarrow c\}$. In the step $f(a) \rightarrow f(b)$, the redex $f(a)$ is destroyed by the contraction of the redex a . The redexes a and $f(a)$ are said to be *overlapping* in the term $f(a)$. Clearly, the presence of overlapping redexes can cause a rewriting system to be non-confluent. Ruling out this source of non-confluence can be done by requiring that in all terms all redexes are non-overlapping. This requirement is guaranteed by a condition concerning the rewrite rules only; the formulation of this condition uses some additional terminology that we discuss now.

A *critical pair* arises from a most general way in which two redexes can be overlapping. Consider for example the term rewriting system $\{a \rightarrow b, f(a, x) \rightarrow g(x)\}$. The term $f(a, x)$ can be rewritten to $f(b, x)$ by an application of the first rewrite rule, and to $g(x)$ by an application of the second rewrite rule. The pair of terms $(f(b, x), g(x))$ is said to be a critical pair. The requirement of maximal generality concerns the surroundings of the two redexes that give rise to the critical pair. The intuition is that the surroundings should be taken to be minimal. For instance, the symbol g is not essential for the overlap between the

redexes a and $f(a, x)$ in the term $g(f(a, x))$; hence $(g(f(b, x)), g(g(x)))$ is not a critical pair. The symbol b is not essential for the overlap between the redexes a and $f(a, b)$ in the term $f(a, b)$; hence $(f(b, b), g(b))$ is not a critical pair either.

The absence of critical pairs in a term rewriting system guarantees that in all terms all redexes are non-overlapping.

A first-order term rewriting system is defined to be *orthogonal* if all its rules are left-linear and it has no critical pairs. At some places in the literature the definition of orthogonality requires instead of the absence of critical pairs that in all terms all redexes are non-overlapping; the two formulations are equivalent.

Now the question arises whether the intuition of orthogonality, namely that rewrite steps are independent, is also for higher-order systems properly captured by requiring that rewrite rules are left-linear and that there are no critical pairs. As it turns out, the presence of bound variables does not yield another way in which contraction of a redex can destroy another redex. Note, however, that there is another way in which contraction of a redex can create another redex: by erasure of a bound variable a redex can be created for a rule with a so-called non-occur check. This happens for instance in the rewrite step $f(x.g(x)) \rightarrow f(x.a)$ in the rewriting system $\{f(x.z) \rightarrow a, g(x) \rightarrow a\}$. This phenomenon plays a rôle in the study of termination rather than in that of confluence.

So the definition of orthogonality for higher-order term rewriting systems is in essence the same as for first-order term rewriting systems. However, in particular the definition of critical pair is technically more complicated. Also unification of higher-order patterns is technically more complicated than that of first-order terms; it still decidable [21] and moreover has linear complexity [31]. Here we do not give a completely formal definition of critical pair, which can be found for instance in [22, 18].

Definition 1.

1. A rewrite rule $\mathbf{x}.l \rightarrow \mathbf{x}.r$ of a higher-order rewriting system is left-linear if every variable $x \in \mathbf{x}$ occurs exactly once in l .
2. Let $C[ls] = gt$ indicate a most general overlap between two redexes, with $l \rightarrow r$ and $g \rightarrow d$ rewrite rules. Then $(C[rs], dt)$ is a critical pair.

Note that a critical pair is ordered. If the context C in the definition is the empty one, then the rewrite rules $l \rightarrow r$ and $g \rightarrow d$ must be different. The critical pair of lambda-calculus with $\beta\eta$ -reduction is $(\mathbf{app}(z, z'), \mathbf{app}(z, z'))$; it arises from the most general overlap in $\mathbf{app}(\mathbf{abs}(x.\mathbf{app}(z, x)), z')$. The definition of orthogonality is now the same as the one for first-order term rewriting.

Definition 2. A higher-order rewriting system is orthogonal if it has no critical pairs and all its rewrite rules are left-linear.

There are two important methods to prove that orthogonal rewriting systems are confluent. The remainder of this subsection is devoted to a discussion of these two methods. The first one, that of simultaneous reduction, makes essential use of the structure of the expressions that are rewritten. The second one, that uses complete developments, is more abstract in nature. Both apply to higher-order rewriting systems, so we have the following result.

Theorem 1. *Orthogonal higher-order rewriting systems are confluent.*

Simultaneous Reduction. First we discuss a method to prove confluence that makes use of simultaneous reduction. It is introduced by Tait and Martin-Löf in their proof of confluence of lambda-calculus with β -reduction, see [3]. The outline of this method is as follows: we inductively define a relation \Rightarrow satisfying the following two properties:

1. It has the diamond property, that is, $t' \Rightarrow r \Leftarrow t$ whenever $t' \Leftarrow s \Rightarrow t$.
2. Its reflexive-transitive closure equals the rewrite relation (\twoheadrightarrow).

It is not difficult to see that this indeed yields confluence of rewriting.

