Reductions in Higher-Order Rewriting and Their Equivalence

Pablo Barenbaum
National University of Quilmes (CONICET), Bernal, Argentina
University of Buenos Aires, Argentina

Eduardo Bonelli
Stevens Institute of Technology, Hoboken, NJ, USA

Abstract

Proof terms are syntactic expressions that represent computations in term rewriting. They were introduced by Meseguer and exploited by van Oostrom and de Vrijer to study *equivalence of reductions* in (left-linear) first-order term rewriting systems. We study the problem of extending the notion of proof term to *higher-order rewriting*, which generalizes the first-order setting by allowing terms with binders and higher-order substitution. In previous works that devise proof terms for higher-order rewriting, such as Bruggink's, it has been noted that the challenge lies in reconciling composition of proof terms and higher-order substitution (β-equivalence). This led Bruggink to reject "nested" composition, other than at the outermost level. In this paper, we propose a notion of higher-order proof term we dub *rewrites* that supports nested composition. We then define *two* notions of equivalence on rewrites, namely *permutation equivalence* and *projection equivalence*, and show that they coincide.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Type theory

Keywords and phrases Term Rewriting, Higher-Order Rewriting, Proof terms, Equivalence of Computations

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.8

Related Version *Extended Version*: `https://arxiv.org/abs/2210.15654` [2]

Funding *Pablo Barenbaum*: Partially supported by project ECOS-Sud A17C01.

1 Introduction

Term rewriting systems model computation as sequences of steps between terms, *reduction sequences*, where steps are instances of term rewriting rules [15]. It is natural to consider reduction sequences up to swapping of orthogonal steps since such reductions perform the "same work". The ensuing notion of equivalence is called *permutation equivalence* and was first studied by Lévy [11] in the setting of the λ-calculus but has appeared in other guises connected with concurrency [15, Rem.8.1.1]. As an example, consider the rewrite rule $\mathbf{c}(x, \mathbf{f}(y)) \rightarrow \mathbf{d}(x, x)$ and the following reduction sequence where, in each step, the contracted redex is underlined:

$$\underline{\mathbf{c}(\mathbf{c}(z,\mathbf{f}(z)),\mathbf{f}(z))} \rightarrow \mathbf{d}(\underline{\mathbf{c}(z,\mathbf{f}(z))},\mathbf{c}(z,\mathbf{f}(z))) \rightarrow \mathbf{d}(\mathbf{d}(z,z),\underline{\mathbf{c}(z,\mathbf{f}(z))}) \rightarrow \mathbf{d}(\mathbf{d}(z,z),\mathbf{d}(z,z)) \tag{1}$$

Performing the innermost redex first, rather than the outermost one, leads to:

$$\mathbf{c}(\underline{\mathbf{c}(z,\mathbf{f}(z))},\mathbf{f}(z)) \rightarrow \underline{\mathbf{c}(\mathbf{d}(z,z),\mathbf{f}(z))} \rightarrow \mathbf{d}(\mathbf{d}(z,z),\mathbf{d}(z,z)) \tag{2}$$

The first step in (1) makes two copies of the innermost redex. It is the two steps contracting these copies that are swapped with the first one in (1) to produce (2). Such duplication (and erasure) contribute most of the complications behind permutation equivalence, both in its formulation and the study of its properties.

