# Comparing Combinatory Reduction Systems and Higher-Order Rewrite Systems

Vincent van Oostrom* and Femke van Raamsdonk**

* *Department of Mathematics and Computer Science, Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam*

** *CWI, P.O. Box 4079, 1009 AB Amsterdam*

**Abstract**

In this paper two formats of higher-order rewriting are compared: Combinatory Reduction Systems introduced by Klop [Klo80] and Higher-order Rewrite Systems defined by Nipkow [Nipa]. Although it always has been obvious that both formats are closely related to each other, up to now the exact relationship between them has not been clear. This was an unsatisfying situation since it meant that proofs for much related frameworks were given twice. We present two translations, one from Combinatory Reduction Systems into Higher-Order Rewrite Systems and one vice versa, based on a detailed comparison of both formats. Since the translations are very 'neat' in the sense that the rewrite relation is preserved and (almost) reflected, we can conclude that as far as rewrite theory is concerned, Combinatory Reduction Systems and Higher-Order Rewrite Systems are equivalent, the only difference being that Combinatory Reduction Systems employ a more 'lazy' evaluation strategy. Moreover, due to this result it is the case that some syntactic properties derived for the one class also hold for the other.

*AMS Subject Classification (1991):* 68Q42
*CR Subject Classification (1991):* F1.1, F4.1, F4.2
*Keywords & Phrases:* higher-order rewriting, $\lambda$-calculus, developments, confluence
*Note:* The research of the second author is supported by NWO/SION project 612-316-606

## 1. Introduction

This paper is concerned with a comparison of two formats of higher-order rewriting: Combinatory Reduction Systems (CRSs) as introduced by Klop [Klo80] and Higher-order Rewrite Systems (HRSs) as introduced by Nipkow [Nipa].

Inspired by Aczel [Acz78], Klop defined CRSs in [Klo80] as first-order term rewriting systems possibly with bound variables, so as to include both first-order rewrite systems such as Curry's Combinatory Logic and rewrite systems with bound variables such as Church's $\lambda$-calculus. The point was that a large amount of syntactic rewrite theory could be developed for this framework.

In [Nipa], Nipkow introduces HRSs as a generalisation of first-order rewrite systems to terms with higher-order functions and bound variables. Furthermore, HRSs were designed to have the same logical basis as systems like Isabelle [Pau90] and $\lambda$Prolog [NM88]. That is, a typed $\lambda$-calculus is used as a meta-language.

These different objectives have led to surprisingly large differences in the presentation of these systems. For CRSs the meta-language, i.e. the language in which the notions of term, substitution and rewrite step are expressed, is left implicit in the presentation. For HRSs the meta-language is Church's $\lambda^\tau$-calculus of simply typed $\lambda$-terms with $\beta$ as rewrite rule. In the case of CRSs, the introduction of a special purpose meta-language makes the definition quite involved. This disadvantage can be taken away by noting that the meta-language is in fact (a polyadic version of) the $\underline{\lambda}$-calculus, of underlined $\lambda$-terms with $\underline{\beta}$ as rewrite rule from [Klo80,

Sec. I.3.5] (cf. also indexed $\lambda$-calculus of [Bar84, §11.1.3]), i.e. the language of *developments* (or `let`-expressions).

Once we have made the meta-language of CRSs explicit, we can compare both formats by comparing their respective meta-languages. Comparing is done by giving encodings of one system into the other and vice versa. The encoding of CRSs into HRSs is straightforward because $\underline{\lambda}$-calculus can be encoded into $\lambda^\tau$-calculus. The encoding of HRSs into CRSs is somewhat more involved; $\lambda^\tau$-calculus cannot be encoded directly into $\underline{\lambda}$-calculus. For example, the latter does enjoy the *disjointness property* (rewriting preserves disjointness, cf. [Klo80, pg. 38]), while the former doesn't. In general, in $\lambda^\tau$-calculus rewrite sequences can be longer than in $\underline{\lambda}$-calculus. Our solution is to add an explicit $\beta$-rule (and the corresponding symbols) to the encoding of an HRS. A rewrite step in the HRS is then simulated by a rewrite step in the CRS followed by an explicit $\beta$-reduction to normal form. More precisely, let $\mathcal{C}$ be a CRS and $\mathcal{H}$ a HRS. We write $\rightarrow_{\mathcal{C}}$ and $\rightarrow_{\mathcal{H}}$ respectively for their rewrite relations. Translating is denoted by $\langle\_\rangle$, and reduction to normal form with respect to the explicit $\beta$-rule is written as $\rightarrow_\beta^!$. Then we have

$$\begin{aligned} \langle\rightarrow_{\mathcal{C}}\rangle &= \rightarrow_{\langle\mathcal{C}\rangle} \\ \langle\rightarrow_{\mathcal{H}}\rangle &= \rightarrow_{\langle\mathcal{H}\rangle} \cdot \xrightarrow[\beta]{!} \end{aligned}$$

if the relations are restricted to the set of translated terms.

How natural an encoding is, can be measured by the properties it preserves and reflects. Our encoding of CRSs into HRSs both preserves and reflects the main property of rewrite systems, i.e. whether one term rewrites (in one step) to another. This allows for a confluence proof for orthogonal CRSs via a proof of confluence for orthogonal HRSs. As noted above the translation the other way around is not that nice. The HRS is simulated by a more refined CRS; 'giant' HRS-steps are simulated by many 'small' CRS-steps. This is analogous to the way in which $\lambda$-calculus is simulated by the $\lambda\sigma$-calculus defined in [ACCL90]. Of course, not every step in the refined system is reflected in the original HRS, but still we can say something: every rewrite sequence between encodings of HRS-terms is reflected in the original HRS. Again, this allows for a confluence proof for orthogonal HRSs via a proof of confluence for orthogonal CRSs. For the moment being, we have only considered use of our translation for confluence results.

Our comparison only considers CRSs versus HRSs. There are some more alternatives such as Khasidashvili's Expression Reduction Systems [Kha90] and Takahashi's Conditional Lambda Calculi [Tak]. We claim that the main differences between these and CRSs (or HRSs) are of a syntactic nature. However, an exact comparison is left to future work.

The paper is organised as follows. In section 2 we will discuss in detail the difference between CRSs and HRSs by first considering only terms and next also the rewrite relation on terms. In section 3 we define a translation from CRSs into HRSs and, using this translation, we give a confluence proof for orthogonal CRSs. The translation from HRSs into CRSs is presented in section 4, again the translation is used to give a confluence proof, now for orthogonal HRSs. Section 5 concludes the paper with some discussion on higher-order rewriting. The reader is assumed to be familiar with term rewriting and (simply typed) $\lambda$-calculus.

NOTATION. We adhere mostly to the notations introduced by Klop for CRSs, and Nipkow for HRSs. Since their introduction both formats have been subject to some change and we will use their most recent presentations, viz. [KOR93] for CRSs and [Nipb] for HRSs. The most notable change is the use of the functional format for CRSs instead of the applicative one of [Klo80]. The reason for choosing the functional format is that it is closer to the usual notation for term

rewriting systems. Moreover, in applicative CRSs the object-language application symbol is left implicit in the notation, while for HRSs the meta-language application symbol is left implicit, which would possibly give rise to confusion in comparing these formats.

## 2. Comparing the Syntax

Since some important differences between CRSs and HRSs are already manifest if only the way terms are built is considered, we first restrict attention to term formation. Next, we fix attention to rule formation and finally the generation of the rewrite relation is considered.

### *2.1 Term Formation*

*2.1.1 CRS Terms* A CRS $\mathcal{C}$ is a pair $(\mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ is its alphabet of symbols and $\mathcal{R}$ its set of *rewrite* or *reduction* rules. (Because of the termination connotation of the word 'reduction' we will use it only in the case of normalising rewrites.) In a CRS a distinction is made between metaterms and terms. The left- and right-hand side of a rule are metaterms, but the rewrite relation is a relation on terms.

The alphabet $\mathcal{A}$ of a CRS $(\mathcal{A}, \mathcal{R})$ consists of

- symbols for variables $x\ y\ z \ldots$,
- the abstraction operator written as $[\_]\_$,
- symbols for operators with a fixed arity $F\ G\ H\ \ldots$,
- symbols for metavariables with a fixed arity $Z\ Z_0\ Z_1\ \ldots$.

The set $\mathsf{MTerms}$ of metaterms is the least set such that
(1) $x \in \mathsf{MTerms}$ for every variable $x$,
(2) $[x]t \in \mathsf{MTerms}$ for a variable $x$ and $t \in \mathsf{MTerms}$,
(3) $F(t_1, \ldots, t_n) \in \mathsf{MTerms}$ if $t_1, \ldots, t_n \in \mathsf{MTerms}$ and $F$ is an $n$-ary operator,
(4) $Z(t_1, \ldots, t_n) \in \mathsf{MTerms}$ if $t_1, \ldots, t_n \in \mathsf{MTerms}$ and $Z$ is an $n$-ary metavariable.

The set $\mathsf{Terms}$ of terms consists of all metaterms without metavariables. In a term or metaterm of the form $[x]t$, we call $t$ the *scope* of $[x]$. A variable $x$ occurs *free* in a term or metaterm if it is not in the scope of an occurrence of $[x]$. A variable x occurs *bound* otherwise. A term is called *closed* if all variables occur bound. Only variables (and no metavariables) can be bound by the abstraction operator. We will sometimes write $[x_1 \ldots x_n]t$ for $[x_1] \ldots [x_n]t$.

Let $\Box$ be a fresh symbol. A *context* is a term with one or more occurrences of $\Box$. A context with exactly one occurrence of $\Box$ is written as $C[\ ]$, and one with $n$ occurrences of $\Box$ as $C[, \ldots, ]$. If $C[, \ldots, ]$ is a context with $n$ occurrences of $\Box$ and $t_1, \ldots, t_n$ are terms, then $C[t_1, \ldots, t_n]$ denotes the result of replacing from left to right the occurrences of $\Box$ by $t_1, \ldots, t_n$.

Example 2.1 The alphabet of $\lambda$-calculus in CRS format consists of two operators: a unary operator $\lambda$ for $\lambda$-abstraction and a binary operator @ for application. Examples of some $\lambda$-terms written in CRS notation:

$\lambda([x]x)$ for $\lambda x.x$,

$@(x, y)$ for $xy$,

$\lambda([x]@(y, x))$ for $\lambda x.yx$, and

$@(\lambda([x]x), y)$ for $(\lambda x.x)y$.

Because of the very liberal term formation in the CRS framework, many terms can be formed from the alphabet consisting of $\lambda$ and @ that do not correspond to any $\lambda$-term.

In the light of this example, it is clear that the notion of CRS needs to be extended to systems with a restricted set of terms. Of course, such a restricted set of terms has to be closed under rewriting. For example, $\lambda$-calculus can be written as a CRS without 'junk', by appropriate restrictions on term formation. If one wants to stress the point that only a subset of the set of terms is considered, one speaks about *sub*-CRSs.

*2.1.2 HRS Terms* A HRS $\mathcal{H}$ is just like a CRS a pair $(\mathcal{A}, \mathcal{R})$ where $\mathcal{A}$ is the alphabet and $\mathcal{R}$ the set of rules. Term formation is specified using $\lambda^\tau$-calculus, Church's simply typed $\lambda$-calculus. (Simple) types are formed from base types written as $\sigma, \tau, \ldots$, and the function type constructor $\rightarrow$.

The alphabet $\mathcal{A}$ of a HRS $\mathcal{H} = (\mathcal{A}, \mathcal{R})$ consists of
- symbols for typed variables $x\ y\ z\ \ldots$,
- a distinguished symbol $\lambda$ for abstraction,
- symbols for typed operators $F\ G\ H\ \ldots$.

Typed terms are formed from abstraction and application according to the following rules:

$$(\mathsf{var})\ \frac{}{x : \tau} \qquad (\mathsf{const})\ \frac{}{F : \tau} \qquad (\mathsf{abstr})\ \frac{\begin{array}{c}[x : \sigma] \\ \vdots \\ t : \tau\end{array}}{\lambda x.t : \sigma \rightarrow \tau} \qquad (\mathsf{appli})\ \frac{t : \sigma \rightarrow \tau \quad t' : \sigma}{tt' : \tau}$$

Although environments are not made explicit, we take it for granted that variables and constants cannot have more than one type. So every typable term has a unique type. Exactly like in $\lambda$-calculus, a variable $x$ occurs *bound* in a term if it occurs in the scope of a $\lambda x$, and it occurs *free* otherwise.

Let $\Box : 0$ be a fresh symbol. A *context* is a term with one or more occurrences of $\Box$. Like in the CRS case, a context with exactly one occurrence of $\Box$ is written as $C[\ ]$, and one with $n$ occurrences of $\Box$ as $C[, \ldots, ]$. If $C[, \ldots, ]$ is a context $C[, \ldots, ]$ with $n$ occurrences of $\Box$ and $t_1, \ldots, t_n$ are terms of type 0, then $C[t_1, \ldots, t_n]$ denotes the result of replacing from left to right the occurrences of $\Box$ by $t_1, \ldots, t_n$.

Only terms (and contexts) in long $\eta$-normal form will be considered:

Definition 2.2 The long $\eta$-normal form of a $\lambda^\tau$-term is obtained by repeatedly replacing $C[t]$ by $C[\lambda x.tx]$, where $x$ does not occur free in $t$, the occurrence of $t$ is non-functional, and $t$ is not an abstraction. This has the effect that all subterms are provided with the right number of arguments.

Example 2.3 In the representation of untyped $\lambda$-calculus as a HRS, we have only one base type 0. The alphabet consists of two operators $\mathsf{app} : 0 \rightarrow (0 \rightarrow 0)$ for application and $\mathsf{abs} : (0 \rightarrow 0) \rightarrow 0$ for $\lambda$-abstraction. Some examples of $\lambda$-terms in this notation are:

$\mathsf{abs}\,(\lambda x.x)$ for $\lambda x.x$,

$\mathsf{app}\,xy$ for $xy$,

$\mathsf{abs}\,(\lambda x.\mathsf{app}yx)$ for $\lambda x.yx$,

$\mathsf{app}\,(\mathsf{abs}\,(\lambda x.x))y$ for $(\lambda x.x)y$.

Like in the CRS case, the HRS-representation of $\lambda$-calculus contains junk. For instance, the term $\lambda x.x$ doesn't correspond to any $\lambda$-term. Note that all $\lambda$-terms and the variables occurring in them have type 0 in this notation.

This example illustrates that a notion of sub-HRS, analogous to the notion of sub-CRS, is called for. Furthermore, the example shows that properties, such as strong normalisation, of the meta-language ($\lambda^\tau$-calculus) have no bearing on properties of the object language ($\lambda$-calculus).