For first-order term rewriting systems, one can take for \Rightarrow *parallel rewriting*. This is the relation that contracts a set of redexes that are in parallel in one step. For instance, in the rewriting system $\{a \rightarrow a', f(x) \rightarrow f'(x)\}$, where there is moreover a binary function symbol g , we have $g(a, a) \dashv\vdash g(a', a')$. The two redexes in the term $f(f(x))$ are not parallel but *nested*, and hence we do not have $f(f(x)) \dashv\vdash f'(f'(x))$. Note however that $f(f(x)) \dashv\vdash f'(f(x))$ and $f(f(x)) \dashv\vdash f(f'(x))$ both hold. The relation $\dashv\vdash$ is defined inductively. The rule

$$l\sigma \dashv\vdash r\sigma$$

with $l \rightarrow r$ a rewrite rule and σ an assignment, expresses that redexes can be contracted, and the rule

$$\frac{s_1 \dashv\vdash s'_1 \dots s_n \dashv\vdash s'_n}{f(s_1, \dots, s_n) \dashv\vdash f(s'_1, \dots, s'_n)}$$

permits to do so in parallel. Finally, there is the rule $x \dashv\vdash x$, with x a variable. The relation $\dashv\vdash$ is reflexive and compatible with the term structure. It can be shown that the reflexive-transitive closure of the parallel rewriting relation equals \twoheadrightarrow , and moreover that $\dashv\vdash$ satisfies the diamond property.

It is possible adapt the definition of $\dashv\vdash$ to the case of higher-order rewriting, and to show that its reflexive-transitive closure equals the rewrite relation. However, due to the possibility of nesting, $\dashv\vdash$ doesn't have the diamond property. Consider for instance the higher-order rewriting system $\{f(x, z(x)) \rightarrow z(z(a)), g(z) \rightarrow h(z)\}$. We have $f(x, g(x)) \dashv\vdash g(g(a))$ and $f(x, g(x)) \dashv\vdash f(x, h(x))$ but not $g(g(a)) \dashv\vdash h(h(a))$ (note however that we do have $f(x, h(x)) \dashv\vdash h(h(a))$).

In lambda-calculus there is also the possibility of nesting of redexes. Indeed, the definition of $\dashv\vdash$ adapted to lambda-calculus does not satisfy the diamond property either: we have $(\lambda x. (\lambda y. x) I) (I I) \dashv\vdash (\lambda y. I I) I$ and on the other hand $(\lambda x. (\lambda y. x) I) (I I) \dashv\vdash (\lambda x. x) I$ but there is no lambda-term M such that $(\lambda y. I I) I \dashv\vdash M$ and $(\lambda x. x) I \dashv\vdash M$.

The relation used by Tait and Martin-Löf in their proof of confluence is essentially different from parallel rewriting in that it permits to contract any set of redexes in a term in one step. The important clause in the definition of this relation, which we denote here by $\dashv\to$, is as follows:

$$\frac{M \multimap M' \quad N \multimap N'}{(\lambda x.M)N \multimap M'[x := N']}$$

This suggests that for higher-order rewriting we should define instead of \multimap a relation, which we denote by \multimap as the one for lambda-calculus, with

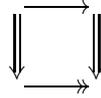
$$\frac{s_1 \multimap s'_1 \dots s_n \multimap s'_n}{\downarrow_\beta s_1 \dots s_n \downarrow_\beta \multimap r s'_1 \dots s'_n \downarrow_\beta}$$

where \downarrow_β denotes β -reduction to normal form. Further, \multimap must be compatible with the term structure. Note that, reconsidering the example give above, we have $g(g(a)) \multimap h(h(a))$. We call the relation \multimap here *simultaneous reduction*; it is also sometimes called parallel reduction.

The relation \multimap can be defined for left-linear higher-order rewriting systems. For orthogonal higher-order rewriting systems, it can be shown that the reflexive-transitive closure of \multimap equals \rightarrow , and further that \multimap satisfies the diamond property. As a consequence, we have the result that orthogonal higher-order rewriting systems are confluent.

Finite Developments. The second method to prove confluence we discuss is the one using developments. It is more abstract in nature than the one using simultaneous reduction, and can therefore more easily be adapted to situations where the rewriting system is not quite orthogonal or where the structures that are rewritten are not quite terms. Developments are used to show confluence of lambda-calculus in [5] and to show confluence of combinatory reduction systems in [16]. We follow here basically the account presented in [25, 26].

The idea is to define a relation \Rightarrow with $\rightarrow \subseteq \Rightarrow \subseteq \multimap$ that satisfies moreover the following diagram:



Confluence follows then by an easy induction, since the diagram above and the fact that $\rightarrow \subseteq \Rightarrow$ permit to construct the *projection* of a rewrite sequence d over a rewrite step, yielding a rewrite sequence e as follows:

$$\begin{array}{l} d : \quad s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \dots \\ \quad \quad \downarrow \quad \quad \Downarrow \quad \quad \Downarrow \quad \quad \Downarrow \\ e : \quad t_0 \multimap t_1 \multimap t_2 \multimap t_3 \multimap \dots \end{array}$$

The second method to prove confluence now proceeds as follows. We define a *development* of a set of redexes U as a special kind of rewrite sequence, namely one in which only residuals of redexes in U are contracted. A development of U is said to be *complete* if it cannot be extended, that is, there are no residuals of redexes in U anymore. In general there can be different ways to perform a complete development, for instance $f(a, a) \rightarrow f(b, a) \rightarrow f(b, b)$ and $f(a, a) \rightarrow f(a, b) \rightarrow f(b, b)$

are both complete developments of the set consisting of the two redexes a in the term $f(a, a)$. Further, it is a priori not clear whether performing a complete development terminates at some point. The crucial step in the confluence proof is the following result, which is called the Finite Developments Theorem.