Proof Terms. *Proof terms* are a natural representation for computations. They were introduced by Meseguer as a means of representing proofs in Rewriting Logic [13] and exploited by van Oostrom and de Vrijer in the setting of first-order left-linear rewriting systems, to study equivalence of reductions in [17] and [15, Chapter 9]. Rewrite rules are assigned *rule symbols* denoting the application of a rewriting rule. Proof terms are expressions built using function symbols, a binary operator "**;**" denoting sequential composition of proof terms, and rule symbols. Assuming the following rule symbol for our rewrite rule $\varrho(x, y) : \mathbf{c}(x, \mathbf{f}(y)) \rightarrow \mathbf{d}(x, x)$, reduction (1) may be represented as the proof term: $\varrho(\mathbf{c}(z, \mathbf{f}(z)), z) \mathbin{\mathbf{;}} \mathbf{d}(\varrho(z, z), \mathbf{c}(z, \mathbf{f}(z))) \mathbin{\mathbf{;}} \mathbf{d}(\mathbf{d}(z, z), \varrho(z, z))$ and reduction (2) as the proof term: $\mathbf{c}(\varrho(z, z), \mathbf{f}(z)) \mathbin{\mathbf{;}} \varrho(\mathbf{d}(z, z), z)$. One notable feature of proof terms is that they support parallel steps. For instance, both proof terms above are permutation equivalent to $\varrho(\mathbf{c}(z, \mathbf{f}(z)), z) \mathbin{\mathbf{;}} \mathbf{d}(\varrho(z, z), \varrho(z, z))$, which performs the two last steps in parallel, as well as to $\varrho(\varrho(z, z), z)$, which performs all steps simultaneously. Permutation equivalence now can be studied in terms of equational theories on proof terms.

Equivalence of Reductions via Proof Terms for First-Order Rewriting. In [17], van Oostrom and de Vrijer characterize permutation equivalence of proof terms in four alternative ways. First, they formulate an equational theory of permutation equivalence $\rho \approx \sigma$ between proof terms, such that for example $\varrho(\mathbf{c}(z, \mathbf{f}(z)), z) \mathbin{\mathbf{;}} \mathbf{d}(\varrho(z, z), \varrho(z, z)) \approx \varrho(\varrho(z, z), z)$ holds. These equations account for the behavior of proof term composition, which has a monoidal structure, in the sense that composition is associative and *empty* steps act as identities. Second, they define an operation of *projection* ρ/σ, denoting the computational work that is left of ρ after σ. For example, $\mathbf{c}(\varrho(z, z), \mathbf{f}(z))/\varrho(\mathbf{c}(z, \mathbf{f}(z)), z) = \mathbf{d}(\varrho(z, z), \varrho(z, z))$. This induces a notion of *projection equivalence* between proof terms ρ and σ, declared to hold when both ρ/σ and σ/ρ are empty, *i.e.* they contain no rule symbols. Third, they define a *standardization procedure* to reorder the steps of a reduction in outside-in order, mapping each proof term ρ to a proof term ρ^* in *standard form*. For example, the (parallel) standard form of $\mathbf{c}(\varrho(z, z), \mathbf{f}(z)) \mathbin{\mathbf{;}} \varrho(\mathbf{d}(z, z), z)$ is $\varrho(\mathbf{c}(z, \mathbf{f}(z)), z) \mathbin{\mathbf{;}} \mathbf{d}(\varrho(z, z), \varrho(z, z))$. This induces a notion of *standardization equivalence* between proof terms ρ and σ, declared to hold when $\rho^* = \sigma^*$. Fourth, they define a notion of *labelling equivalence*, based on lifting computational steps to labelled terms. Although these notions of equivalence were known prior to [17], the main result of that paper is that they are systematically studied using proof terms and, moreover, shown to coincide.

Higher-Order Rewriting. Higher-order term rewriting (HOR) generalizes first-order term rewriting by allowing binders. Function symbols are generalized to constants of any given simple type, and first-order terms are generalized to simply-typed λ-terms, including constants and up to $\beta\eta$-equivalence. The paradigmatic example of a higher-order rewriting system is the λ-calculus. It includes a base type ι and two constants $\mathbf{app} : \iota \to \iota \to \iota$ and $\mathbf{lam} : (\iota \to \iota) \to \iota$; β-reduction may be expressed as the higher-order rewrite rule $\mathbf{app}\,(\mathbf{lam}\,(\lambda z.x\,z))\,y \rightarrow x\,y$. A sample reduction sequence is:

$$\mathbf{lam}(\lambda v.\mathbf{app}(\mathbf{lam}(\lambda x.x), \underline{\mathbf{app}(\mathbf{lam}(\lambda w.w), v)})) \rightarrow \mathbf{lam}(\lambda v.\underline{\mathbf{app}(\mathbf{lam}(\lambda x.x), v)}) \rightarrow \mathbf{lam}(\lambda v.v) \quad (3)$$