*2.1.3 Comparing Term Formation* We discuss the two most important differences between both formats.

In CRSs metaterm formation is given by a direct inductive definition. Function symbols and metavariables come equipped with an arity and metaterms are formed by supplying these symbols with the right number of arguments. Terms are metaterms not containing metavariables.

In HRSs a direct inductive definition of terms is circumvented by making use of $\lambda^\tau$-calculus term formation. Function symbols come as constants equipped with a type and are combined using the formation rules of $\lambda^\tau$-calculus. Attention is then restricted to terms in long $\eta$-normal form. Most of the time, except at intermediate stages of a computation, attention is further restricted to terms also in $\beta$-normal form.

NOTATION. To allow for an easy comparison we will call HRS-terms of base type in long $\beta\eta$-normal form, in which all variables are of base type just *terms* and the others *metaterms*.

Note that the typing does not mean that only typed systems can be written as a HRS; the typing takes place on a metalevel. If an untyped system is represented as a HRS, then only one base type 0 is used (and all terms of the HRS corresponding to a term in the untyped system we are considering, are of type 0). The base type 0 can be thought of as the set of all well-formed terms. The statement $t : 0$ can be read as '$t$ is a well-formed term'.

Typing in this way, such that well-formed terms are of base type, actually establishes two things. For discussing them, first the *arity* and the *order* of a type are defined.

DEFINITION 2.4 The arity $\mathsf{Ar}(\sigma)$ of a type $\sigma$ is inductively defined as follows:

$$\begin{aligned} \mathsf{Ar}(\sigma) &= 0 \text{ (if } \sigma \text{ is a base type)} \\ \mathsf{Ar}(\sigma \to \tau) &= 1 + \mathsf{Ar}(\tau) \end{aligned}$$

The order $\mathsf{Ord}(\sigma)$ of a type $\sigma$ is defined as follows:

$$\begin{aligned} \mathsf{Ord}(\sigma) &= 0 \text{ (if } \sigma \text{ is a base type)} \\ \mathsf{Ord}(\sigma \to \tau) &= \mathsf{max}(1 + \mathsf{Ord}(\sigma), \mathsf{Ord}(\tau)) \end{aligned}$$

The arity (order) of a term is defined to be the arity (order) of its type.

First, in a term every operator has exactly as many arguments as prescribed by the arity of its type. This is because terms must be in long $\eta$-normal form. For instance, an operator $F : 0 \to (0 \to 0)$ can form a term only if it is provided with two arguments $t_1$ and $t_2$ of type 0. So the type of an operator, like the arity of an operator in CRSs, determines how many arguments it should have. Second, in a term all the arguments have the right *order*, indicating how active they are, or, whether they can be applied to other terms. For example an operator $G : (0 \to 0) \to 0$ should have one argument of order 1. The order of an operator cannot be directly expressed in the CRS framework. The arity of an CRS operator only prescribes

how many arguments this operator should get, but nothing is specified about the orders these arguments should have.

The second difference is that in CRSs a distinction is made between metavariables and variables and metaterms and terms. Metavariables occur only in metaterms, which in turn occur only as the left- or right-hand side of rewrite rules. The objects which are rewritten are terms. This distinction is made in order to stress the point that a rewrite rule acts as a scheme, so its left- and right-hand side are not ordinary terms. Taking this point of view, $x$ in $F(x)$-as-a-term is a variable, and $x$ in $F(x)$-as-a-left-or-right-hand-side is a metavariable. In CRS notation, the former is written as $F(x)$ and the latter as $F(Z)$. In HRSs no distinction is made between metavariables and variables. Both terms and metaterms can be rewritten. The metavariables in CRS-rules correspond to free variables in HRS-rules.

*2.2 Rule Formation*
In this section we will compare the rule formation of CRSs with the one of HRSs. We show that rewrite rules in both formats satisfy equivalent requirements.

*2.2.1 CRS Rules* In a CRS, a rewrite rule $l \to r$ must satisfy the following:
(1) $l$ and $r$ are metaterms,
(2) the head-symbol of $l$ is an operator symbol,
(3) all metavariables in $r$ occur in $l$ as well, $l$ and $r$ are closed,
(4) a metavariable $Z$ in $l$ occurs only in the form $Z(x_1, \ldots, x_n)$ with $x_1, \ldots, x_n$ distinct bound variables.
We call the last condition the *pattern*-condition.

EXAMPLE 2.5 The $\beta$-rule of $\lambda$-calculus, $(\lambda x.M)N \to M[x := N]$ is written in CRS format as

$$@(\lambda([x]Z(x)), Z') \to Z(Z')$$

The head-symbol of the left-hand side is @, and the metavariables $Z$ and $Z'$ occur in both sides.

*2.2.2 HRS Rules* A rewrite rule $l \to r$ in a HRS must meet the following requirements:
(1) $l$ and $r$ are both long $\beta\eta$-normal forms of base type,
(2) $l$ is not $\eta$-equivalent to a free variable,
(3) all free variables in $r$ occur in $l$ as well,
(4) a free variable $z$ in $l$ occurs only in the form $zt_1 \ldots t_n$ with $t_1, \ldots, t_n$ $\eta$-equivalent to $n$ distinct bound variables.
Like for CRSs, the last condition is called the *pattern*-condition.

EXAMPLE 2.6 The $\beta$-rewrite rule in HRS notation is

$$\mathsf{app}\,(\mathsf{abs}\,(\lambda x.yx))z \to yz$$

with $x, z : 0$ and $y : 0 \to 0$.

*2.2.3 Comparing Rule Formation* Remembering that metavariables in rewrite rules of CRSs correspond to free variables in rewrite rules, it is not difficult to see that the requirements (1)–(4) of CRS rules correspond to the same ones of HRS rules.

The first condition specifies that rules are built from metaterms. The second one states that left-hand sides must have some structure and the third one that rewriting cannot introduce arbitrary terms. These conditions are familiar from first-order rewriting. The last condition is

the pattern-condition. By that condition only names (simple objects), not values (compound objects) can occur as arguments of free variables. Both in the case of CRSs and of HRSs it establishes decidability of unification of patterns, and computability of the rewrite relation, a result of [Mil]. Intuitively this is so because it is still possible to 'see' the pattern from its substitution instances, that is, instantiating a pattern doesn't really change its structure.

*2.3 Rewrite Step Generation*

Once we know what requirements the rewrite rules should satisfy, we have to define for both formats how rewrite rules are instantiated in order to obtain an actual rewrite step. In both cases, we have to plug in some term in the 'holes' of the rule. In CRSs, the holes in the rule are the metavariables, and in HRSs the free variables. The ways in which metavariables and free variables are assigned a value, are related, but nevertheless essentially different.

For defining substitution for CRSs, $\underline{\lambda}$-calculus is used. The substitution is performed by replacing a metavariable by a (special form of a) $\lambda$-term, and by reducing, in the term obtained by this replacement, all residuals of $\beta$-redexes that are present in the initial term, i.e. by performing a development (or expanding `let`-constructs). The well-known result in $\lambda$-calculus that all developments are finite, guarantees that the substitution is well-defined.

For defining substitution for HRSs, like for defining term and rule formation, $\lambda^\tau$-calculus is used as a metalanguage. The substitution is performed by replacing a free variable by a term of the same type, and reducing the result of the replacement to $\beta$-normal form. In this case, substitution is well-defined since in $\lambda^\tau$-calculus all $\beta$-rewrite sequences eventually terminate.

*2.3.1 CRS Rewrite Steps* In order to define assignments for CRSs we first introduce a new concept: the so-called *substitutes* (cf. [Kah92]). An $n$-ary substitute is an expression of the form $\underline{\lambda}(x_1,\ldots,x_n).s$, where $s$ is a term, $\underline{\lambda}$ a 'metalambda' and $(x_1,\ldots,x_n)$ a tuple of $n$ distinct variables, which are considered to be bound by $\underline{\lambda}$ and may be renamed in the usual way. A substitute $\underline{\lambda}(x_1,\ldots,x_n).s$ can be applied to an $n$-tuple of terms $(t_1,\ldots,t_n)$, yielding $s$ with $x_1,\ldots,x_n$ simultaneously replaced by $t_1,\ldots,t_n$ respectively:

$$(\underline{\lambda}(x_1,\ldots,x_n).s)(t_1,\ldots,t_n) = s[x_1 := t_1 \ldots x_n := t_n]$$

An *assignment* $\sigma$ consists of assigning $n$-ary substitutes to $n$-ary metavariables:

$$\sigma(Z) = \underline{\lambda}(x_1,\ldots,x_n).s \quad (Z \text{ an } n\text{-ary metavariable})$$

It is extended to a mapping from terms to terms in the following way:

$$\begin{array}{rcl} x^\sigma & = & x \\ ([x]t)^\sigma & = & [x]t^\sigma \\ (F(t_1,\ldots,t_n))^\sigma & = & F(t_1^\sigma,\ldots,t_n^\sigma) \\ (Z(t_1,\ldots.t_n))^\sigma & = & \sigma(Z)(t_1^\sigma,\ldots,t_n^\sigma) \end{array}$$

Note that the result of applying $\sigma(Z)$ to $(t_1^\sigma,\ldots,t_n^\sigma)$ in the last clause is indeed a term.

A variable in an instance of a metavariable should be bound only if it is bound in the occurrence of the metavariable. Unintended bindings occur for instance in $(F[x]Z)^\sigma$ if $\sigma(Z) = x$, and in $(Z(Z'))^\sigma$ if $\sigma(Z) = \underline{\lambda}(x).[y]x$ and $\sigma(Z') = y$. These problems can be avoided by renaming bound variables. In the following we will assume that this is done whenever necessary.

NOTATION. In this paper we stick to the definition of [KOR93] of substitution as a one-stage process. If we would use $\underline{\lambda}$-calculus as a meta-language, we would obtain substitution as a

two-stage process: first replacing the metavariables by the terms assigned to them, and then explicitly developing the $\underline{\beta}$-redexes. This would yield a presentation closer to the one of HRSs.

Rewrite rules generate a rewrite relation $\rightarrow$ on terms in the following way. If $l \rightarrow r$ is a rewrite rule and $\sigma$ an assignment, then $C[l^\sigma] \rightarrow C[r^\sigma]$ is a *rewrite* or *reduction step*, where $C[\,]$ is some context. A *contraction* is defined as $l^\sigma \rightarrow r^\sigma$. The reflexive-transitive closure of $\rightarrow$ is called *rewriting* and is denoted as $\twoheadrightarrow$. If $s \twoheadrightarrow t$ then we say that $s$ *rewrites* to $t$. If we want to make explicit that a rewrite rule $R$ is applied in a rewrite step we write $\rightarrow_R$ instead of $\rightarrow$.

*2.3.2 HRS Rewrite Steps* In a HRS, an assignment is a finite mapping from variables to terms in long $\beta\eta$-normal form of the same type. Using the variable convention of $\lambda$-calculus, an assignment $\sigma$ is extended to a mapping from terms to metaterms, in the following way:

$$\begin{aligned} F^\sigma &= F \quad \text{(for a constant } F\text{)} \\ x^\sigma &= \sigma(x) \text{ (for a variable } x\text{)} \\ (\lambda x.t)^\sigma &= \lambda x.t^\sigma \\ (tt')^\sigma &= t^\sigma t'^\sigma \end{aligned}$$

A rewrite relation $\rightarrow$ on terms in long $\beta\eta$ normal form is generated in the following way. If $l \rightarrow r$ is a rewrite rule and $\sigma$ an assignment, then $C[l^\sigma\downarrow_\beta] \rightarrow C[r^\sigma\downarrow_\beta]$ is a rewrite step. Here $\downarrow_\beta$ denotes $\beta$-reduction to normal form. Such a normal form indeed exists since simply typed $\lambda$-calculus is considered. A contraction is defined as $l^\sigma\downarrow_\beta \rightarrow r^\sigma\downarrow_\beta$. The terminology of rewriting is the standard one like in the CRS case.

*2.3.3 Comparing Rewrite Step Generation* In both formats it is the case that the first step in performing a substitution is to replace a 'hole' in the rewrite rule by a kind of '$\lambda$-term'. Then we compute the result of this replacement. And here lies the difference: since in the case of CRSs we perform only a development of the $\lambda$-terms, there is no reduction of created redexes. On the other hand, to compute the result for HRSs full fledged $\lambda^\tau$-calculus is used, that is, redexes that are created during rewriting are also contracted.

To get an idea of what kind of difference in the rewrite relations we have due to these distinct evaluation mechanisms, consider the following example. The HRS rule $F(\lambda y.z(\lambda x.yx)) \rightarrow z(\lambda x.x)$ with assignment $\sigma : z \mapsto \lambda u.uK$. We have the rewrite step:

$$\begin{aligned} F(\lambda y.yK) &= F(\lambda y.(\lambda x.yx)K)\downarrow_\beta \\ &= F(\lambda y.(\lambda u.uK)(\lambda x.yx))\downarrow_\beta \\ &= (F(\lambda y.z(\lambda x.yx)))^\sigma\downarrow_\beta \\ &\rightarrow (z(\lambda x.x))^\sigma\downarrow_\beta \\ &= (\lambda u.uK)\lambda x.x\downarrow_\beta \\ &= (\lambda x.x)K\downarrow_\beta \\ &= K \end{aligned}$$

Observe how the complete development of the $\lambda$-redexes of the assignment creates a new redex which is also contracted (in the last line). This redex is 'created downwards', so for this process to end, we cannot rely on termination of developments or even superdevelopments (cf. [Raa93]), but really need strong normalisation of simply typed $\lambda$-calculus. On the other hand, the corresponding CRS rule $F([y]Z(y)) \rightarrow Z([x]x)$ and assignment $\sigma : Z \mapsto \underline{\lambda}(u).@(u, K)$, act more lazily:

$$F([y]@(y, K)) = (F([y]Z(y)))^\sigma$$

$$\begin{array}{lll} & \rightarrow & (Z([x]x))^{\sigma} \\ & = & @([x]x, K) \end{array}$$

The substitution is evaluated by a complete development of the $\underline{\lambda}$-redex. We have to add an explicit $\beta$-reduction step, namely

$$@([x]x, K) \quad \rightarrow \quad K$$

in order to simulate the HRS rewrite step completely.

An other way of looking at it is to view $[\_]\_$ really as an abbreviation of $\Lambda(\underline{\lambda}\_.\_)$, for some fresh symbol $\Lambda$. Now, although it seems that $\underline{\beta}$-redexes can be created in the substitution process above due to the presence of $\underline{\lambda}$'s in terms, this is not the case because they are always 'blocked' by the $\Lambda$. In the example, we end up with the term $@(\Lambda(\underline{\lambda}x.x), K)$. A 'rule' like $@(\Lambda(Z), Z') \rightarrow ZZ'$ is needed to 'unblock' the metalanguage redex $(\underline{\lambda}x.x)K$. This is the only thing used in the translation of CRSs into HRSs.