Theorem 2. *All complete developments of a set of redexes U are finite, end in the same term, and induce the same descendant relation.*

Hence all ways of sequentialising the contraction of a set of redexes are finite, and essentially the same. As a consequence, the complete developments relation satisfies the diamond property, so we can take it for \Rightarrow .

We denote the complete developments relation just as simultaneous reduction by \multimap . This is justified since there is the following relation between simultaneous reduction and complete developments: a proof of $s \multimap t$ using the rules for simultaneous reduction corresponds to a complete development in which the redexes are contracted from the inside to the outside.

The proof of the Finite Developments Theorem is essentially more complex for lambda-calculus and higher-order rewriting than for first-order term rewriting systems, due the possibility of nesting. Consider for instance the higher-order rewriting system $\{f(x.z(x)) \rightarrow z(z(a)), g(z) \rightarrow h(z, z)\}$. In the complete development $f(x.g(x)) \rightarrow g(g(a)) \rightarrow h(g(a), g(a)) \rightarrow h(h(a, a), g(a)) \rightarrow h(h(a, a), h(a, a))$, two residuals of the redex $g(x)$ in the initial term first get nested and then the innermost one is duplicated by the outermost one.

The Finite Developments Theorem is not only useful to derive confluence, but in any situation where a rewrite sequence is projected over a rewrite step.

3.2 Weak Orthogonality

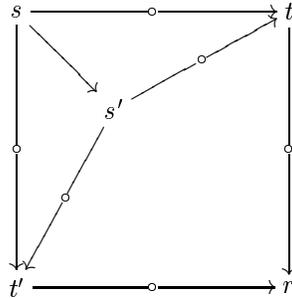
The methods to prove confluence of orthogonal higher-order rewriting systems both can be adapted to the case where critical pairs are allowed, but only if they are of the form (s, s) . Such a critical pair is said to be *trivial*. The notion of trivial critical pair is used to define the class of weakly orthogonal higher-order rewriting systems; the definition is analogous to the one for the first-order case.

Definition 3. *A higher-order rewriting system is weakly orthogonal if it is left-linear and all its critical pairs are trivial.*

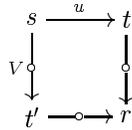
Examples of weakly orthogonal rewriting systems that are not orthogonal are $\{a \rightarrow b, f(a) \rightarrow f(b)\}$ and $\{f(x) \rightarrow f(b), f(a) \rightarrow f(b)\}$. Moreover, lambda-calculus with both β -and η -reduction is a weakly orthogonal rewriting system.

Orthogonality is defined in terms of the left-hand sides of the rewrite rules only, whereas weak orthogonality concerns the right-hand sides of rewrite rules. This explains why extensions from the orthogonal to the weakly orthogonal case may cause considerable complications, also for first-order rewriting.

Simultaneous Reduction. The method using simultaneous reduction can be extended to the weakly orthogonal case as follows. The idea is to proceed by induction on a measure that roughly speaking counts the overlap between two co-initial steps $s \rightarrow t$ and $s \rightarrow t'$. In case there is no overlap, we proceed as in the orthogonal case. In case there is overlap, we take minimal overlapping redexes u and u' such that u is contracted in $s \rightarrow t$ and u' is contracted in $s \rightarrow t'$. Since all trivial pairs are trivial, we have that contracting u yields the same term as contracting u' , say s' . The proof proceeds now by showing that $s' \rightarrow t$ and $s' \rightarrow t'$, and that the measure of this divergence is smaller than that of the original one. It follows then by induction that \rightarrow satisfies the diamond property. The proof is illustrated by the following diagram.



Finite Developments. The proof using developments is adapted to the case of weakly orthogonal systems as follows [25]. First, complete developments are defined for sets of redexes that are pairwise non-overlapping. It turns out that this is sufficient to ensure that complete developments can be defined properly; a global condition on the whole set of redexes under consideration is not necessary. The aim is to show that whenever $s \rightarrow t$, by contracting a redex u , and $s \rightarrow t'$, by contracting a set of redexes V we have that there exists a term r such that $t \rightarrow r$ and $t' \rightarrow r$:



There are two possibilities. Either the redex u doesn't overlap with any redex in the set V . In that case both t and t' rewrite by a complete development to the term obtained by performing a complete development of $\{u\} \cup V$, starting in s . In the other case, there is a redex v in the set V such that u and v are overlapping, and hence give rise to a critical pair. Since all critical pairs are trivial, we have that s also rewrites to t by contracting the redex v instead of u . Now we have that t rewrites to t' by performing a complete development of all residuals of V in t .

3.3 Critical Pairs

In the previous subsection we have seen that for left-linear systems, confluence holds if all critical pairs are trivial. Further, we have seen examples showing that a term rewriting system that is not left-linear need not be confluent, even in the absence of critical pairs. A natural next step is to investigate to what extent the condition on critical pairs can be relaxed while maintaining confluence, or a weaker version of it.

Huet [11] (see also [17]) shows that a first-order term rewriting system is locally confluent (that is, $t' \rightarrow r \leftarrow t$ whenever $t' \leftarrow s \rightarrow t$) if all its critical pairs are confluent. A critical pair (s, s') is said to be confluent if s and s' have a common reduct. Using Newman's Lemma, confluence of a terminating first-order term rewriting system follows from confluence of its critical pairs.