Generalizing proof terms to the setting of higher-order rewriting is a natural goal. Just like in the first-order case, we assign rule symbols to rewrite rules. One would then expect to obtain proof terms by adding these rule symbols and the "$;$" composition operator to the simply typed λ-calculus. If we assume the following rule symbol for our rewrite rule $\varrho\, x\, y : \mathbf{app}\,(\mathbf{lam}\,(\lambda z.x\, z))\, y \rightarrow x\, y$, then an example of a higher-order proof term for (3) is:

$$\mathbf{lam}\Big(\lambda v.\big(\mathbf{app}(\mathbf{lam}(\lambda x.x), \varrho\,(\lambda w.w)\, v)\ ;\ \varrho\,(\lambda u.u)\, v\big)\Big)$$

However, higher-order substitution and proof term composition seem not to be in consonance, an issue already observed by Bruggink [4]. Consider a variable x. This variable itself denotes an empty computation $x \rightarrow x$, so the composition $(x\ ;\ x)$ also denotes an empty computation $x \rightarrow x$. If σ is an arbitrary proof term $s \rightarrow t$, the proof term $(\lambda x.(x\ ;\ x))\,\sigma$ should, in principle, represent a computation $(\lambda x.x)\, s \rightarrow (\lambda x.x)\, t$. This is the same as $s \rightarrow t$, because terms are regarded up to $\beta\eta$-equivalence. The challenge lies in lifting $\beta\eta$-equivalence to the level of proof terms: if β-reduction is naively extended to operate on proof terms, the well-formed proof term $(\lambda x.(x\ ;\ x))\,\sigma$ becomes equal to $(\sigma\ ;\ \sigma)$, which is ill-formed because σ is not composable with itself if $s \neq_{\beta\eta} t$. Rather than simply disallowing the use of "$;$" under applications and abstractions (the route taken in [4]), our aim is to integrate it with $\beta\eta$-reduction.

Contribution. We propose a **syntax for higher-order proof terms**, called **rewrites**, that includes $\beta\eta$-equivalence and allows rewrites to be freely composed. We then define a relation $\rho \approx \sigma$ of **permutation equivalence** between rewrites, the central notion of our work. The issue mentioned above is avoided by *disallowing* the ill-behaved substitution of a rewrite in a rewrite "$\rho\{x\backslash\sigma\}$", and by only allowing notions of substitution of a term in a rewrite $\rho\{x\backslash s\}$, and of a rewrite in a term $s\{x\backslash\!\!\backslash\rho\}$. From these, a well-behaved notion of substitution of a rewrite in a rewrite $\rho\{x\backslash\!\!\backslash\sigma\}$ can be shown to be *derivable*. We also define a notion of **projection** $\rho /\!\!/ \sigma$. The induced notion of **projection equivalence coincides with permutation equivalence**, in the sense that $\rho \approx \sigma$ iff $\rho /\!\!/ \sigma \approx \sigma^{\mathsf{tgt}}$ and $\sigma /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$, where ρ^{tgt} stands for the *target* term of ρ. The equivalence is established by means of **flattening**, a method to convert an arbitrary rewrite ρ into a (*flat*) representative $\rho^\flat$ that only uses the composition operator "$;$" at the top level and a notion of **flat permutation equivalence** $\rho \sim \sigma$. Flattening is achieved by means of a rewriting system whose objects are themselves rewrites. This system is shown to be confluent and strongly normalizing. We also show that **permutation equivalence is sound and complete with respect to flat permutation equivalence** in the sense that $\rho \approx \sigma$ if and only if $\rho^\flat \sim \sigma^\flat$.