The same 'blocking' idea of this translation can also be used to show that developments of terms in $\lambda$-calculus must terminate: put fresh variables 'in front of' abstractions and applications not taking part in a $\beta$-redex. This gives a trivial typable $\lambda^{\tau}$-term which exactly simulates developments. Creating new redexes is prevented by the presence of the fresh variables.

## 3. Translating a CRS into a HRS

In this section we will show that a CRS can be translated into a HRS such that there is a one-to-one correspondence between rewritings in the CRS and in its associated HRS. We use $\langle\_\rangle$ as notation for the translation. The mapping $\langle\_\rangle$ is chosen to be injective.

Definition 3.1 The HRS alphabet $\langle\mathcal{A}\rangle$ associated with a CRS alphabet $\mathcal{A}$ consists of

- the symbol $\Lambda : (0 \rightarrow 0) \rightarrow 0$ (meant to 'collapse' a functional type),
- a variable $\langle x\rangle = x : 0$, for each variable $x$ in $\mathcal{A}$,
- a constant $\langle F\rangle = F : 0 \rightarrow \ldots \rightarrow 0 \rightarrow 0$ ($\mathsf{Ar}(F) + 1$ times 0), for each operator $F$ in $\mathcal{A}$,
- a variable $\langle Z\rangle = z : 0 \rightarrow \ldots \rightarrow 0 \rightarrow 0$ ($\mathsf{Ar}(Z) + 1$ times 0), for each metavariable $Z$ in $\mathcal{A}$,

and the ordinary symbols in a HRS alphabet.

Note that only one base type, namely 0, is used The translation CRS metaterms and contexts is defined by extending the translation of symbols as follows

Definition 3.2 (1) $\langle[x]t\rangle = \Lambda(\lambda x.\langle t\rangle)$

Abstractions are translated as projected $\lambda$-abstractions.

(2) $\langle F(t_1, \ldots, t_n)\rangle = F\langle t_1\rangle \ldots \langle t_n\rangle$

$\langle Z(t_1, \ldots, t_n)\rangle = z\langle t_1\rangle \ldots \langle t_n\rangle$

Functional terms are translated by currying.

(3) $\langle\Box\rangle = \Box : 0$

Holes are of base type.

The translation of a context $\langle C[\,]\rangle$ is denoted by $\langle C\rangle[\,]$. The translation of CRS rule $R : l \rightarrow r$ is defined as $\langle R\rangle : \langle l\rangle \rightarrow \langle r\rangle$.

For a CRS $\mathcal{C}$, the HRS $\langle\mathcal{C}\rangle$ is obtained by translating the alphabet and the set of rules of $\mathcal{C}$. We first give the translation of the main ingredient needed in a rewrite step: assignment.

Definition 3.3 The translation $\langle\sigma\rangle$ of an assignment $\sigma$ is defined as follows: if $\sigma(Z) = \underline{\lambda}(x_1, \ldots, x_n).s$, then $\langle\sigma\rangle(z) = \lambda x_1 \ldots x_n.\langle s\rangle$.

Of course, we have to show that these translations are correct in the sense that a CRS concept yields the corresponding HRS concept.

PROPOSITION 3.4 *Let $s'$ be the translation $\langle s \rangle$ of a CRS metaterm $s$. Then*
**a** *$s' : 0$, moreover there is a bijective correspondence between subterms of $s$ and subterms of type 0 of $s'$,*
**b** *$s'$ is in long $\beta\eta$-normal form,*
**c** *if $s$ satisfies the pattern-condition, then $s'$ satisfies the pattern-condition,*
**d** *$\langle \mathsf{Fvar}(s) \rangle = \mathsf{Fvar}(s')$, where Fvar denotes the set of free variables in a CRS (or HRS) metaterm.*

PROOF. The four properties are proved simultaneously, by structural induction. □

The bijective correspondence in **a** can be made more precise using the notion of position.

PROPOSITION 3.5 *The translations of CRS rules, contexts, and assignments yield the corresponding concepts in the associated HRS.*

PROOF. The translation $\langle R \rangle : \langle l \rangle \to \langle r \rangle$ of a CRS rule $R : l \to r$, is a HRS rule because one easily verifies the conditions on the form of the rule:
- The head-symbol of $l$ is an operator symbol, so $\langle l \rangle = F \ldots$ is not $\eta$-equivalent to a free variable.
- $$\begin{array}{rcll} \mathsf{Fvar}(\langle l \rangle) & = & \langle \mathsf{Fvar}(l) \rangle & (\text{by } \mathbf{d}) \\ & = & \langle \mathsf{Mvar}(l) \rangle & (l \text{ is closed}) \\ & \supseteq & \langle \mathsf{Mvar}(r) \rangle & (\text{by assumption}) \\ & = & \langle \mathsf{Fvar}(r) \rangle & (r \text{ is closed}) \\ & = & \mathsf{Fvar}(\langle r \rangle) & (\text{by } \mathbf{d}) \end{array}$$
- By assumption $l$ is closed, hence by the assumption that $l$ satisfies the pattern-condition we conclude from **c**, that $\langle l \rangle$ satisfies the pattern-condition.
- The fact that $l$ and $r$ are terms of type 0 in long $\beta\eta$-normal form follows directly from **a** and **b**.

Next, it suffices to note that the proof for terms carries over to contexts in order to prove that CRS contexts are translated into HRS contexts. Finally, let $\sigma : Z \mapsto \underline{\lambda}(x_1, \ldots, x_n).t$ be a CRS assignment. Then $\langle \sigma \rangle : z \mapsto \lambda x_1 \ldots x_n.\langle t \rangle$.

$$\frac{x_1 : 0 \quad \ldots \quad x_n : 0 \quad \langle t \rangle : 0}{\lambda x_1 \ldots x_n.\langle t \rangle : 0 \to \ldots \to 0 \to 0}$$

Using **a** and **b**, one easily checks that $\lambda x_1 \ldots x_n.\langle t \rangle$ is in long $\beta\eta$-normal form and has the same type as $z$. □

Next we state some propositions expressing the interaction between forming contexts and applying assignments on the one hand and translating on the other hand.

PROPOSITION 3.6
**a** $\langle C[t] \rangle = \langle C \rangle[\langle t \rangle]$
**b** $\langle C[t_1, \ldots, t_n] \rangle = \langle C \rangle[\langle t_1 \rangle, \ldots, \langle t_n \rangle]$

**c** $\langle s[x_1 := t_1 \ldots x_n := t_n]\rangle = \langle s\rangle[x_1 := \langle t_1\rangle \ldots x_n := \langle t_n\rangle]$

PROOF.
**a** The proof proceeds by induction on the structure of $C[\,]$.
**b** By repeatedly applying **a**.
**c** Choose a context $C[,\ldots,]$ such that $s = C[x_{i_1},\ldots,x_{i_j}]$ and precisely the occurrences of the variables $x_1,\ldots,x_n$ are being displayed. Then

$$\begin{aligned}\langle s[x_1 := t_1 \ldots x_n := t_n]\rangle &= \langle C[t_{i_1},\ldots,t_{i_j}]\rangle \\ &= \langle C\rangle[\langle t_{i_1}\rangle,\ldots,\langle t_{i_j}\rangle] && \text{(by } \mathbf{b}\text{)} \\ &= \langle s\rangle[x_1 := \langle t_1\rangle \ldots x_n := \langle t_n\rangle] && \text{(use Proposition 3.4 } \mathbf{a}\text{)}\end{aligned}$$

□

PROPOSITION 3.7 *For every metaterm $t$ and assignment $\sigma$ we have $\langle t^\sigma\rangle = \langle t\rangle^{\langle\sigma\rangle}\!\downarrow_\beta$.*

PROOF. The proposition is proved by induction on the structure of the metaterm $t$. Note that all $\lambda$'s in $\langle t\rangle$ that are introduced by translating the abstraction operator do not yield a $\beta$-redex since they are 'blocked' by their big brother $\Lambda$. □

We combine the previous two propositions to show that rewrite steps are naturally preserved by the translation.

THEOREM 3.8 *If $s \to_R t$ in a CRS $\mathcal{C}$ with $R : l \to r$ a rewrite rule, then we have $\langle s\rangle \to_{\langle R\rangle} \langle t\rangle$ in the corresponding HRS $\mathcal{H}$.*

PROOF. Let $s \to_R t$ in $\mathcal{C}$, where $s = C[l^\sigma]$ and $t = C[r^\sigma]$, for some context $C[\,]$ and some assignment $\sigma$. Then,

$$\begin{aligned}\langle s\rangle &= \langle C[l^\sigma]\rangle \\ &= \langle C\rangle[\langle l^\sigma\rangle] && \text{(by Proposition 3.6)} \\ &= \langle C\rangle[\langle l\rangle^{\langle\sigma\rangle}\!\downarrow_\beta] && \text{(by Proposition 3.7)} \\ &\to \langle C\rangle[\langle r\rangle^{\langle\sigma\rangle}\!\downarrow_\beta] \\ &= \langle C\rangle[\langle r^\sigma\rangle] && \text{(by Proposition 3.7)} \\ &= \langle C[r^\sigma]\rangle && \text{(by Proposition 3.6)} \\ &= \langle t\rangle\end{aligned}$$

□

This proves that for every rewrite step in a CRS $\mathcal{C}$ a rewrite step in the associated HRS $\mathcal{H}$ can be performed. Now we will show that a rewrite step in the translation of a term must originate from a rewrite step in $\mathcal{C}$ itself. For this, we will use that both contexts and assignments in $\mathcal{H}$ can be translated back into the corresponding concepts in $\mathcal{C}$, under the proper restrictions.

PROPOSITION 3.9 *If $C'[t'] = \langle s\rangle$, then there exist $C[\,]$ and $t$ such that $\langle C[\,]\rangle = C'[\,]$ and $\langle t\rangle = t'$.*

PROOF. From Proposition 3.4 **a** we have $C'[\,] : 0$ and by the definition of a HRS context $t' : 0$. Then the bijective correspondence of Proposition 3.4 **a** provides us with suitable $C[\,]$ and $t$. □

PROPOSITION 3.10 *If $\langle l\rangle^{\sigma'}\downarrow_\beta = \langle s\rangle$ and $l$ satisfies the conditions for a left-hand side of a CRS rule, then there exists a $\sigma$ such that $\langle\sigma\rangle = \sigma'$.*

PROOF. By Proposition 3.4 **d**, we know that a free variable $z$ in $\langle l\rangle$ stems from a metavariable $Z$ in $l$. By the pattern-condition, each free variable occurs only in subterms of the form $zx_1 \dots x_n$ in $\langle l\rangle$, where $n$ is the arity of $Z$, which have type 0 by Proposition 3.4 **a**. Note that all the $x_i$ have type 0, so this subterm is $\eta$-expanded. If $\sigma'(z) = \lambda y_1 \dots y_m.t'$, then $m = n$, because $\sigma'(z)$ is by definition in long $\beta\eta$-normal form. Hence, $(zx_1 \dots x_n)^{\sigma'}\downarrow_\beta = t'[y_1 := x_1 \dots y_n := x_n]\downarrow_\beta = t'[y_1 := x_1 \dots y_n := x_n]$, because renaming doesn't create redexes. It is easy to show that $t'[y_1 := x_1 \dots y_n := x_n]$ must in fact be a subterm (of type 0!) of $\langle s\rangle$ and therefore a translation of some term $\hat{t}$. Now we can define $\sigma(Z) = \underline{\lambda}(y_1, \dots, y_n).\hat{t}[x_1 := y_1 \dots x_n := y_n]$, which meets the requirements. Note that by the pattern-condition all the $x_i$ are distinct. □

THEOREM 3.11 *Let $\mathcal{C}$ be a CRS and $\mathcal{H}$ its associated HRS. If $\langle s\rangle \to_{\langle R\rangle} t'$ in $\mathcal{H}$ by rewrite rule $\langle R\rangle : \langle l\rangle \to \langle r\rangle$, then $s \to_R t$ for some CRS term $t$ such that $\langle t\rangle = t'$.*

PROOF. Let $\langle s\rangle \to_R t'$ in $\mathcal{H}$, where $\langle s\rangle = C'[\langle l\rangle^{\sigma'}\downarrow_\beta]$ and $t' = C'[\langle r\rangle^{\sigma'}\downarrow_\beta]$, for some context $C'[\,]$ and some assignment $\sigma'$. Then

$$\begin{aligned}
\langle s\rangle &= C'[\langle l\rangle^{\sigma'}\downarrow_\beta] \\
&= \langle C\rangle[\langle l\rangle^{\sigma'}\downarrow_\beta] && \text{(by Proposition 3.9)} \\
&= \langle C\rangle[\langle l\rangle^{\langle\sigma\rangle}\downarrow_\beta] && \text{(by Proposition 3.10)} \\
&= \langle C\rangle[\langle l^\sigma\rangle] && \text{(by Proposition 3.7)} \\
&= \langle C[l^\sigma]\rangle && \text{(by Proposition 3.6)}
\end{aligned}$$

By injectivity we have $s = C[l^\sigma]$. If we take $t = C[r^\sigma]$, then $s \to t$ and

$$\begin{aligned}
\langle t\rangle &= \langle C[r^\sigma]\rangle \\
&= \langle C\rangle[\langle r^\sigma\rangle] && \text{(by Proposition 3.6)} \\
&= \langle C\rangle[\langle r\rangle^{\langle\sigma\rangle}\downarrow_\beta] && \text{(by Proposition 3.7)} \\
&= \langle C\rangle[\langle r\rangle^{\sigma'}\downarrow_\beta] \\
&= t'
\end{aligned}$$

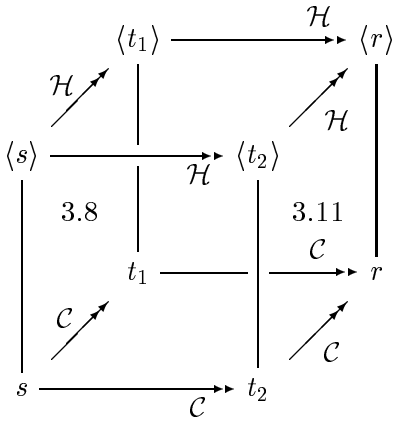
□

How natural a translation is can be measured by the properties which it preserves and reflects. Theorems 3.8 and 3.11 state that the main property of CRSs and HRSs, i.e. whether one term rewrites (in one step) to another, is both preserved and reflected. Combining this with the fact that orthogonality is preserved, we obtain a confluence proof for orthogonal CRSs via confluence of their associated HRS. For the definition of orthogonality of CRSs and HRSs we refer the reader to [Klo80] and [Nipb].