The idea of the proof as given in [11] is globally as follows. Consider two co-initial rewrite steps $s \rightarrow t$, obtained by contraction of a redex u , and $s \rightarrow t'$, obtained by contraction of a redex u' . If u and u' do not overlap, an analysis of their relative positions yields that a common reduct of t and t' can be found. Now suppose that u and u' are overlapping redexes and let u' be above u . The key auxiliary result states that in that case there must be a critical pair (r, r') such that $t = C[r\sigma]$ and $t' = C[r'\sigma]$. Since critical pairs are confluent, r and r' have a common reduct. As a consequence, t and t' have a common reduct.

The result of [11] is generalised to higher-order term rewriting systems by Nipkow [22, 18]. The proof proceeds basically as in the first-order case, but it is technically significantly more difficult to show the key auxiliary result.

Theorem 3. *A higher-order rewriting system is locally confluent if all its critical pairs are confluent.*

Consider for example the rewrite rules for surjective pairing: $\{\pi_1(\pi(z, z')) \rightarrow z, \pi_2(\pi(z, z')) \rightarrow z', \pi(\pi_1(z), \pi_2(z)) \rightarrow z\}$. There are two critical pairs, namely $(\pi(z, \pi_2(\pi(z, z'))), \pi(z, z'))$ and $(\pi(\pi_1(\pi(z, z')), z'), \pi(z, z'))$. Both are confluent. As a consequence, the rewriting system is locally confluent. Note that also local confluence of lambda-calculus with β -reduction extended with the rules for surjective pairing is obtained as an application of the theorem above.

3.4 Development Closed

For left-linear rewriting systems that are not terminating, confluence of the critical pairs does not guarantee confluence of the rewriting system. This is for instance illustrated by the rewriting system $\{a \rightarrow b, a \rightarrow c, b \rightarrow a, b \rightarrow d\}$.

Huet [11] formulates a criterion on critical pairs that is stronger than the one of being only confluent, and shows that it is a sufficient condition for confluence of left-linear first-order term rewriting systems. This criterion is as follows: if (s, t) is a critical pair, then we have $s \dashv\vdash t$. A critical pair satisfying this criterion is said to be *parallel closed*, and a rewriting system is said to be parallel closed if all its critical pairs are so. This result yields for instance confluence of the term

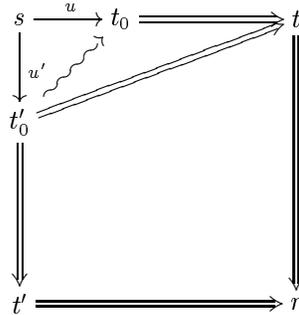
rewriting system $\{f(g(x), b) \rightarrow f(g(x), b'), g(a') \rightarrow g(a), a \rightarrow a', b \rightarrow b'\}$, because for the critical pair $(f(g(a), b), f(g(a'), b'))$ we have $f(g(a), b) \dashrightarrow f(g(a'), b')$.

Van Oostrom extends in [26] the result that parallel closed rewriting systems are confluent to the higher-order case. The skeleton of the proof is the same as that for first-order rewriting, but the difference is that now another invariant is needed: instead of parallel reduction (\dashrightarrow) the relation of complete developments (\dashrightarrow) is used. Correspondingly, the requirement that all critical pairs are parallel closed can be relaxed: instead, for every critical pair (s, t) it is required that $s \dashrightarrow t$, that is, there is a complete development from s to t . A critical pair satisfying this criterion is said to be *development closed*, and a rewriting system is said to be development closed if all its critical pairs are so. The result of [26] states that a left-linear higher-order rewriting system is confluent if all its critical pairs are development closed; this is an extension of the result that parallel closed rewriting systems are confluent also if only first-order rewriting is considered.

The skeleton of the proof is as follows. We look for a rewriting relation \Rightarrow such that the diamond property of \Rightarrow implies confluence of \rightarrow . For proving the diamond property of \Rightarrow , suppose that $s \Rightarrow t$ and $s \Rightarrow t'$. We proceed by induction on some measure that roughly speaking counts the overlap between those two steps. So first we need to prove the diamond property of \Rightarrow in the case that the measure is zero, that is, there is no overlap between the steps $s \Rightarrow t$ and $s \Rightarrow t'$. In case the measure is greater than zero, we single out a critical divergence between the step $s \rightarrow t_0$, obtained by contracting some redex u , and the step $s \rightarrow t'_0$, obtained by contracting some redex u' . Suppose that the condition imposed on critical pairs yields that $t'_0 \rightsquigarrow t_0$. Now the following is shown:

- We have $t_0 \Rightarrow t$ and $t'_0 \Rightarrow t'$.
- The steps $t'_0 \rightsquigarrow t_0$ and $t_0 \Rightarrow t$ can be joined to form a step $t'_0 \Rightarrow t$.
- The new divergence $t' \leftarrow t'_0 \Rightarrow t$ is smaller than the old one $t' \leftarrow s \Rightarrow t$.

It then follows then \Rightarrow has the diamond property, which yields confluence of \rightarrow . The proof idea is illustrated in the following diagram.



For first-order rewriting [11], parallel reduction (\dashrightarrow) is taken for \Rightarrow . For higher-order rewriting this doesn't work since it doesn't satisfy the diamond property. Instead, the complete development relation (\dashrightarrow) is taken for \Rightarrow [26].