Structure of the Paper. In Section 2 we review Nipkow's Higher-Order Rewriting Systems. Section 3 proposes our notion of rewrite and Section 4 introduces permutation equivalence for them. Flattening is presented in Section 5. In this section, we also formulate an equational theory defining the relation $\rho \sim \sigma$ of flat permutation equivalence between flat rewrites. It relies crucially on a ternary relation between *multisteps*, called *splitting* and written $\mu \Leftrightarrow \mu_1\ ;\ \mu_2$, meaning that μ and $\mu_1\ ;\ \mu_2$ perform the same computational work. In Section 6 we first define a projection operator for flat rewrites ρ/σ, and we lift it to a projection operator for arbitrary rewrites $\rho /\!\!/ \sigma \overset{\text{def}}{=} \rho^\flat/\sigma^\flat$. Then we show that the induced notion of projection equivalence coincides with permutation equivalence. Finally, we conclude and discuss related and future work. Detailed proofs can be found in the accompanying technical report [2].

2 Higher-Order Rewriting

There are various approaches to HOR in the literature, including Klop's Combinatory Reduction Systems (CRSs) [8] and Nipkow's Higher-Order Rewriting Systems (HRSs) [14, 12]. We consider HRSs in this paper. Their use of the simply-typed lambda calculus for representing terms and substitution provides a suitable starting point for modeling our rewrites. Moreover, HRS are arguably more general than CRS in that their instantiation mechanism is more powerful [15, Sec.11.4.2]. We next introduce HRS. Assume given a denumerably infinite set of *variables* $(x, y, \ldots)$, *base types* $(\alpha, \beta, \ldots)$, and *constant symbols* $(\mathbf{c}, \mathbf{d}, \ldots)$. The sets of *terms* $(s, t, \ldots)$ and *types* $(A, B, \ldots)$ are given by:

$$s \quad ::= \quad x \mid \mathbf{c} \mid \lambda x.s \mid s\,s \qquad\qquad A \quad ::= \quad \alpha \mid A \to A$$

A term can either be a variable, a constant, an abstraction or an application. A type can either be a base type or an arrow type. We write $\mathsf{fv}(s)$ for the free variables of s. We use $\overline{X_n}$, or sometimes just $\overline{X}$ if n is clear from the context, to denote a sequence $X_1, \ldots, X_n$. Following standard conventions, $s\,\overline{t_n}$ stands for the iterated application $s\,t_1 \ldots t_n$, and $\overline{A_n} \to B$ for the type $A_1 \to \ldots A_n \to B$. We write $s\{x\backslash t\}$ for the capture-avoiding substitution of all free occurrences of x in s with t and call it a *term/term substitution.* We identify terms that differ only in the names of their bound variables. A *typing context* $(\Gamma, \Gamma', \ldots)$ is a partial function from variables to types. We write $\mathsf{dom}(\Gamma)$ for the *domain* of Γ. Given a typing context Γ and $x \notin \mathsf{dom}(\Gamma)$, we write $\Gamma, x : A$ for the typing context such that $(\Gamma, x : A)(x) = A$, and $(\Gamma, x : A)(y) = \Gamma(y)$ whenever $y \neq x$. We write $\cdot$ for the empty typing context and $x \in \Gamma$ if $x \in \mathsf{dom}(\Gamma)$. A *signature* of a HRS is a set $\mathcal{C}$ of typed constants $\mathbf{c} : A$. A sample signature is $\mathcal{C} = \{\mathbf{app} : \iota \to \iota \to \iota, \mathbf{lam} : (\iota \to \iota) \to \iota\}$ for ι a base type.

▶ **Definition 1** (Type system for terms). *Terms are typed using the usual typing rules of the simply-typed λ-calculus:*

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{Var} \qquad \frac{(\mathbf{c} : A) \in \mathcal{C}}{\Gamma \vdash \mathbf{c} : A}\,\mathsf{Con} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B}\,\mathsf{Abs} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash s\,t : B}\,\mathsf{App}$$

Given any Γ and A such that $\Gamma \vdash s : A$ can be proved using these rules, we say s is a typed term *over $\mathcal{C}$. We typically drop $\mathcal{C}$ assuming it is implicit.*