COROLLARY 3.12 *Orthogonal CRSs are confluent.*

PROOF. Let $s \twoheadrightarrow_{\mathcal{C}} t_1$ and $s \twoheadrightarrow_{\mathcal{C}} t_2$ be rewrites in an orthogonal CRS $\mathcal{C}$. By Theorem 3.8, we can lift these to rewrites $\langle s\rangle \twoheadrightarrow_{\mathcal{H}} \langle t_1\rangle$ and $\langle t\rangle \twoheadrightarrow_{\mathcal{H}} \langle t_2\rangle$ in the HRS $\mathcal{H}$ associated to $\mathcal{C}$. Because $\mathcal{H}$ is easily seen to be orthogonal, we conclude from [Nipb, Cor. 4.9] that it is confluent, hence there exist rewrites $\langle t_1\rangle \twoheadrightarrow_{\mathcal{H}} r'$ and $\langle t_2\rangle \twoheadrightarrow_{\mathcal{H}} r'$, for some $r'$. These sequences can be projected again to

form $t_1 \twoheadrightarrow_{\mathcal{C}} r$ and $t_2 \twoheadrightarrow_{\mathcal{C}} r$ by Theorem 3.11, also showing that $\langle r \rangle = r'$. The proof is expressed by the following diagram.




□

## 4. Translating a HRS into a CRS plus explicit $\beta$

In this section we define a translation from HRSs into CRSs. Due to the difference in defining assignments, discussed in section 2, the translation is not as straightforward as the translation the other way round. The reason is that in CRSs, developments of untyped $\lambda$-calculus are used to define assignments, whereas in the case of HRSs this is done by reductions to normal form in simply typed $\lambda$-calculus.

In order to be able to simulate every rewrite of a HRS in its associated CRS, a $\beta$-reduction rule has to be added to the translation. It is given as

$$@([x]Z(x), Z') \to_\beta Z(Z')$$

To simplify the notation a bit we sometimes use $@_n$ to abbreviate $n$ applications, for instance $@_2(A, B, C)$ stands for $@(@(A, B), C)$. Formally, $@_n$ for $n \geq 1$ is defined as

$$\begin{aligned} @_1(t, t_1) &= @(t, t_1) \\ @_{n+1}(t, t_1, \ldots, t_n) &= @(@_n(t, t_1, \ldots, t_{n-1}), t_n) \end{aligned}$$

Again, the translation is denoted as $\langle \_ \rangle$ and is chosen to be injective. In this case we do not obtain a $1-1$-correspondence between rewrite steps in a HRS $\mathcal{H}$ and the rewrite steps in its encoding. Let's write $\mathcal{C}$ for the CRS with a set of rules consisting of the translated $\mathcal{H}$-rules plus the $\beta$-rules. The translation then satisfies the weaker property that if $s \to_{\mathcal{H}} t$ in a HRS $\mathcal{H}$, then $\langle s \rangle \to_{\mathcal{C}\beta} \langle t \rangle$, where $\to_{\mathcal{C}\beta}$ is defined to be a rewrite in $\mathcal{C}$ consisting of one step via a translated $\mathcal{H}$-rule followed by a $\beta$-reduction to $\beta$-normal form. Moreover, we obtain that a rewrite in the encoding of $\mathcal{H}$ starting with the encoding of some term of $\mathcal{H}$ can be extended to a rewrite corresponding to a rewrite in the original HRS. We will denote $\beta$-reduction to normal form as $\to_\beta^!$.

First to an alphabet $\mathcal{A}$ of a HRS a CRS alphabet $\langle \mathcal{A} \rangle$ is associated.

Definition 4.1 The CRS alphabet $\langle \mathcal{A} \rangle$ associated with a HRS alphabet $\mathcal{A}$ consists of

- a symbol @ for application,
- for every symbol $F \in \mathcal{A}$ for an operator of type $\tau$, a symbol $F$ for an operator with arity $n = \mathsf{Ar}(\tau)$,

- the ordinary symbols of a CRS alphabet, i.e. symbols for variables $x\ y\ z\ldots$, symbols for metavariables with a fixed arity $Z\ Z_0\ Z_1\ldots$ and a symbol for abstraction, $[\_]\_$.

DEFINITION 4.2 The translation of terms in long $\beta\eta$-normal form is defined inductively as follows:
- $\langle\lambda x_1\ldots x_n.xt_1\ldots t_n\rangle = [x_1]\ldots[x_n]@_n(x,\langle t_1\rangle,\ldots,\langle t_n\rangle)$,
- $\langle\lambda x_1\ldots x_n.Ft_1\ldots t_n\rangle = [x_1]\ldots[x_n]F(\langle t_1\rangle,\ldots,\langle t_n\rangle)$.

It is extended to contexts by defining $\langle\Box\rangle = \Box$. We write $\langle C\rangle[\,]$ for the translation of $C[\,]$.

Free variables in terms of HRSs correspond to free variables in terms of CRSs. Free variables in rules of HRSs correspond to metavariables in rules of CRSs. Therefore a separate definition of the translation of a rule has to be given, in which free variables are translated in another way than in the translation of a term.

DEFINITION 4.3 The translation $\langle l\to r\rangle$ of a HRS rule $l\to r$ is defined as $\langle l\rangle\to\langle r\rangle$, where $\langle l\rangle$ and $\langle r\rangle$ are defined inductively as follows.

**a** The left-hand side $l$ of a HRS rewrite rule is of the form $l = Ft_1\ldots t_k$. Here $t_1,\ldots,t_k$ are long $\beta\eta$-normal forms in which inputs of free variables are $\eta$-equivalent to distinct bound variables. The translation $\langle l\rangle$ of $l$ is defined by induction on the structure of such a long $\beta\eta$-normal form.
- $\langle\lambda x_1\ldots x_m.xt_1\ldots t_n\rangle = [x_1\ldots x_m]@(x,\langle t_1\rangle,\ldots,\langle t_n\rangle)$, if $x$ is a variable which is bound in $l$,
- $\langle\lambda x_1\ldots x_m.zt_1\ldots t_n\rangle = [x_1\ldots x_m]Z(t_1{\downarrow_\eta},\ldots,t_n{\downarrow_\eta})$, if $z$ is a variable which is free in $l$ (note that $t_1,\ldots,t_n$ are $\eta$-equivalent to distinct bound variables by the pattern-condition),
- $\langle\lambda x_1\ldots x_m.Ft_1\ldots t_n\rangle = [x_1\ldots x_m]F(\langle t_1\rangle,\ldots,\langle t_n\rangle)$.

**b** The right-hand side $r$ of a HRS rewrite rule is of the form $r = st_1\ldots t_k$ with $s$ a symbol standing for a free variable or an operator and $t_1,\ldots,t_k$ in long $\beta\eta$-normal form. The translation $\langle r\rangle$ of $r$ is by induction on the structure of a long $\beta\eta$-normal form.
- $\langle\lambda x_1\ldots x_m.xt_1\ldots t_n\rangle = [x_1\ldots x_m]@(x,\langle t_1\rangle,\ldots,\langle t_n\rangle)$, if $x$ is a variable bound in $r$,
- $\langle\lambda x_1\ldots x_m.zt_1\ldots t_n\rangle = [x_1\ldots x_m]Z(\langle t_1\rangle,\ldots,\langle t_n\rangle)$, if $z$ is a variable free in $r$,
- $\langle\lambda x_1\ldots x_m.Ft_1\ldots t_n\rangle = [x_1\ldots x_m]F(\langle t_1\rangle,\ldots,\langle t_n\rangle)$.

As in the translation from CRSs to HRSs, we show that rewrite steps in a HRS can be simulated by essentially the same step in the associated CRS. To that end, the translation is extended to assignments.

DEFINITION 4.4 An assignment of a HRS assigns to a variable a term of the same type in long $\beta\eta$-normal form. So an assignment assigns to a variable $y$ of type $\tau$ a term of the form $\lambda x_1\ldots x_n.t$ with $n = \mathsf{Ar}(\tau)$ and $t$ not a $\lambda$-abstraction. The translation $\langle\sigma\rangle$ of an assignment $\sigma$ is defined as follows: if $\sigma(y) = \lambda x_1\ldots x_n.t$, then $\langle\sigma\rangle(Y) = \underline{\lambda}(x_1,\ldots,x_n).\langle t\rangle$.

First we show that the translation produces correct terms.

PROPOSITION 4.5 *If $t$ is a HRS term in long $\beta\eta$-normal form, then $\langle t\rangle$ is well-defined as a CRS term.*

PROOF. The proof proceeds by induction on the structure of a long $\beta\eta$-normal form. □

The next proposition states that the translation of a rewrite rule is well-defined.

PROPOSITION 4.6 *The translation $\langle l \to r \rangle$ of a HRS rewrite rule $l \to r$ satisfies the definition of a CRS rewrite rule.*

PROOF. It is easy to verify that $\langle l \rangle$ and $\langle r \rangle$ are both well-formed CRS metaterms. Further, observe that
- $\mathsf{Mvar}(l) \supset \mathsf{Mvar}(r)$ since metavariables originate from free variables,
- the head-symbol of $\langle l \rangle$ is an operator symbol,
- all variables occur bound,
- metavariables occurring in $\langle l \rangle$ have distinct bound variables as input.

□

Then we show that decomposing a term by a context, commutes with the translation.

PROPOSITION 4.7 $\langle C[t] \rangle = \langle C \rangle [\langle t \rangle]$.

PROOF. The proof proceeds by induction on the definition of $C[\,]$. □

Finally we show that decomposing a term into a (meta)term and an assignment almost commutes with the translation. For a decomposition into a left-hand side, i.e. a pattern, the commutation is perfect, but for right-hand sides we need additional $\beta$-steps. This is proved in the following two propositions.

PROPOSITION 4.8
**a** *Let $s$ be a term in long $\beta\eta$-normal form, and $u_1, \ldots, u_n$ terms that are $\eta$-equivalent to distinct variables. Then $\langle s \rangle [z_1 := u_1 \downarrow_\eta \ldots z_n := u_n \downarrow_\eta] = \langle s[z_1 := u_1 \ldots z_n := u_n] \downarrow_\beta \rangle$.*
**b** $\langle l \rangle^{\langle \sigma \rangle} = \langle l^\sigma \downarrow_\beta \rangle$.

PROOF.
**a** The proof proceeds by induction on the structure of $s$.
**b** The is proved by induction on the structure of a long $\beta\eta$-normal form, in which arguments of free variables are $\eta$-equivalent to distinct bound variables.

□

Combining the last two propositions, we observe that the 'matching power' (or complexity of matching, depending on one's point of view) of HRSs is already present in CRSs, making a natural encoding of the former into the latter possible. This is due to the pattern-condition of HRSs. For HRSs not satisfying the pattern-condition (cf. [Wol93]) this is no longer the case and an encoding seems to be not straightforward anymore (even if we would lift some of the restrictions on left-hand sides of CRS-rules). The next proposition shows that although CRSs and HRSs have the same matching power, HRSs have more 'rewrite power', i.e. they can do more in one step.

PROPOSITION 4.9
**a** *Let $s$ and $u_1, \ldots, u_n$ be terms in long $\beta\eta$-normal form. Then we have $\langle s \rangle [z_1 := \langle u_1 \rangle \ldots z_n := \langle u_n \rangle] \to_\beta^! \langle s[z_1 := u_1 \ldots z_n := u_n] \downarrow_\beta \rangle$.*
**b** $\langle r \rangle^{\langle \sigma \rangle} \to_\beta^! \langle r^\sigma \downarrow_\beta \rangle$.

PROOF.
**a** The proof proceeds by induction on the maximal length of the $\beta$-reduction of $s[z_1 := u_1 \ldots z_1 := u_n]$ to normal form.

**b** The proof proceeds by induction on the structure of $r$.
□

Now we can collect the results of this section to show that every rewrite step in a HRS $\mathcal{H}$ can be simulated in its corresponding CRS $\mathcal{C}$.

THEOREM 4.10 *If $s \to_R t$ by rewrite rule $R = l \to r$ in $\mathcal{H}$, then we have $\langle s \rangle \to_{\langle R \rangle} \to_\beta^! \langle t \rangle$ in the corresponding CRS $\mathcal{C}$.*

PROOF. The term $s$ is of the form $C[l^\sigma\downarrow_\beta]$, and we have $s = C[l^\sigma\downarrow_\beta] \to_R C[r^\sigma\downarrow_\beta] = t$. We have

$$\begin{array}{rcll}
\langle s \rangle & = & \langle C[l^\sigma\downarrow_\beta] \rangle & \\
& = & \langle C \rangle[\langle l^\sigma\downarrow_\beta \rangle] & \text{(by Proposition 4.7)} \\
& = & \langle C \rangle[\langle l \rangle^{\langle \sigma \rangle}] & \text{(by Proposition 4.8)} \\
& \to_{\langle R \rangle} & \langle C \rangle[\langle r \rangle^{\langle \sigma \rangle}] & \\
& \to_\beta^! & \langle C \rangle[\langle r^\sigma\downarrow_\beta \rangle] & \text{(by Proposition 4.9)} \\
& = & \langle C[r^\sigma\downarrow_\beta] \rangle & \text{(by Proposition 4.7)} \\
& = & \langle t \rangle &
\end{array}$$

□

The next thing to be done is to connect somehow a rewrite step in the translation of a HRS with a rewrite step in the original HRS itself. Since the translation of a HRS $\mathcal{H}$ acts as a refinement of $\mathcal{H}$, we cannot hope for a result as neat as in the previous section. But still something can be said. First we will show that if we have the rewrite $\langle s \rangle \to_{\mathcal{C}\beta} t'$ in the translation of a HRS, then we can project it to a rewrite step $s \to_{\mathcal{H}} t$, such that $\langle t \rangle = t'$.

The first observation we need is that there is a 1-1 correspondence between functional subterms in $\langle s \rangle$ , i.e. subterms with a function symbol (also taking the $@_n$ into account) as head, and subterms of type 0 in $s$. Further, we need two propositions.