Another question is what should be taken for \rightsquigarrow . It is needed that $\rightsquigarrow \subseteq \Rightarrow$, and the result is the strongest if $\rightsquigarrow = \Rightarrow$. The latter is indeed done in both cases.

It is more difficult to show that the \rightarrow -step joining the critical pair and the remainder of the original \rightarrow -step can be combined into a new \rightarrow -step than to show the analogous statement for the parallel rewriting relation \twoheadrightarrow . Also the argument that the measure decreases is more complex in the higher-order than in the first-order case.

Theorem 4. *A left-linear higher-order rewriting system is confluent if it is development closed.*

As an application of this result, we obtain that the higher-order term rewriting system $\{f(g(x,z(x))) \rightarrow f(z(z(a))), g(x,h'(x)) \rightarrow h(h(a)), h(x) \rightarrow h'(x)\}$ is confluent, because for the critical pair $(f(h(h(a))), f(h'(h'(a))))$ we have that $f(h(h(a))) \rightarrow f(h'(h'(a)))$.

4 Termination

A rewriting system is *terminating* if it does not admit infinite rewrite sequences; then all rewrite sequences end in a normal form. For first-order rewriting, a lot of techniques and methods to prove termination exist. The theory of termination of higher-order rewriting is so far significantly less well-developed. In this section we discuss two important methods to prove termination that are extended to the higher-order case. The last subsection is concerned with a normalising strategy.

4.1 Termination Models

Termination of a rewriting system can be proved by mapping every term to an element of a non-empty set A that is equipped with a well-founded partial order $>$, in such a way that $\llbracket s \rrbracket > \llbracket t \rrbracket$ whenever $s \rightarrow t$, where the mapping is denoted by $\llbracket \cdot \rrbracket$. For first-order term rewriting, this means that we need to have $\llbracket C[l\sigma] \rrbracket > \llbracket C[r\sigma] \rrbracket$ for every rewrite rule $l \rightarrow r$, context C , and assignment σ . It is clearly desirable to eliminate some of the quantifications here.

This is done in the method using termination models: here a requirement concerning the rewrite rules guarantees the inequality $\llbracket C[l\sigma] \rrbracket > \llbracket C[r\sigma] \rrbracket$ to hold for every context C and every assignment σ . A *termination model* for a first-order term rewriting system is an algebra for the signature of the rewriting system in which the terms can be interpreted as usual, with the following properties:

1. The algebra is equipped with a well-founded partial order $>$.
2. The functions $f : A^n \rightarrow A$ of the algebra are required to be *monotonic* in the following sense: $f(\dots, a, \dots) > f(\dots, a', \dots)$ whenever $a > a'$.
3. For every rewrite rule $l \rightarrow r$ and assignment σ we have $\llbracket l\sigma \rrbracket > \llbracket r\sigma \rrbracket$.

In fact, instead of the last requirement a condition on the algebraic side is formulated in terms of valuations; that condition implies the last requirement as above. The second requirement yields that $\llbracket C[s] \rrbracket > \llbracket C[t] \rrbracket$ whenever $\llbracket s \rrbracket > \llbracket t \rrbracket$.

Huet and Oppen [12] and Zantema [37] prove that a first-order term rewriting system is terminating if and only if it has a termination model. This result provides a complete but not algorithmic method for proving termination.

Van De Pol presents in [28, 29] a generalisation of this result to the higher-order case. This generalisation makes use of ideas that are also present in the work by Gandy [7] and De Vrijer [35, 36] on termination of simply typed λ -calculus. For every type A , the set of *functionals* of type A is defined by induction on the definition of simple types: a base type is interpreted as some fixed non-empty set, and an arrow type $A \rightarrow B$ is interpreted as the function space $\llbracket A \rrbracket \Rightarrow \llbracket B \rrbracket$, where we use the notation $\llbracket \cdot \rrbracket$ also for the interpretation of types.

In order to prove termination of simply typed λ -calculus, the λI -terms (λ -terms that cannot erase arguments, for instance $\lambda x.y$ is not a λI -term) are interpreted as certain functionals, called the hereditarily monotonic ones, which come equipped with an ordering. This interpretation cannot be used for proving termination of a higher-order rewriting system for the following reason. First, contraction of a β -redex yields a decrease in the ordering, whereas in higher-order rewriting we work modulo β and hence contraction of a β -redex shouldn't change the interpretation. Second, for proving termination of higher-order rewriting, the ordering should be adapted to make it closed under contexts in $\beta\bar{\eta}$ -normal form. These two requirements together yield that the ordering must be reflexive, which makes it useless for proving termination.

The solution presented in [28, 29] is to define two sets of functionals: the set of *weakly monotonic functionals*, and the set of *strict functionals*. Both come equipped with an order. The first is a superset of the set of hereditarily monotonic functionals and can be used for the interpretation of arbitrary λ -terms, the second is a subset of the set of hereditarily monotonic functionals and is used for the interpretation of the function symbols and variables of a higher-order rewriting system. Now all these ingredients are present in the definition of a *termination model* for a higher-order rewriting system, which is given as follows:

1. Every base type is interpreted as a well-founded partial order with some additional property that guarantees the existence of weakly monotonic and strict functionals for all types.
2. Function symbols of the higher-order rewriting system are interpreted as strict functionals.
3. For every rewrite rule $l \rightarrow r$, we have $\llbracket l \rrbracket > \llbracket r \rrbracket$ in the weakly monotonic ordering.