We assume the usual definition of β and η-reduction between terms. Recall that β-reduction (resp. η-reduction) is confluent and terminating on typed terms. We write $s\downarrow^{\beta}$ (resp. $s\downarrow^{\eta}$) for the unique β-normal form (resp. η-normal form) of s. The β-normal form of a term s has the form $\lambda\overline{x_k}.a\,t_1 \ldots t_m$, for a either a constant or a variable. The η-expanded form of s is defined as:

$$s\uparrow^{\eta} \quad \overset{\text{def}}{=} \quad \lambda\overline{x_{n+k}}.a\,(\overline{t_m}\uparrow^{\eta})\,(x_{n+1}\uparrow^{\eta}) \ldots (x_{n+k}\uparrow^{\eta})$$

where s is assumed to have type $\overline{A_{n+k}} \to B$ and the $x_{n+1}, \ldots, x_{n+k}$ are fresh. We use $s\updownarrow^{\eta}_{\beta}$ to denote the term $s\downarrow^{\beta}\uparrow^{\eta}$ and call it the $\beta\overline{\eta}$-normal form of s.

A *substitution* θ is a function from variables to typed terms such that $\theta(x) \neq x$ only for finitely many x. The *domain* of a substitution is defined as $\mathsf{dom}(\theta) = \{x \mid \theta(x) \neq x\}$. The application of a substitution $\theta = \{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ to a term t is defined as $\theta\,t \overset{\text{def}}{=} ((\lambda\overline{x_n}.t)\,\overline{s_n})\updownarrow^{\eta}_{\beta}$.

▶ **Definition 2.** *A* pattern *is a typed term in β-normal form such that all free occurrences of a variable x_i are in a subterm of the form $x_i\, t_1 \ldots t_k$ with $t_1, \ldots, t_k$ η-equivalent to distinct bound variables. A* rewriting rule *is a pair $\langle \ell, r\rangle$ of typed terms in $\beta\overline{\eta}$-normal form of the same base type with ℓ a pattern not η-equivalent to a variable and $\mathsf{fv}(r) \subseteq \mathsf{fv}(\ell)$. An HRS is a pair consisting of a signature and a set of rewriting rules over that signature. We typically omit the signature.*

▶ **Definition 3.** *The rewrite relation $\to_{\mathcal{R}}$ for an HRS $\mathcal{R}$ is the relation over typed terms in $\beta\overline{\eta}$-normal form defined as follows:*

$$\dfrac{\langle \ell, r\rangle \in \mathcal{R}}{\theta\,\ell \to_{\mathcal{R}} \theta\, r}\,\mathsf{Root} \qquad \dfrac{s \to_{\mathcal{R}} t}{a\,\overline{r_m}\,s\,\overline{p_n} \to_{\mathcal{R}} a\,\overline{r_m}\,t\,\overline{p_n}}\,\mathsf{App} \qquad \dfrac{s \to_{\mathcal{R}} t}{\lambda x.s \to_{\mathcal{R}} \lambda x.t}\,\mathsf{Abs}$$

where a is either a constant or a variable of type $\overline{A_{m+1+n}} \to B$. We write $\overset{*}{\to}_{\mathcal{R}}$ (resp. $\overset{*}{\leftrightarrow}_{\mathcal{R}}$) for the reflexive, transitive (resp. reflexive, symmetric and transtive) closure of $\to_{\mathcal{R}}$.

▶ **Example 4.** Consider a base type ι and typed constants $\mathbf{mu} : (\iota \to \iota) \to \iota$ and $\mathbf{f} : \iota \to \iota$. Two sample rewriting rules are: $\langle \mathbf{mu}(\lambda y.x\, y), x\,(\mathbf{mu}(\lambda y.x\, y))\rangle$ and $\langle \mathbf{f}\, x, \mathbf{g}\, x\rangle$. All four terms have base type ι. An example of a sequence of rewrite steps is $\mathbf{mu}\,(\lambda x.\mathbf{f}\, x) \to_{\mathcal{R}} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{f}\, x)) \to_{\mathcal{R}} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\, x)) \to_{\mathcal{R}} \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\, x))$.