PROPOSITION 4.11 *Suppose $C'[t'] = \langle s \rangle$ with $t'$ a functional term. Then a context $C[\,]$ and a term $t$ exist such that $\langle C \rangle[\,] = C'[\,]$ and $\langle t \rangle = t'$.*

PROOF. Via the correspondence we obtain an appropriate subterm of $s$, i.e. a context $C[\,]$ and a term $t$ such that $s = C[t]$ and $\langle t \rangle = t'$. Using Proposition 4.7 we have that $\langle C \rangle[\,] = C'[\,]$. □

PROPOSITION 4.12 *Let $l$ be the left-hand side of a HRS rewrite rule. If $\langle l \rangle^{\sigma'} = \langle s \rangle$, then there is an assignment $\sigma$ with $\langle \sigma \rangle = \sigma'$.*

PROOF. Metavariables $Z_i$ occur in $\langle l \rangle$ in the form $Z(x_1, \ldots x_n)$. Suppose $\sigma'$ is defined as $\sigma'(Z_i) = \underline{\lambda}(u_1, \ldots, u_n).t'$, hence $(Z(x_1, \ldots x_n))^{\sigma'} = t'[u_1 := x_1 \ldots u_n := x_n]$. We know that $t' \neq [x]t''$, because otherwise we cannot have $\langle l \rangle^{\sigma'} = \langle s \rangle$ due to the typing. Hence, $t[\ldots]$ is a functional term, i.e. of one of the forms $F(\ldots)$ or $@_n(\ldots)$ with $n \geq 0$ and so in $s$ there is a corresponding subterm $t$ of type 0, such that $\langle t \rangle = t'[u_1 := x_1 \ldots u_n := x_n]$. Define $\sigma$ as $\sigma z = \lambda x_1 \ldots x_n.t$. Then $\langle \sigma \rangle = \sigma'$. □

THEOREM 4.13 *If $\langle s \rangle \to_{\langle R \rangle} \to_\beta^! \langle t \rangle$ in the CRS $\mathcal{C}$ by rewrite rule $R = l \to r$, then $s \to_R t$ in $\mathcal{H}$, for some $t$ such that $\langle t \rangle = t'$.*

PROOF. We have by definition of $\rightarrow_{\mathcal{C}\beta}$, $\langle s\rangle = C'[\langle l\rangle^{\sigma'}] \rightarrow_{\mathcal{C}} t_0 = C'[\langle r\rangle^{\sigma'}] \rightarrow_{\beta}^{!} t'$, for some context $C'[\,]$, assignment $\sigma'$, and term $t_0$. Now we have

$$\begin{aligned}
\langle s\rangle &= C'[\langle l\rangle^{\sigma'}] \\
&= \langle C\rangle[\langle l\rangle^{\sigma'}] \quad \text{(by Proposition 4.11)} \\
&= \langle C\rangle[\langle l\rangle^{\langle\sigma\rangle}] \quad \text{(by Proposition 4.12)} \\
&= \langle C\rangle[\langle l^{\sigma}\downarrow_{\beta}\rangle] \quad \text{(by Proposition 4.8)} \\
&= \langle C[l^{\sigma}\downarrow_{\beta}]\rangle \quad \text{(by Proposition 4.7)}
\end{aligned}$$

By injectivity of $\langle\_\rangle$, we have $s = C[l^{\sigma}\downarrow_{\beta}]$, take $t = C[r^{\sigma}\downarrow_{\beta}]$, then $s \rightarrow_R t$ and

$$\begin{aligned}
\langle t\rangle &= \langle C[r^{\sigma}\downarrow_{\beta}]\rangle \\
&= \langle C\rangle[\langle r^{\sigma}\downarrow_{\beta}\rangle] \quad \text{(by Proposition 4.7)} \\
&\leftarrow_{\beta}^{!} \langle C\rangle[\langle r\rangle^{\langle\sigma\rangle}] \quad \text{(by Proposition 4.9)} \\
&= \langle C\rangle[\langle r\rangle^{\sigma'}] \\
&= t_0
\end{aligned}$$

By confluence of $\beta$, we have $\langle t\rangle = t'$. □

If we want to prove the Church-Rosser property for orthogonal HRSs via the same property for CRSs, Theorem 4.13 is not quite enough. The $\beta$-rule, by construction, indeed is orthogonal to the other rules and coinitial rewrites can be lifted, but only $\mathcal{C}\beta$-steps can be projected, not arbitrary $\mathcal{C}$-rewrites. We now show that every rewrite in an *arbitrary* CRS $\mathcal{C}$ starting with a term which is the translation of some HRS-term, can be completed, by performing a $\beta$-reduction to $\beta$-normal form, to a rewrite which can be simulated by a 'standard' rewrite consisting of $\mathcal{C}\beta$-steps.

The proof follows the strategy employed for proving $\Sigma \models \mathrm{WCR}^{+}$ in [Klo80, pp. 144–148]. However, some difficulties arise. First, because of the possible non-left-linearity of the rules. Second, because simply typed $\lambda$-calculus doesn't satisfy the disjointness property in contrast to underlined $\lambda$-calculus.

The main property to be proved is that $\beta$-reductions to normal form do not interfere with rewrite steps. To do this we first need to define some tracing mechanisms.

DEFINITION 4.14

**a** Let $R = l \rightarrow r$ be a rewrite rule. Its *conditional version* $R_c = l_c \rightarrow r$ is obtained by repeatedly replacing occurrences of a metavariable $Z$ which occurs at least twice in $l$ by a fresh metavariable $Z'$ and adding the condition $Z \equiv Z'$ to the rule. Its *linearisation* is $R_l$ obtained from $R_c$ by omitting the conditions.

**b** Let $r$ be a metaterm, and $\vec{Z} = Z_1, \ldots, Z_n$ be a list of metavariables containing the ones in $r$, then the *freezing* $r_f$ of $\vec{Z}$ in $r$ is defined by $r_f = @_n([\vec{z}]r', [\vec{x_1}]Z_1(\vec{x_1}), \ldots [\vec{x_n}]Z_n(\vec{x_n}))$, where $\vec{z} = z_1, \ldots, z_n$ is a list of fresh variables, and $r'$ is obtained by replacing in $r$ all occurrences of $Z(\vec{t})$ by $@(z, \vec{t})$. For a rule $R = l \rightarrow r$, the freezing $R_f = l \rightarrow r_f$ is defined by freezing the metavariables of $l$ in $r$. We define $R_{f\beta}$ to be the CRS with rules $R_f$ and $\beta$.

**c** Let $R = l \rightarrow r$ be a rewrite rule. Its *underlining* $\underline{R}$ is obtained from $R_{cf}$ by underlining the head-symbol of $l_c$. (So $R$ is first made conditional, then frozen and finally its head-symbol is underlined.)

**d** An ($R$-)underlining of a term $s$ is a term containing some underlined symbols, which are the head-symbols of $\underline{R}$-redexes, and which is equal to $s$ after removing the underlining.

**e** A rewrite $s \twoheadrightarrow_{\mathcal{C}} t$ is an ($R$-)*development* if there is some underlining $\underline{s}$ of $s$, such that $\underline{s} \twoheadrightarrow_{\underline{R}+\beta} \underline{t}$ and the underlined rewrite 'projects' onto the original one. Note that, due to non-left-linearity, the terms in the underlined rewrite need not be underlinings of terms.

NOTATION. This underlining of (head symbols of) redexes might be be considered confusing, because underlinings were also used in $\underline{\lambda}$-calculus. Yet, we think it is the right notation because the underlinings express the same idea of marking both times.

EXAMPLE 4.15 The linearisation of the rule $R = \nu([x]Z(x), [x]Z(x)) \to Z(\nu([x]Z(x)))$ is the rule $R_l = \nu([x]Z(x), [x]Z'(x)) \to Z(\nu[x]Z(x))$. The freezing of $R$ is the rule $R_f = \nu([x]Z(x), [x]Z(x)) \to @([z]@(z, \nu[x]@(z, x)), [x]Z(x))$. The underlining of $R$ is the rule $\underline{R} = \underline{\nu}([x]Z(x), [x]Z'(x)) \to @([z, z']@(z, \nu[x]@(z, x)), [x]Z(x), [x]Z'(x))$ if $Z \equiv Z'$.

The idea of freezing is the one of [Lan93], postponing both duplication of the metavariables and substitution into the metavariables. It is more extensive than the one in [Klo80], where only substitution is postponed. Both postponed actions can be performed by $\beta$-reduction:

PROPOSITION 4.16
**a** *$s \to_{\mathcal{C}\beta} t$ is a development.*
**b** *Let $\underline{s}$ be equal to $s$ after removal of underlinings. If $\underline{s} \to_{\underline{R}\beta} \underline{t}$, then $s \to_{\mathcal{C}\beta} t$. Here $\to_{\underline{R}\beta} = \to_{\underline{R}} \to_{\beta}^{!}$.*

PROOF.
**a** Idea. Underline the redex to obtain an underlining of $s$. Rewrite it with the underlined rule and then to $\beta$-nf.
**b** Idea. In the $\beta$-reduction to normal form, we can do the postponed duplication and substitution steps first and then the others. This rewrite can be projected to a $\mathcal{C}\beta$-step.
□

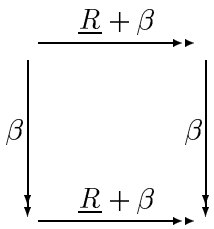
It is not difficult to see that explicit $\beta$-reductions in translated terms can be made to correspond to $\beta$-reductions in $\lambda^{\tau}$-calculus. (Define a suitable forgetful map, forgetting explicit @'s and replacing $[\_]$ by $\lambda$, giving typable terms). Hence $\beta$ is terminating. In the following we only consider rewrites that start with the translation of some term in the HRS.

PROPOSITION 4.17 *Every rewrite in $\underline{R} + \beta$ terminates.*

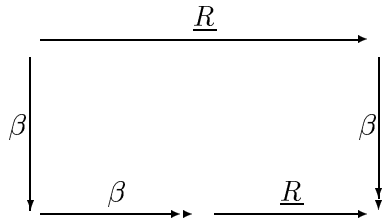
PROOF. Sketch. Let $\underline{R} = \underline{l} \to \underline{r}$. Let $\underline{s}'$ be obtained from $\underline{s}$ by replacing all $\underline{R}$ redexes $\underline{l}^{\sigma}$ by $@([x]x, \underline{r}^{\sigma})$. This can be done unambiguously because $\underline{R}$ doesn't have overlap with itself. The idea is that we have replaced left-hand sides by their right-hand sides in advance, but have put an extra identity in, to keep in mind that we have to do some work to simulate a step. (This replacement works only because the rule is (left- and right-)linear). Now we have that every rewrite starting from $\underline{s}$ can be simulated by a rewrite of the same length starting from $\underline{s}'$. More precisely, each $\beta$-step is simulated by a $\beta$-step and an $\underline{R}$-step $C[\underline{l}^{\sigma}] \to_{\underline{R}} C[\underline{r}^{\sigma}]$ is simulated by the corresponding $\beta$-step $C'[@([x]x, \underline{r}^{\sigma'})] \to_{\beta} C'[\underline{r}^{\sigma'}]$. Since $\beta$-rewriting terminates, the $\underline{R}+\beta$-rewrite must be finite. □

COROLLARY 4.18 *Every development is finite.*

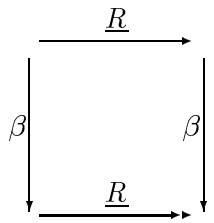
PROPOSITION 4.19 *$\beta$ commutes with $\underline{R}+\beta$. That is*




PROOF. By termination of $\underline{R}+\beta$ (Proposition 4.17) and Newman's Lemma (see e.g. [Oos93]) it suffices to consider only local divergences in proving commutativity. The case of a local divergence of $\beta$-steps is covered by confluence of $\beta$. The other case follows by considering the relative positions of the $\beta$- and $\underline{R}$-redex. If the $\beta$-redex is inside the $\underline{R}$-redex, we have the following diagram.



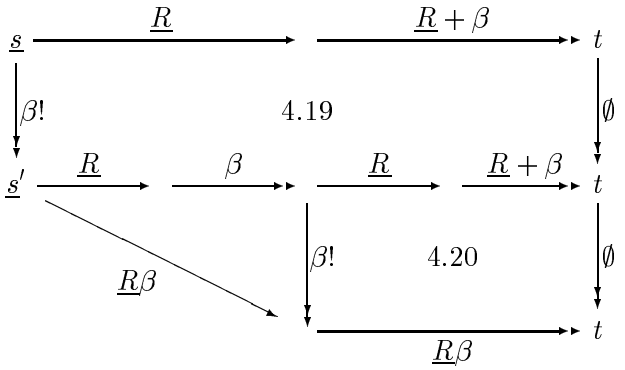

We may need some 'compensating' $\beta$-steps due to possible conditions on $\underline{R}$ (which originate from non-left-linearity of $R$). If the $\beta$-redex is outside the $\underline{R}$-redex, we have the diagram:




The $\beta$-step may duplicate and even nest $\underline{R}$-redexes, but this does not cause much trouble. A parallel inside-out reduction of $\underline{R}$-redexes works. (This is just as easy as for combinations of $\lambda$-calculus and first-order rewrite systems, as $\underline{R}$ is linear.) $\square$

PROPOSITION 4.20 *If $s \to_{\mathcal{C}\beta} t$ and $s \to^{!}_{\beta} s'$, then $s' \twoheadrightarrow_{\mathcal{C}\beta} t$.*

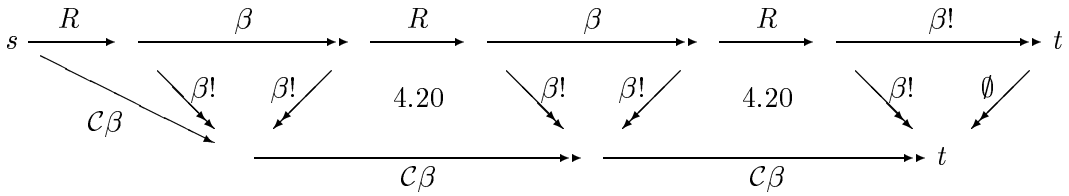
PROOF. By Proposition 4.16 we can construct the reduction $\underline{s} \to_{\underline{R}} \to^{!}_{\beta} t$, for some rule $R$ and underlining $\underline{s}$ of $s$. We prove that if $\underline{s} \to_{\underline{R}} \twoheadrightarrow_{\underline{R}+\beta} t$, such that $t$ is in $\beta$-normal form, and $\underline{s} \to^{!}_{\beta} \underline{s}'$, then $\underline{s}' \to_{\underline{R}\beta} t$. The proof is by induction on the maximal length of a $\underline{R}+\beta$ reduction sequence starting from $\underline{s}$ and expressed by the following diagram.