The two key results for the termination method are now as follows. First, whenever $\llbracket s \rrbracket > \llbracket t \rrbracket$ in the weakly monotonic ordering, we have in the strict ordering that $\llbracket C[s] \rrbracket > \llbracket C[t] \rrbracket$ with C a context in $\beta\bar{\eta}$ -normal form, provided that function symbols and variables are interpreted strictly. Second, it is shown that $\llbracket s \rrbracket = \llbracket t \rrbracket$ if s and t are β -equal. This is used to prove the following [28, 29].

Theorem 5. *A higher-order rewriting system is terminating if it has a termination model.*

The reverse of the statement does not hold; a simplification of a counterexample in [14] is given in [29]. The restriction that left-hand sides of the rewrite rules should be patterns is not necessary for the proof of the theorem. Further, a corollary is that the combination of a terminating first-order term rewriting system and simply typed λ -calculus with β -reduction is terminating.

To illustrate the use of termination models, we consider as an example (taken from [29]) the higher-order rewriting system $\{\text{and}(z, \text{forall}(x.z'(x))) \rightarrow \text{forall}(x.\text{and}(z, z'(x)))\}$ with $\text{and} : \text{form} \rightarrow \text{form} \rightarrow \text{form}$ and $\text{forall} : (\text{term} \rightarrow \text{form}) \rightarrow \text{form}$. Intuitively, term represents the set of terms and form represents the set of formulas. Both are interpreted as the set of natural numbers. Further,

- and is interpreted as $\lambda m, n. \in \mathbb{N}. (2 \cdot m + 2 \cdot n)$,
- forall is interpreted as $\lambda f \in \mathbb{N} \Rightarrow \mathbb{N}. (f(0) + 1)$.

Now we have:

$$\begin{aligned} \llbracket \text{and}(z, \text{forall}(x.z'(x))) \rrbracket &= 2 \cdot \llbracket z \rrbracket + 2 \cdot (\llbracket z' \rrbracket(0) + 1) \\ &> 2 \cdot \llbracket z \rrbracket + 2 \cdot \llbracket z' \rrbracket(0) + 1 \\ &= \llbracket \text{forall}(x.\text{and}(z, z'(x))) \rrbracket. \end{aligned}$$

4.2 Recursive Path Ordering

An important method to prove termination of a first-order term rewriting system is the one using the recursive path ordering [6]. This ordering is roughly speaking defined by extending a well-founded ordering on function symbols in a recursive way. The two key results concerning the recursive path ordering are that it is a well-founded ordering on terms, and that $l > r$ implies $C[l\sigma] > C[r\sigma]$ for every context C and every assignment σ . This yields the following method to prove termination of a first-order term rewriting system: define a well-founded ordering on the function symbols and show that in its extension to the recursive path ordering we have $l > r$ for every rewrite rule.

Jouannaud and Rubio define in a recent paper [13] a generalisation of the recursive path ordering to higher-order terms, and show that it provides a method to prove termination of higher-order term rewriting systems. The remarkable feature of their proof is that Kruskal's Tree Theorem is not used to show well-foundedness of the recursive path ordering.

To start with, we show that termination of the first-order term rewriting system $\{f(z, g(z')) \rightarrow g(f(z, z'))\}$ can be proved using the recursive path ordering. Taking $f > g$ as ordering on the function symbols yields that $f(z, g(z')) > g(f(z, z'))$ in the recursive path ordering: first, one is allowed to make copies of $f(z, g(z'))$ under a smaller function symbol (g), yielding $g(f(z, g(z')))$, and second, it is possible to get rid of the innermost symbol g by selecting only its argument z' , yielding $f(z, g(z')) > g(f(z, z'))$.

Next, we reconsider the higher-order term rewriting system of the previous subsection $\{\text{and}(z, \text{forall}(x.z'(x))) \rightarrow \text{forall}(x.\text{and}(z, z'(x)))\}$. The structure of this rewrite rule is similar to the one of the first-order example above. Taking as ordering on the function symbols $\text{and} > \text{forall}$ yields that $\text{and}(z, \text{forall}(x.z'(x))) > \text{forall}(x.\text{and}(z, z'(x)))$ in the recursive path ordering defined in [13].

4.3 Outermost-Fair Rewriting

It may occur that a term can be rewritten to normal form but is also the starting point of an infinite rewrite sequence: consider for example the term $f(a, b)$ in the rewriting system $\{f(a, x) \rightarrow c, b \rightarrow b\}$. In such a situation it is important to know how to rewrite the term such that eventually a normal form is obtained. In the case of $f(a, b)$, at some point the redex $f(a, b)$ should be contracted in order to reach the normal form c . A *strategy* can be seen as selecting one or more redex occurrences in a term that is not in normal form. A strategy is said to be *normalising* if repeatedly contracting redex occurrences selected by the strategy yields a normal form whenever the initial term has one.

O'Donnell shows in [24] that the *parallel-outermost* strategy, which contracts all redexes that are outermost in a term in one step, is normalising for first-order term rewriting systems that are left-linear, and where trivial critical pairs are only allowed in a certain restricted form. A stronger result obtained in [24] is that *outermost-fair* rewriting is normalising. A rewrite sequence is said to be outermost-fair if every outermost redex occurrence is eliminated eventually. For example, in the rewriting system $\{a \rightarrow a, b \rightarrow c, f(c, x) \rightarrow f(b, x)\}$, the rewrite sequence $f(b, a) \rightarrow f(c, a) \rightarrow f(b, a) \rightarrow \dots$ is outermost-fair, but the rewrite sequence $f(b, a) \rightarrow f(b, a) \rightarrow f(b, a) \rightarrow \dots$ is not. A parallel-outermost rewrite sequence is outermost-fair, since in every step all outermost redexes are eliminated. Hence normalisation of the parallel-outermost strategy is a direct consequence of the normalisation of outermost-fair rewriting.

The idea of the proof that outermost-fair rewriting is normalising as given in [24] is roughly as follows. Consider a term s that has a normal form s' . The intention is to show that all outermost-fair rewrite sequences starting in s end in s' . We fix a rewrite sequence $f : s \rightarrow s'$ and proceed by induction on its length. If f consists of zero steps, there is nothing to prove. If f consists of one or more rewrite steps, then it is of the form $s \rightarrow t \rightarrow s'$. Let d be an outermost-fair rewrite sequence starting in s . Now the rewrite sequence d is projected over the rewrite step $s \rightarrow t$, as in the following picture, where $s_0 = s$ and $t_0 = t$:

$$\begin{array}{c}
 d : \quad s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \dots \\
 \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \\
 e : \quad t_0 \longrightarrow t_1 \longrightarrow t_2 \longrightarrow t_3 \longrightarrow \dots
 \end{array}$$

Then the following is shown:

1. If d is outermost-fair then e is outermost-fair.
2. If d is outermost-fair and e ends in a normal form r , then d also ends in r .

It now follows by the induction hypothesis that the outermost-fair rewrite sequence d ends in the normal form s' of s .

We don't consider the proof in more detail here, but just discuss the restriction of left-linearity and the one imposed on critical pairs.

An important observation is that outermost-fair rewriting is not normalising if a redex that is not outermost can create an outermost redex. This can happen if the rewriting system is not left-linear. Consider for instance $\{a \rightarrow b, f(x, x) \rightarrow b, g(x) \rightarrow g(x)\}$. The term $f(g(a), g(b))$ has a normal form, namely b , but it is not reached by the outermost-fair rewrite sequence $f(g(a), g(b)) \rightarrow f(g(a), g(b)) \rightarrow f(g(a), g(b)) \rightarrow \dots$ in which alternately the redexes $g(a)$ and $g(b)$ are contracted. In a rewrite sequence from $f(g(a), g(b))$ to normal form, as for instance $f(g(a), g(b)) \rightarrow f(g(b), g(b)) \rightarrow b$, the contraction of the redex a , which is not outermost, creates the outermost redex $f(g(b), g(b))$.

Further, outermost-fair rewriting is not normalising if redex patterns interfere in an arbitrary way. The term a in the rewriting system $\{a \rightarrow a, a \rightarrow b\}$ has a normal form which is not reached by the outermost-fair rewrite sequence $a \rightarrow a \rightarrow a \rightarrow \dots$. This interference is ruled out by forbidding the presence of critical pairs. However, it doesn't show that it is necessary to restrict attention to systems without critical pairs. Actually, the result in [24] is obtained for rewriting systems where a special kind of critical pairs is allowed, namely the ones that are trivial and where the overlap is at the top. Rewriting systems that are left-linear and have only this kind of trivial pair, caused by overlap at the top, are said to be *almost orthogonal*. Almost orthogonal systems form a superclass of the orthogonal systems, and a subclass of the weakly orthogonal systems. The rewriting system $\{f(a) \rightarrow f(a), f(x) \rightarrow f(a)\}$ is almost orthogonal but not orthogonal. Note that its trivial critical pair is caused by an overlap at the top. The critical pair in $\{a \rightarrow a, f(a) \rightarrow f(a)\}$ is trivial but the overlap is not at the top; this system is weakly orthogonal but not almost orthogonal.

The above discussion shows that it is necessary to restrict attention to left-linear systems (otherwise outermost-fair rewriting need not be normalising). It is however not yet clear whether the restriction to critical pairs that are trivial and have overlap at the top is necessary. We come back to this point below.

The question arises whether the result of [24] can be extended to the higher-order case. It turns out that this is indeed the case, provided an additional restriction on the rewriting systems is imposed. The point is that in higher-order rewriting, there is, due to the presence of bound variables, yet another way in which a redex that is not outermost can create an outermost redex. As a consequence, outermost-fair rewriting need not be normalising if this additional restriction is not imposed. The following example, due to Van Oostrom, shows the essence of the problem: $\{f(x.z) \rightarrow a, g(z) \rightarrow g(z), h(z) \rightarrow a\}$. The term $f(x.g(h(x)))$ has a normal form: $f(x.g(h(x))) \rightarrow f(x.g(a)) \rightarrow a$. However, this normal form is not reached by the outermost-fair rewrite sequence $f(x.g(h(x))) \rightarrow f(x.g(h(x))) \rightarrow f(x.g(h(x))) \rightarrow \dots$. Note that contraction of the innermost redex $h(x)$ creates an outermost redex, namely $f(x.g(a))$, by erasing the bound variable; it is only possible to apply the first rewrite rule to a term of the form $f(x.t)$ if t doesn't contain free occurrences of x . This way of creating redexes is ruled out by disallowing the non-occur check as present in the example. Formally, this is done by requiring the rewrite rules to be *fully extended*, a notion which is defined as follows [9].