By applying Proposition 4.16 again (in the reverse direction), we are done. □

LEMMA 4.21 *Suppose $s \twoheadrightarrow_{\mathcal{C}} t$, and $s$ and $t$ are in $\beta$-normal form, then $s \twoheadrightarrow_{\mathcal{C}\beta} t$.*

PROOF. The proof is expressed by the following diagram



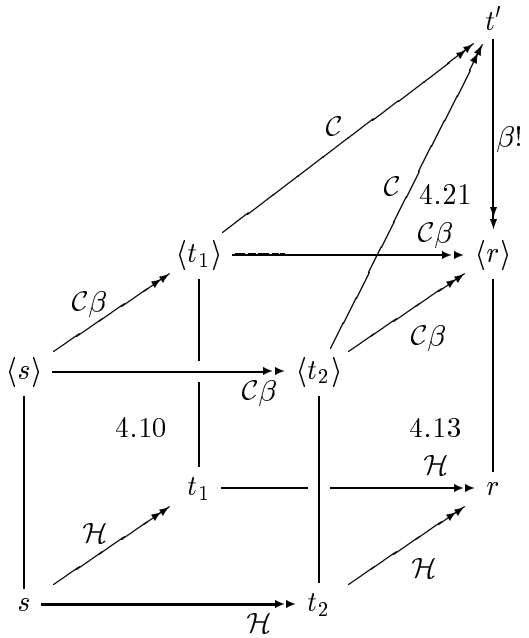

□

Note: The results in the literature on modularity of confluence for combinations of typed $\lambda$-calculi with various kinds of rewriting do not seem to apply here. This is because the rewrite rules are not first-order. They can be frozen as above into a 'first-order part' and a 'substitution part', but the former may contain rules with $\beta$-redexes on their right-hand sides, which is not allowed for the systems studied in the literature. On the other hand, the method employed here seems to be quite flexible, since it makes use only of completeness of (typed) $\beta$. For example, the confluence result of [BTG89] should be an easy consequence.

COROLLARY 4.22 *Orthogonal HRSs are confluent.*

PROOF. Suppose we have two coinitial rewrites, $s \twoheadrightarrow_{\mathcal{H}} t_1$ and $s \twoheadrightarrow_{\mathcal{H}} t_2$, we can lift them by Theorem 4.10 to rewrites $\langle s\rangle \twoheadrightarrow_{\mathcal{C}\beta} \langle t_1\rangle$ and $\langle s\rangle \twoheadrightarrow_{\mathcal{C}\beta} \langle t_2\rangle$. Because $\mathcal{C}$ is an orthogonal CRS we can find by [Klo80, Thm. II.3.11] (or [Raa93]), convergent rewrites $\langle t_1\rangle \twoheadrightarrow_{\mathcal{C}} t'$ and $\langle t_2\rangle \twoheadrightarrow_{\mathcal{C}} t'$, for some $t'$. If we reduce $t'$ to $\beta$-normal form $r'$, then Lemma 4.21 says that the resulting rewrites can be simulated by rewrites $\langle t_1\rangle \twoheadrightarrow_{\mathcal{C}\beta} r'$ and $\langle t_2\rangle \twoheadrightarrow_{\mathcal{C}\beta} r'$. Finally, by Theorem 4.13 we can construct rewrites $t_1 \twoheadrightarrow_{\mathcal{H}} r$ and $t_2 \twoheadrightarrow_{\mathcal{H}} r$, such that $\langle r\rangle = r'$. □

The proof is expressed by the following diagram.




In fact, the reduction sequence $t' \twoheadrightarrow^{!}_{\beta} \langle r\rangle$, can be shown to be empty, giving a somewhat stronger result.

## 5. Discussion

We have shown two extensions of first-order rewriting to higher order, CRSs and HRSs, to be almost equivalent. The difference lies in the metalanguage used; they employ different flavours of the $\lambda$-calculus to generate their rewrite relations. For CRSs the underlined $\lambda$-calculus is used, while for HRSs the simply typed $\lambda$-calculus is used.

The translations from one system to the other are relatively simple because both are based on $\lambda$-calculus. The situation would be different for arbitrary meta-languages. But in fact it is hard to imagine a meta-language essentially different from $\lambda$-calculus. The basic steps of a rewrite (or redex-reaction) are: decomposing an object into a context and a redex, decomposing a redex into a pattern and a substitution, replacing the pattern with some other pattern, and then composing everything in the reverse order. The $\lambda$-calculus can be viewed as a 'calculus of (de)composing', so seems to be basic to any meta-language. If we look at other higher order rewrite formalisms, such as the Expression Reduction Systems of Khasidashvili [Kha90] and the Conditional Lambda Calculi of Takahashi [Tak], this claim seems to be supported. The precise interrelation is left to future work. We do note however that the similarities between these systems are obfuscated by the surprisingly large syntactical differences.

The work in this paper seems to suggest that only two basic properties are required for the flavour of $\lambda$-calculus one uses for the meta-language: confluence and termination. One can view CRSs and HRSs then as special cases of such a unifying theory of Higher Order Rewrite Systems (HORS). A large part of the syntactic rewrite theory should carry over to higher-order rewriting with more powerful meta-languages such as higher-order $\lambda$-calculi. This is left to future work.

## 6. Acknowledgements

We would like to thank Fer-Jan de Vries for comments on an earlier version of this paper. We have benefitted from discussions with and between Jan Willem Klop, Tobias Nipkow and Stefan Kahrs on this subject.

## References

[ACCL90] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In *Proceedings of the ACM Conference on Principles of Programming Languages*, San Francisco, 1990.

[Acz78] P. Aczel. A general Church-Rosser theorem. Technical report, University of Manchester, 1978.

[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. (Second printing 1985).

[BG93] M. Bezem and J.F. Groote, editors. *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, Utrecht, The Netherlands, March 1993. Springer-Verlag.

[BTG89] V. Breazu-Tannen and J. Gallier. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 137–150, 1989.

[Kah92] S. Kahrs. Context rewriting. In M. Rusinowitch and J.L. Rémy, editors, *Proceedings of the Third International Workshop on Conditional and Typed Rewriting Systems*, pages 21–35, 1992.

[Kha90] Z.O. Khasidashvili. Expression reduction systems. In *Proceedings of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220, Tbilisi, 1990.

[Klo80] J.W. Klop. *Combinatory Reduction Systems.* Mathematical Centre Tracts Nr. 127. CWI, Amsterdam, 1980. PhD Thesis.

[KOR93] J.W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems, introduction and survey. *Theoretical Computer Science*, 1993. To appear.

[Lan93] C. Laneve. *Optimality and Concurrency in Interaction Systems.* PhD thesis, dipartimento di informatica università di pisa, March 1993.

[LIC91] Amsterdam, The Netherlands. *Proceedings of the sixth annual IEEE Symposium on Logic in Computer Science*, Los Alamitos, July 1991. IEEE Computer Society Press.

[Mil] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In [Sie91].

[Nipa] T. Nipkow. Higher-order critical pairs. In [LIC91].

[Nipb] T. Nipkow. Orthogonal Higher-Order Rewrite Systems are Confluent. In [BG93].

[NM88] G. Nadathur and D. Miller. An overview of λProlog. In R.A. Kowalski and K.A. Bowen, editors, *Proc. 5th Int. Logic Programming Conference*, pages 810–827. MIT Press, 1988.

[Oos93] V. van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 1993. To appear.

[Pau90] L.C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–385. Academic Press, 1990.

[Raa93] F. van Raamsdonk. Confluence and superdevelopments. In C. Kirchner, editor, *Proceedings of the 5th International Conference on Rewrite Techniques and Applications*, 1993.

[Sie91] J. Siekmann, editor. *Extensions of Logic Programming*, volume 475 of *Lecture Notes in Artificial Intelligence.* Tübingen, FRG, Springer-Verlag, December 1991.

[Tak] M. Takahashi. λ-calculi with conditional rules. In [BG93].

[Wol93] D.A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1993.

## A. Induction Proofs of the CRS to HRS translation

This Appendix contains the 'trivial but long' induction-proofs of the various propositions.

Proposition 3.4 *Let $s$ be a CRS metaterm, $s' = \langle s \rangle$, then*

**a** *$s' : 0$, moreover there is a bijective correspondence between subterms of $s$ and subterms of type $0$ of $s'$,*

**b** *$s'$ is in long $\beta\eta$-normal form,*

**c** *if $s$ satisfies the pattern-condition, then $s'$ satisfies the pattern-condition w.r.t. translated metavariables,*

**d** $\langle \mathsf{Fvar}(s) \rangle = \mathsf{Fvar}(s')$.

Proof. The properties are proved simultaneously, by structural induction.

- If $s = x$, then $s' = x$, and we have

**a** by definition,

$$\frac{}{x : 0}$$

**b** because $x : 0$,
**c** obvious, $x$ is not a metavariable,
**d** $\langle\{x\}\rangle = \{x\} = \{\langle x\rangle\}$.

- If $s = [x]t$, then $s' = \Lambda\lambda x.\langle t\rangle$, so
  **a** by definition and induction hypothesis for $t$,

$$\frac{\dfrac{x : 0 \quad \langle t\rangle : 0}{\Lambda : (0 \to 0) \to 0 \quad \lambda x.\langle t\rangle : 0 \to 0}}{\Lambda\lambda x.\langle t\rangle : 0}$$

  The correspondence is the one induced by the correspondence between $t$ and $\langle t\rangle$ and by relating $s$ to $s'$. Both the subterms $\Lambda$ and $\lambda x.\langle t\rangle$ are not of type 0.
  **b** by induction hypothesis for $t$. Both $s' : 0$ and $\lambda x.\langle t\rangle : 0 \to 0$ are $\eta$-expanded.
  **c** by induction hypothesis for $t$. No new metavariables do occur.
  **d** $\langle \mathsf{Fvar}(t) - \{x\}\rangle = \mathsf{Fvar}(\langle t\rangle) - \{x\}$.
- If $s = F(t_1, \ldots, t_n)$, then $s' = F\langle t_1\rangle \ldots \langle t_n\rangle$
  **a** by definition and by induction hypothesis for $t_1$,... $t_n$,

$$\frac{F : 0 \to \ldots \to 0 \to 0 \quad \langle t_1\rangle : 0 \quad \ldots \quad \langle t_n\rangle : 0}{F\langle t_1\rangle \ldots \langle t_n\rangle : 0}$$

  The correspondence is the one induced by the correspondence between $t_1$,...,$t_n$ and $\langle t_1\rangle$,...,$\langle t_n\rangle$, relating $s$ to $s'$. Subterms $F\langle t_1\rangle \ldots \langle t_i\rangle$, for $i < n$, are not of type 0.
  **b** by induction hypothesis for $t_1$,...,$t_n$. $F\langle t_1\rangle \ldots \langle t_n\rangle : 0$ is $\eta$-expanded.
  **c** by induction hypothesis for $t_1$,...,$t_n$.
  **d**

$$\begin{aligned}\langle \mathsf{Fvar}(F(t_1, \ldots, t_n))\rangle &= \langle \mathsf{Fvar}(t_1) \cup \cdots \cup \mathsf{Fvar}(t_n)\rangle \\ &= \mathsf{Fvar}(\langle t_1\rangle) \cup \cdots \cup \mathsf{Fvar}(\langle t_n\rangle) \\ &= \mathsf{Fvar}(\langle F(t_1, \ldots, t_n)\rangle)\end{aligned}$$

- If $s = Z(t_1, \ldots, t_n)$, then $s' = z\langle t_1\rangle \ldots \langle t_n\rangle$, hence
  **a** by definition and induction hypothesis for $t_1$,... $t_n$,

$$\frac{z : 0 \to \ldots \to 0 \to 0 \quad \langle t_1\rangle : 0 \quad \ldots \quad \langle t_n\rangle : 0}{z\langle t_1\rangle \ldots \langle t_n\rangle : 0}$$

  The correspondence is obtained as in the previous case.
  **b** by induction hypothesis for $t_1$,...,$t_n$. $z\langle t_1\rangle \ldots \langle t_n\rangle : 0$ is $\eta$-expanded.
  **c** by induction hypothesis for $t_1$,...,$t_n$. If $Z(t_1, \ldots, t_n)$ satisfies the pattern-condition, then $z\langle t_1\rangle \ldots \langle t_n\rangle$ satisfies the pattern-condition, because distinctness of the variables is preserved by injectivity of $\langle\_\rangle$.
  **d**

$$\begin{aligned}\langle \mathsf{Fvar}(Z(t_1, \ldots, t_n))\rangle &= \langle \{Z\} \cup \mathsf{Fvar}(t_1) \cup \cdots \cup \mathsf{Fvar}(t_n)\rangle \\ &= \{z\} \cup \mathsf{Fvar}(\langle t_1\rangle) \cup \cdots \cup \mathsf{Fvar}(\langle t_n\rangle) \\ &= \mathsf{Fvar}(\langle Z(t_1, \ldots, t_n)\rangle)\end{aligned}$$

□

PROPOSITION 3.6
**a** $\langle C[t]\rangle = \langle C\rangle[\langle t\rangle]$

PROOF.
**a** The proof proceeds by induction on the structure of $C[\,]$.
- If $C[\,] = \Box$, then it is obvious.
- If $C[\,] = [x]C'[\,]$, then we have for every term $t$:
$$\begin{aligned}\langle C[t]\rangle &= \langle [x]C'[t]\rangle \\ &= \Lambda\lambda x.\langle C'[t]\rangle \\ &= \Lambda\lambda x.\langle C'\rangle[\langle t\rangle] \text{ (by induction hypothesis)} \\ &= \langle [x]C'\rangle[\langle t\rangle] \\ &= \langle C\rangle[\langle t\rangle]\end{aligned}$$
- If $C[\,]$ is of the form $F(s_1,\ldots,C'[\,],\ldots,s_n)$, then we have for every term $t$:
$$\begin{aligned}\langle C[t]\rangle &= \langle F(s_1,\ldots,C'[t],\ldots,s_n)\rangle \\ &= \langle F\rangle\langle s_1\rangle\ldots\langle C'[t]\rangle\ldots\langle s_n\rangle \\ &= \langle F\rangle\langle s_1\rangle\ldots\langle C'\rangle[\langle t\rangle]\ldots\langle s_n\rangle \text{ (by induction hypothesis)} \\ &= \langle C\rangle[\langle t\rangle]\end{aligned}$$
- If $C[\,]$ is of the form $Z(s_1,\ldots,C'[\,],\ldots,s_n)$, then we proceed as in the previous case.