Definition 4.

1. Let $z.l$ be a rule-pattern. An occurrence of $z \in z$ in l is fully extended if it is of the form $z(x_1, \dots, x_n)$ with x_1, \dots, x_n all the variables bound above it.
2. A rewrite rule $z.l \rightarrow z.r$ is fully extended if every occurrence of every $z_i \in z$ in l is fully extended.
3. A higher-order rewriting system is fully extended if all its rewrite rules are.

The rewrite rule $z.f(x.z(x)) \rightarrow r$ is fully extended. The rewrite rules $z.f(x.z) \rightarrow r$ and $\text{abs}(x.\text{app}(z, x)) \rightarrow z$ are not fully extended. Recall that $z.f(x.z(x, y))$ and $z.f(x.z(x, x))$ are not even rule-patterns.

Now we come back to the restriction imposed on critical pairs. Recently, it is shown in [27] that outermost-fair rewriting is normalising for *weakly* orthogonal rewriting systems, where arbitrary trivial pairs are allowed. It seems that the restriction to trivial critical pairs cannot be relaxed much more. Consider for instance the rewriting system $\{a \rightarrow b, g(x) \rightarrow g(x), f(g(b)) \rightarrow b\}$. It is parallel closed since we have $f(g(b)) \rightarrow b$. The term $f(g(a))$ has a normal form, namely b , but it is not reached by the outermost-fair rewrite sequence $f(g(a)) \rightarrow f(g(a)) \rightarrow f(g(a)) \rightarrow \dots$. The proof presented in [27] is abstract in nature and applies also to the case of higher-order rewriting. It makes use of ideas that are also present in [34, 20, 8]. For proofs according to the sketch given above, see [24, 4, 32].

Theorem 6. *Outermost-fair rewriting is normalising for higher-order rewriting systems that are weakly orthogonal and fully extended.*

References

1. P. Aczel. A general Church-Rosser theorem. University of Manchester, July 1978.
2. A. Asperti and C. Laneve. Interaction systems I: The theory of optimal reductions. *Mathematical Structures in Computer Science*, 4:457–504, 1994.
3. 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.
4. J.A. Bergstra and J.W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.
5. A. Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936.
6. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.
7. 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.
8. J. 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)*, volume 1059 of *Lecture Notes in Computer Science*, pages 180–195, April 1996.

9. M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA '96)*, volume 1103 of *Lecture Notes in Computer Science*, pages 138–152, New Brunswick, USA, 1996.
10. R. Hindley. Reductions of residuals are finite. *Transactions of the American Mathematical Society*, 240:345–361, June 1978.
11. G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, October 1980.
12. G. Huet and D.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.
13. J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *Proceedings of the 14th annual IEEE Symposium on Logic in Computer Science (LICS '99)*, Trento, Italy, 1999. To appear.
14. S. Kahrs. Termination proofs in an abstract setting. To appear in *Mathematical Structures in Computer Science*.
15. Z.O. Khasidashvili. Expression Reduction Systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220, Tbilisi, 1990.
16. J.W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, Amsterdam, 1980. PhD Thesis.
17. D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
18. R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.
19. P.-A. Melliès. *Description Abstraite des Systèmes de Réécriture*. PhD thesis, Université de Paris VII, Paris, France, 1996.
20. A. Middeldorp. Call by need computations to root-stable form. In *Proceedings of the 24th Symposium on Principles of Programming Languages (POPL '97)*, pages 94–105, Paris, France, January 1997. ACM Press.
21. D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.
22. T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, Amsterdam, The Netherlands, July 1991.
23. T. Nipkow and C. Prehofer. Higher-order rewriting and equational reasoning. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations*, volume 8 of *Applied Logic Series*, pages 399–430. Kluwer, 1998.
24. M.J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer Verlag, 1977.
25. V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, March 1994.
26. V. van Oostrom. Developing developments. *Theoretical Computer Science*, 175(1):159–181, 1997.
27. V. van Oostrom. Normalisation in weakly orthogonal rewriting. In *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (RTA '99)*, 1999.

28. 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.
29. J.C. van de Pol. *Termination of Higher-order Rewrite Systems*. PhD thesis, Utrecht University, Utrecht, The Netherlands, December 1996.
30. C. Prehofer. *Solving Higher-Order Equations: From Logic to Programming*. PhD thesis, Technische Universität München, 1995.
31. Z. Qian. Unification of higher-order patterns in linear time and space. *Journal of Logic and Computation*, 6:315–341, 1996.
32. F. van Raamsdonk. *Confluence and Normalisation for Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, May 1996.
33. B.K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the Association for Computing Machinery*, 20(1):160–187, January 1973.
34. R.C. Sekar and I.V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. In *Proceedings of the 7th International Symposium on Logic in Computer Science (LICS '90)*, pages 230–241, 1990.
35. R. de Vrijer. Exactly estimating functionals and strong normalization. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen*, 90(4), December 1987.
36. R. de Vrijer. *Surjective Pairing and Strong Normalization: Two Themes in Lambda Calculus*. PhD thesis, Universiteit van Amsterdam, January 1987.
37. H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.