□

PROPOSITION 3.7 *For every metaterm $t$ and assignment $\sigma$ we have $\langle t^\sigma\rangle = \langle t\rangle^{\langle\sigma\rangle}\downarrow_\beta$.*

PROOF. The proposition is proved by induction on the structure of the metaterm $t$.
- If $t = x$, then we have
$$\begin{aligned}\langle t^\sigma\rangle &= \langle x\rangle \\ &= x \\ &= x^{\langle\sigma\rangle} \\ &= \langle x\rangle^{\langle\sigma\rangle} \\ &= \langle x\rangle^{\langle\sigma\rangle}\downarrow_\beta \\ &= \langle t\rangle^{\langle\sigma\rangle}\downarrow_\beta\end{aligned}$$
- If $t = [x]t'$, then we have
$$\begin{aligned}\langle t^\sigma\rangle &= \langle([x]t')^\sigma\rangle \\ &= \langle[x]t'^\sigma\rangle \\ &= \Lambda\lambda x.\langle t'^\sigma\rangle \\ &= \Lambda\lambda x.(\langle t'\rangle^{\langle\sigma\rangle}\downarrow_\beta) \\ &= (\Lambda\lambda x.\langle t'\rangle^{\langle\sigma\rangle})\downarrow_\beta \\ &= (\Lambda\lambda x.\langle t'\rangle^{\langle\sigma\rangle})\downarrow_\beta \\ &= (\langle t\rangle^{\langle\sigma\rangle})\downarrow_\beta\end{aligned}$$
- If $t = F(t_1,\ldots,t_n)$, then we have
$$\langle t^\sigma\rangle = \langle(F(t_1,\ldots,t_n))^\sigma\rangle$$

$$\begin{aligned}
&= \langle F(t_1^\sigma, \ldots, t_n^\sigma)\rangle \\
&= F\langle t_1^\sigma\rangle \ldots \langle t_n^\sigma\rangle \\
&= F(\langle t_1\rangle^{\langle\sigma\rangle})\downarrow_\beta \ldots (\langle t_n\rangle^{\langle\sigma\rangle})\downarrow_\beta \\
&= (F(\langle t_1\rangle^{\langle\sigma\rangle}) \ldots (\langle t_n\rangle^{\langle\sigma\rangle}))\downarrow_\beta \\
&= (F\langle t_1\rangle \ldots \langle t_n\rangle^{\langle\sigma\rangle})\downarrow_\beta \\
&= \langle t\rangle^{\langle\sigma\rangle}\downarrow_\beta
\end{aligned}$$

- Finally the case that $t = Z(t_1, \ldots, t_n)$ is considered. Suppose $\sigma(Z) = \underline{\lambda}(x_1, \ldots, x_n).s$. Then we have

$$\begin{aligned}
\langle t^\sigma\rangle &= \langle s[x_1 := t_1 \ldots x_n := t_n]\rangle \\
&= \langle s\rangle[x_1 := \langle t_1\rangle \ldots x_n := \langle t_n\rangle] \\
&= (\lambda x_1 \ldots x_n.\langle s\rangle)\langle t_1^\sigma\rangle \ldots \langle t_n^\sigma\rangle\downarrow_\beta \\
&= \Big((\langle\sigma\rangle(z))(\langle t_1\rangle^{\langle\sigma\rangle}\downarrow_\beta) \ldots (\langle t_n\rangle^{\langle\sigma\rangle}\downarrow_\beta)\Big)\downarrow_\beta \\
&= \Big((\langle\sigma\rangle(z))(\langle t_1\rangle^{\langle\sigma\rangle}) \ldots (\langle t_n\rangle^{\langle\sigma\rangle})\Big)\downarrow_\beta \\
&= (z\langle t_1\rangle \ldots \langle t_n\rangle^{\langle\sigma\rangle})\downarrow_\beta \\
&= \langle Z(t_1, \ldots, t_n)\rangle^{\langle\sigma\rangle}\downarrow_\beta
\end{aligned}$$

□

## B. Induction Proofs of the HRS to CRS translation

Proposition 4.5 *If $t$ is a HRS term in long $\beta\eta$-normal form, then $\langle t\rangle$ is well-defined as a CRS term.*

Proof. The proof proceeds by induction on the structure of a long $\beta\eta$-normal form. If $t$ is in long $\beta\eta$-normal form, then $t$ is of the form $\lambda x_1 \ldots x_m.st_1 \ldots t_n$ with $s$ a symbol standing for a variable or an operator, and $t_1, \ldots, t_n$ terms in long $\beta\eta$-normal form. If $t = \lambda x_1 \ldots x_m.xt_1 \ldots t_n$, then we have (since $t$ is in long $\beta\eta$-normal form), that $x : \tau$ with $\mathsf{Ar}(\tau) = n$. Then,

$$\begin{aligned}
\langle t\rangle &= \langle \lambda x_1 \ldots x_m.xt_1 \ldots t_n\rangle \\
&= [x_1 \ldots x_m]\langle x\rangle\langle t_1\rangle \ldots \langle t_n\rangle \\
&= [x_1 \ldots x_m]\underline{\lambda} y_1 \ldots y_n.@_n(x, y_1, \ldots, y_n)\langle t_1\rangle \ldots \langle t_n\rangle \\
&= [x_1 \ldots x_m]@_n(x, \langle t_1\rangle, \ldots, \langle t_n\rangle)
\end{aligned}$$

The terms $t_1, \ldots, t_n$ are in long $\beta\eta$-normal form, so by induction hypothesis $\langle t_1\rangle, \ldots, \langle t_n\rangle$ are well-formed. This yields that $\langle t\rangle$ is a well-formed CRS term. Second, if $t = \lambda x_1 \ldots x_n.Ft_1 \ldots t_n$, then we have $F : \tau$ with $\mathsf{Ar}(\tau) = n$ since $t$ is in long $\beta\eta$-normal form. Then we have

$$\begin{aligned}
\langle t\rangle &= \langle \lambda x_1 \ldots x_m.Ft_1 \ldots t_n\rangle \\
&= [x_1 \ldots x_m]\langle F\rangle\langle t_1\rangle \ldots \langle t_n\rangle \\
&= [x_1 \ldots x_m]\underline{\lambda} y_1 \ldots y_n.F(y_1, \ldots, y_n)\langle t_1\rangle \ldots \langle t_n\rangle \\
&= [x_1 \ldots x_m]F(\langle t_1\rangle, \ldots, \langle t_n\rangle)
\end{aligned}$$

By induction hypothesis, this is a well-formed CRS term. □

Proposition 4.7 $\langle C[t]\rangle = \langle C\rangle[\langle t\rangle]$.

PROOF. The proof proceeds by induction on the definition of $C[\,]$. If $C[\,] = \Box$, then it's trivial. If $C[\,]$ is an abstraction term, then we distinguish three possibilities. If $C[\,] = \lambda x_1 \ldots x_m.\Box$ then again it's trivial. If $C[\,] = \lambda x_1 \ldots x_m.xt_1 \ldots C'[\,] \ldots t_n$, then

$$\begin{aligned}
\langle C[t]\rangle &= \langle \lambda x_1 \ldots x_m.xt_1 \ldots C'[t] \ldots t_n\rangle \\
&= [x_1 \ldots x_m]@_n(x, \langle t_1\rangle, \ldots, \langle C'[t]\rangle, \ldots, \langle t_n\rangle) \\
&= [x_1 \ldots x_m]@_n(x, \langle t_1\rangle, \ldots, \langle C'\rangle[\langle t\rangle], \ldots, \langle t_n\rangle) \\
&= \langle C\rangle[\langle t\rangle]
\end{aligned}$$

If $C[\,] = \lambda x_1 \ldots x_m.Ft_1 \ldots C'[\,] \ldots t_n$, then

$$\begin{aligned}
\langle C[t]\rangle &= \langle \lambda x_1 \ldots x_m.Ft_1 \ldots C'[\,] \ldots t_n\rangle \\
&= [x_1 \ldots x_m]F(\langle t_1\rangle, \ldots, \langle C'[t]\rangle, \ldots, \langle t_n\rangle) \\
&= [x_1 \ldots x_m]F(\langle t_1\rangle, \ldots, \langle C'\rangle[\langle t\rangle], \ldots, \langle t_n\rangle) \\
&= \langle C\rangle[\langle t\rangle]
\end{aligned}$$

□

PROPOSITION 4.8

**a** *Let $s$ be a term in long $\beta\eta$-normal form, and $u_1, \ldots, u_n$ terms that are $\eta$-equivalent to distinct variables. Then $\langle s\rangle[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}] = \langle s[z_1 := u_1 \ldots z_n := u_n]{\downarrow_\beta}\rangle$.*

**b** $\langle l\rangle^{\langle\sigma\rangle} = \langle l^\sigma{\downarrow_\beta}\rangle$.

PROOF.

**a** The proof proceeds by induction on the structure of $s$.

- If $s = \lambda x_1 \ldots x_m.xs_1 \ldots s_k$ with $x$ a variable and $x \neq z_j$ for $j = 1, \ldots, n$, then
$$\begin{aligned}
\langle s\rangle[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}] &= [x_1 \ldots x_m]@(x, \langle s_1\rangle, \ldots, \langle s_k\rangle)[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}] \\
&= [x_1 \ldots x_m]@(x, \langle s_1\rangle[\ldots], \ldots, \langle s_k\rangle[\ldots]) \\
&= [x_1 \ldots x_m]@(x, \langle s_1[\ldots]{\downarrow_\beta}\rangle, \ldots, \langle s_k[\ldots]{\downarrow_\beta}\rangle) \\
&= \langle \lambda x_1 \ldots x_m.x(s_1[\ldots]{\downarrow_\beta}) \ldots (s_k[\ldots]{\downarrow_\beta})\rangle \\
&= \langle \lambda x_1 \ldots x_m.x(s_1[\ldots]) \ldots (s_k[\ldots]){\downarrow_\beta}\rangle \\
&= \langle \lambda x_1 \ldots x_m.xs_1 \ldots s_k[\ldots]{\downarrow_\beta}\rangle \\
&= \langle s[z_1 := u_1 \ldots z_n := u_n]{\downarrow_\beta}\rangle
\end{aligned}$$
- If $s = \lambda x_1 \ldots x_m.z_j s_1 \ldots s_k$, then
$$\begin{aligned}
\langle s\rangle[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}] &= \langle \lambda x_1 \ldots x_m.z_j s_1 \ldots s_k\rangle[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}] \\
&= [x_1 \ldots x_m]@(z_j, \langle s_1\rangle, \ldots, \langle s_k\rangle)[\ldots] \\
&= [x_1 \ldots x_m]@(u_j{\downarrow_\eta}, \langle s_1\rangle[\ldots], \ldots, \langle s_k\rangle[\ldots]) \\
&= [x_1 \ldots x_m]@(u_j{\downarrow_\eta}, \langle s_1[\ldots]{\downarrow_\beta}\rangle, \ldots, \langle s_k[\ldots]{\downarrow_\beta}\rangle) \\
&= \langle \lambda x_1 \ldots x_m.u_j(s_1[\ldots]) \ldots (s_k[\ldots]){\downarrow_\beta}\rangle \\
&= \langle \lambda x_1 \ldots x_m.u_j s_1 \ldots s_k[\ldots]{\downarrow_\beta}\rangle \\
&= \langle \lambda x_1 \ldots x_m.z_j s_1 \ldots s_k[\ldots]{\downarrow_\beta}\rangle \\
&= \langle s[z_1 := u_1 \ldots z_n := u_n]{\downarrow_\beta}\rangle
\end{aligned}$$
Note that we must have $u_j = \lambda y_1 \ldots y_k.(u_j{\downarrow_\eta})y_1 \ldots y_k$.
- If $s = \lambda x_1 \ldots x_m.Fs_1 \ldots s_k$ with $F$ an operator symbol, then
$$\langle s\rangle[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}] = \langle \lambda x_1 \ldots x_m.Fs_1 \ldots s_k\rangle[z_1 := u_1{\downarrow_\eta} \ldots z_n := u_n{\downarrow_\eta}]$$

$$\begin{aligned}
&= [x_1 \ldots x_m]F(\langle s_1\rangle, \ldots, \langle s_k\rangle)[\ldots] \\
&= [x_1 \ldots x_m]F(\langle s_1\rangle[\ldots], \ldots, \langle s_k\rangle[\ldots]) \\
&= [x_1 \ldots x_m]F(\langle s_1[\ldots]\downarrow_\beta\rangle, \ldots, \langle s_k[\ldots]\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \ldots x_m.F(s_1[\ldots]\downarrow_\beta) \ldots (s_k[\ldots]\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \ldots x_m.F(s_1[\ldots]) \ldots (s_k[\ldots])\downarrow_\beta\rangle \\
&= \langle \lambda x_1 \ldots x_m.F s_1 \ldots s_k[\ldots]\downarrow_\beta\rangle \\
&= \langle s[z_1 := u_1 \ldots z_n := u_n]\downarrow_\beta\rangle
\end{aligned}$$

**b** The is proved by induction on the structure of a long $\beta\eta$-normal form, in which arguments of free variables are $\eta$-equivalent to distinct bound variables.

- If $l = \lambda x_1 \ldots x_m.x t_1 \ldots t_n$ with $x$ a bound variable, then
$$\begin{aligned}
\langle l\rangle^{\langle\sigma\rangle} &= \langle \lambda x_1 \ldots x_m.x t_1 \ldots t_n\rangle^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]@_n(x, \langle t_1\rangle, \ldots, \langle t_n\rangle)^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_n]@_n(x, \langle t_1\rangle^{\langle\sigma\rangle}, \ldots, \langle t_n\rangle^{\langle\sigma\rangle}) \\
&= [x_1 \ldots x_m]@_n(x, \langle t_1^\sigma\downarrow_\beta\rangle, \ldots, \langle t_n^\sigma\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \ldots x_m.x(t_1^\sigma\downarrow_\beta) \ldots (t_n^\sigma\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \ldots x_m.x(t_1^\sigma) \ldots (t_n^\sigma)\downarrow_\beta\rangle \\
&= \langle (\lambda x_1 \ldots x_m.x t_1 \ldots t_n)^\sigma\downarrow_\beta\rangle
\end{aligned}$$
- If $l = \lambda x_1 \ldots x_m.F t_1 \ldots t_n$ with $F$ an operator symbol, then
$$\begin{aligned}
\langle l\rangle^{\langle\sigma\rangle} &= \langle \lambda x_1 \ldots x_m.F t_1 \ldots t_n\rangle^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]F(\langle t_1\rangle, \ldots, \langle t_n\rangle)^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]F(\langle t_1\rangle^{\langle\sigma\rangle}, \ldots, \langle t_n\rangle^{\langle\sigma\rangle}) \\
&= [x_1 \ldots x_m]F(\langle t_1^\sigma\downarrow_\beta\rangle, \ldots, \langle t_n^\sigma\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \ldots x_m.F(t_1^\sigma\downarrow_\beta) \ldots (t_n^\sigma\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \ldots x_m.F(t_1^\sigma) \ldots (t_n^\sigma)\downarrow_\beta\rangle \\
&= \langle (\lambda x_1 \ldots x_m.F t_1 \ldots t_n)^\sigma\downarrow_\beta\rangle
\end{aligned}$$
- If $l = \lambda x_1 \ldots x_m.z t_1 \ldots t_n$ with $z$ a free variable, then $z : \tau$ with $\mathsf{Ar}(\tau) = n$. Suppose $\sigma(z) = \lambda y_1 \ldots y_n.s$, then $\langle\sigma\rangle(Z) = \underline{\lambda}(y_1, \ldots, y_n).\langle s\rangle$. Then we have
$$\begin{aligned}
\langle l\rangle^{\langle\sigma\rangle} &= \langle \lambda x_1 \ldots x_m.z t_1 \ldots t_n\rangle^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]Z(t_1\downarrow_\eta, \ldots, t_n\downarrow_\eta)^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]\langle s\rangle[y_1 := t_1\downarrow_\eta \ldots y_n := t_n\downarrow_\eta] \\
&= [x_1 \ldots x_m]\langle s[y_1 := t_1 \ldots y_n := t_n]\downarrow_\beta\rangle \\
&= \langle (\lambda x_1 \ldots x_m.z t_1 \ldots t_n)^\sigma\downarrow_\beta\rangle
\end{aligned}$$

□

Proposition 4.9

**a** *Let $s$ and $u_1, \ldots, u_n$ be terms in long $\beta\eta$-normal form. Then we have $\langle s\rangle[z_1 := \langle u_1\rangle \ldots z_n := \langle u_n\rangle] \to_\beta^! \langle s[z_1 := u_1 \ldots z_n := u_n]\downarrow_\beta\rangle$.*

**b** $\langle r\rangle^{\langle\sigma\rangle} \to_\beta^! \langle r^\sigma\downarrow_\beta\rangle$.

Proof.

**a** The proof proceeds by induction on the maximal length of the $\beta$- reduction of $s[z_1 := u_1 \ldots z_1 := u_n]$ to normal form. In the base case, the maximal length of the reduction sequence is 0. In this case, there are three possibilities for the form of $s$:

- $s = \lambda x_1 \dots x_m.xs_1 \dots s_k$ with $x \neq z_i$ for $i = 1, \dots, n$,
- $s = \lambda x_1 \dots x_m.Fs_1 \dots s_k$,
- $s = \lambda x_1 \dots x_m.z_j$

Note that if $s$ has a head-variable that is among the $z_j$, the type of $z_j$ is a base type, because otherwise a $\beta$-reduction for performing the substitution would be needed. The proof of the base case proceeds by induction on the structure of $s$.

- In the first case, if $s = \lambda x_1 \dots x_m.xs_1 \dots s_k$ with $x \neq z_i$ for $i = 1, \dots, n$, then we have

$$\begin{aligned}
\langle s\rangle[z_1 := \langle u_1\rangle \dots z_n := \langle u_n\rangle] &= \langle \lambda x_1 \dots x_m.xs_1 \dots s_k\rangle[\dots] \\
&= [x_1 \dots x_m]@_k(x, \langle s_1\rangle, \dots, \langle s_k\rangle)[\dots] \\
&= [x_1 \dots x_m]@_k(x, \langle s_1\rangle[\dots], \dots, \langle s_k\rangle[\dots]) \\
&= [x_1 \dots x_m]@_k(x, \langle s_1[\dots]\downarrow_\beta\rangle, \dots, \langle s_k[\dots]\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \dots x_m.x(s_1[\dots]\downarrow_\beta) \dots (s_k[\dots]\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \dots x_m.x(s_1[\dots]) \dots (s_k[\dots])\downarrow_\beta\rangle \\
&= \langle \lambda x_1 \dots x_m.xs_1 \dots s_k[\dots]\downarrow_\beta\rangle \\
&= \langle s[z_1 := u_1 \dots z_n := u_n]\downarrow_\beta\rangle
\end{aligned}$$

- If $s = \lambda x_1 \dots x_m.Fs_1 \dots s_k$, then the statement follows directly by induction hypothesis.
- If $s = \lambda x_1 \dots x_m.z_j$, then we have

$$\begin{aligned}
\langle s\rangle[z_1 := \langle u_1\rangle \dots z_n := \langle u_n\rangle] &= \langle \lambda x_1 \dots x_m.z_j\rangle[z_1 := \langle u_1\rangle \dots z_n := \langle u_n\rangle] \\
&= [x_1 \dots x_m]z_j[z_1 := \langle u_1\rangle \dots z_n := \langle u_n\rangle] \\
&= [x_1 \dots x_m]\langle u_j\rangle \\
&= \langle \lambda x_1 \dots x_m.z_j[z_1 := u_1 \dots z_n := u_n]\downarrow_\beta\rangle
\end{aligned}$$

In the induction step we will consider the case that the length of the maximal $\beta$-reduction sequence of $s[z_1 := u_1 \dots z_1 := u_n]$ to normal form is greater than 0. Again there are three different possibilities for the form of $s$:

- $s = \lambda x_1 \dots x_m.z_j s_1 \dots s_k$ for some $j \in \{1, \dots, n\}$
- $s = \lambda x_1 \dots x_m.xs_1 \dots s_k$ with $x \neq z_j$ for $j = 1, \dots k$
- $s = \lambda x_1 \dots x_k.Fs_1 \dots s_k$

Like the proof in the base case, the proof of the induction step proceeds by induction on the structure of $s$.

- If $s = \lambda x_1 \dots x_m.z_j s_1 \dots s_k$, then we have

$$\begin{aligned}
\langle s\rangle[z_1 := \langle u_1\rangle \dots z_n := \langle u_n\rangle] &= \langle \lambda x_1 \dots x_m.z_j s_1 \dots s_k\rangle[\dots] \\
&= [x_1 \dots x_m]@_k(z_j, \langle s_1\rangle, \dots, \langle s_k\rangle)[\dots] \\
&= [x_1 \dots x_m]@_k(\langle u_j\rangle, \langle s_1\rangle[\dots], \dots, \langle s_k\rangle[\dots]) \\
&= [x_1 \dots x_m]@_k(\langle \lambda y_1 \dots y_k.u'_j\rangle, \langle s_1\rangle[\dots], \dots, \langle s_k\rangle[\dots]) \\
&\twoheadrightarrow_\beta [x_1 \dots x_m]@_k(\langle \lambda y_1 \dots y_k.u'_j\rangle, \langle s_1[\dots]\downarrow_\beta\rangle, \dots, \langle s_k[\dots]\downarrow_\beta\rangle) \\
&= [x_1 \dots x_m]@_k([y_1 \dots y_k]\langle u'_j\rangle, \langle s_1[\dots]\downarrow_\beta\rangle, \dots, \langle s_k[\dots]\downarrow_\beta\rangle) \\
&\to_\beta [x_1 \dots x_m]\langle u'_j\rangle[y_1 := \langle s_1[\dots]\downarrow_\beta\rangle \dots y_k := \langle s_k[\dots]\downarrow_\beta\rangle] \\
&\to^!_\beta [x_1 \dots x_m]\langle u'_j[y_1 := s_1[\dots]\downarrow_\beta \dots y_k := s_k[\dots]\downarrow_\beta]\downarrow_\beta\rangle \\
&= [x_1 \dots x_m]\langle u'_j[y_1 := s_1[\dots] \dots y_k := s_k[\dots]]\downarrow_\beta\rangle \\
&= [x_1 \dots x_m]\langle (\lambda y_1 \dots y_k.u'_j)s_1[\dots] \dots s_k[\dots]\downarrow_\beta\rangle \\
&= [x_1 \dots x_m]\langle z_j s_1 \dots s_k[\dots]\downarrow_\beta\rangle \\
&= \langle \lambda x_1 \dots x_m.z_j s_1 \dots s_k[\dots]\downarrow_\beta\rangle
\end{aligned}$$

$$= \quad \langle s[\ldots]\downarrow_\beta\rangle$$

- If $s = \lambda x_1 \ldots x_m.xs_1 \ldots s_k$ with $x \neq z_j$ for $j = 1, \ldots k$, then the statement follows by induction hypothesis:

$$\begin{aligned}
\langle s\rangle[z_1 := \langle u_1\rangle \ldots z_n := \langle u_n\rangle] &= \langle \lambda x_1 \ldots x_m.xs_1 \ldots s_k\rangle[\ldots] \\
&= [x_1 \ldots x_m]@_k(x, \langle s_1\rangle, \ldots, \langle s_k\rangle)[\ldots] \\
&= [x_1 \ldots x_m]@_k(x, \langle s_1\rangle[\ldots], \ldots, \langle s_k\rangle[\ldots]) \\
&\to^!_\beta [x_1 \ldots x_m]@_k(x, \langle s_1[\ldots]\downarrow_\beta\rangle, \ldots, \langle s_k[\ldots]\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \ldots x_m.x(s_1[\ldots]\downarrow_\beta) \ldots (s_k[\ldots]\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \ldots x_m.x(s_1[\ldots]) \ldots (s_k[\ldots])\downarrow_\beta\rangle \\
&= \langle \lambda x_1 \ldots x_m.xs_1 \ldots s_k[\ldots]\downarrow_\beta\rangle \\
&= \langle s[\ldots]\downarrow_\beta\rangle
\end{aligned}$$

- If $s = \lambda x_1 \ldots x_m.Fs_1 \ldots s_k$, then we have by induction

$$\begin{aligned}
\langle s\rangle[z_1 := \langle u_1\rangle \ldots z_n := \langle u_n\rangle] &= \langle \lambda x_1 \ldots x_m.Fs_1 \ldots s_k\rangle[\ldots] \\
&= [x_1 \ldots x_m]F(\langle s_1\rangle, \ldots, \langle s_k\rangle)[\ldots] \\
&= [x_1 \ldots x_m]F(\langle s_1\rangle[\ldots], \ldots, \langle s_k\rangle[\ldots]) \\
&\to^!_\beta [x_1 \ldots x_m]F(\langle s_1[\ldots]\downarrow_\beta\rangle, \ldots, \langle s_k[\ldots]\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \ldots x_m.F(s_1[\ldots]\downarrow_\beta) \ldots (s_k[\ldots]\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \ldots x_m.F(s_1[\ldots]) \ldots (s_k[\ldots])\downarrow_\beta\rangle \\
&= \langle \lambda x_1 \ldots x_m.Fs_1 \ldots s_k[\ldots]\downarrow_\beta\rangle \\
&= \langle s[\ldots]\downarrow_\beta\rangle
\end{aligned}$$

**b** The proof proceeds by induction on the structure of $r$.

- If $r = \lambda x_1 \ldots x_m.zt_1 \ldots t_n$, then we have

$$\begin{aligned}
\langle r\rangle^{\langle\sigma\rangle} &= [x_1 \ldots x_m]Z(\langle t_1\rangle, \ldots, \langle t_n\rangle)^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]\langle s\rangle[z_1 := \langle t_1\rangle^{\langle\sigma\rangle} \ldots z_n := \langle t_n\rangle^{\langle\sigma\rangle}] \\
&\to\!\!\!\!\to_\beta [x_1 \ldots x_m]\langle s\rangle[z_1 := \langle t_1^\sigma\downarrow_\beta\rangle \ldots z_n := \langle t_n^\sigma\downarrow_\beta\rangle] \\
&\to^!_\beta [x_1 \ldots x_m]\langle s[z_1 := t_1^\sigma\downarrow_\beta \ldots z_n := t_n^\sigma\downarrow_\beta]\downarrow_\beta\rangle \\
&= [x_1 \ldots x_m]\langle s[z_1 := t_1^\sigma \ldots z_n := t_n^\sigma]\downarrow_\beta\rangle \\
&= \langle \lambda x_1 \ldots x_m.s[z_1 := t_1^\sigma \ldots z_n := t_n^\sigma]\downarrow_\beta\rangle \\
&= \langle (\lambda x_1 \ldots x_m.zt_1 \ldots t_n)^\sigma\downarrow_\beta\rangle \\
&= \langle r^\sigma\downarrow_\beta\rangle
\end{aligned}$$

- If $r = \lambda x_1 \ldots x_m.xt_1 \ldots t_n$, then we have

$$\begin{aligned}
\langle r\rangle^{\langle\sigma\rangle} &= [x_1 \ldots x_m]@_n(x, \langle t_1\rangle, \ldots, \langle t_n\rangle^{\langle\sigma\rangle}) \\
&= [x_1 \ldots x_m]@_n(x, \langle t_1\rangle^{\langle\sigma\rangle}, \ldots, \langle t_n\rangle^{\langle\sigma\rangle}) \\
&\to^!_\beta [x_1 \ldots x_m]@_n(x, \langle t_1^\sigma\downarrow_\beta\rangle, \ldots, \langle t_n^\sigma\downarrow_\beta\rangle) \\
&= \langle \lambda x_1 \ldots x_m.x(t_1^\sigma\downarrow_\beta) \ldots (t_n^\sigma\downarrow_\beta)\rangle \\
&= \langle \lambda x_1 \ldots x_m.x(t_1^\sigma) \ldots (t_n^\sigma)\downarrow_\beta\rangle \\
&= \langle (\lambda x_1 \ldots x_m.xt_1 \ldots t_n)^\sigma\downarrow_\beta\rangle \\
&= \langle r^\sigma\downarrow_\beta\rangle
\end{aligned}$$

- If $r = \lambda x_1 \ldots x_m.Ft_1 \ldots t_n$, then we have

$$\begin{aligned}
\langle r\rangle^{\langle\sigma\rangle} &= [x_1 \ldots x_m]F(\langle t_1\rangle, \ldots, \langle t_n\rangle)^{\langle\sigma\rangle} \\
&= [x_1 \ldots x_m]F(\langle t_1\rangle^{\langle\sigma\rangle}, \ldots, \langle t_n\rangle^{\langle\sigma\rangle})
\end{aligned}$$

$$
\begin{array}{rl}
\rightarrow_\beta^! & [x_1 \ldots x_m]F(t_1^\sigma\downarrow_\beta, \ldots, t_n^\sigma\downarrow_\beta) \\
= & \langle \lambda x_1 \ldots x_m.F(t_1^\sigma\downarrow_\beta) \ldots (t_n^\sigma\downarrow_\beta)\rangle \\
= & \langle \lambda x_1 \ldots x_m.F(t_1^\sigma) \ldots (t_n^\sigma)\downarrow_\beta\rangle \\
= & \langle (\lambda x_1 \ldots x_m.F t_1 \ldots t_n)^\sigma\downarrow_\beta\rangle \\
= & \langle r^\sigma\downarrow_\beta\rangle
\end{array}
$$

□