An HRS is *orthogonal* if: **1.** The rules are *left-linear*, *i.e.* if the left-hand side ℓ has $\mathsf{fv}(\ell) = \{x_1, \ldots, x_n\}$, then there is *exactly* one free occurrence of x_i in ℓ, for each $1 \leq i \leq n$. **2.** There are *no critical pairs*, as defined for example in [14, Def. 4.1]. Orthogonal HRSs are deterministic in the sense that their rewrite relation is confluent. All of the examples of HRSs presented above are orthogonal. In the sequel of this paper, we assume given a fixed, orthogonal HRS $\mathcal{R}$.

3 Rewrites

In this section we propose a syntax for higher-order proof terms, called **rewrites**[1]. Rewrites for an HRS $\mathcal{R}$ are a means for denoting proofs in Higher-Order Rewriting Logic (HORL, *cf.* Def. 7) which, in turn, correspond to reduction sequences in $\mathcal{R}$ (*cf.* Thm. 9). As in the first-order case [13], HORL is simply the equational theory that results from an HRS but disregarding symmetry. Given an HRS $\mathcal{R}$, let $\mathcal{R}^c$ denote the set of pairs $\langle \lambda\overline{x_n}.\ell, \lambda\overline{x_n}.r\rangle$ such that $\langle \ell, r\rangle \in \mathcal{R}$ and $\{x_1, \ldots, x_n\} = \mathsf{fv}(\ell)$. We begin by recalling the definition of equational logic (*cf.* Def. 5), the equational theory induced by an HRS. It is essentially that of [12, Def. 3.11], except that in the inference rule ERule we use $\mathcal{R}^c$ rather than $\mathcal{R}$. This equivalent formulation will be convenient when introducing rewrites since free variables in the LHS of a rewrite rule will be reflected in the rewrite too.

[1] Our notion of rewrite is unrelated to that of Def. 2.4 in [13]; it corresponds to "proof terms" as introduced in Sec. 3.1 in [13].

▶ **Definition 5** (Equational Logic). *An HRS $\mathcal{R}$ induces a relation $\doteq_{\mathcal{R}}$ on terms defined by the following rules:*

$$\frac{\Gamma, x : A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash (\lambda x.s)\, t \doteq_{\mathcal{R}} s\{x\backslash t\} : B}\,\textsf{EBeta} \qquad \frac{\Gamma, x : A \vdash s : B \qquad x \notin \mathsf{fv}(s)}{\Gamma \vdash \lambda x.s\, x \doteq_{\mathcal{R}} s : B}\,\textsf{EEta}$$

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \doteq_{\mathcal{R}} x : A}\,\textsf{EVar} \qquad \frac{(\mathbf{c} : A) \in \mathcal{C}}{\Gamma \vdash \mathbf{c} \doteq_{\mathcal{R}} \mathbf{c} : A}\,\textsf{ECon} \qquad \frac{\Gamma, x : A \vdash s_0 \doteq_{\mathcal{R}} s_1 : B}{\Gamma \vdash \lambda x.s_0 \doteq_{\mathcal{R}} \lambda x.s_1 : A \to B}\,\textsf{EAbs}$$

$$\frac{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_1 : A \to B \qquad \Gamma \vdash t_0 \doteq_{\mathcal{R}} t_1 : A}{\Gamma \vdash s_0\, t_0 \doteq_{\mathcal{R}} s_1\, t_1 : B}\,\textsf{EApp} \qquad \frac{\langle s, t\rangle \in \mathcal{R}^c \qquad \cdot \vdash s : A \qquad \cdot \vdash t : A}{\Gamma \vdash s \doteq_{\mathcal{R}} t : A}\,\textsf{ERule}$$

$$\frac{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_1 : A}{\Gamma \vdash s_1 \doteq_{\mathcal{R}} s_0 : A}\,\textsf{ESymm} \qquad \frac{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_1 : A \qquad \Gamma \vdash s_1 \doteq_{\mathcal{R}} s_2 : A}{\Gamma \vdash s_0 \doteq_{\mathcal{R}} s_2 : A}\,\textsf{ETrans}$$

▶ **Theorem 6** (Thm. 3.12 in [12]). $\Gamma \vdash s \doteq_{\mathcal{R}} t : A$ *iff* $s \updownarrow_{\beta}^{\eta} \leftrightarrow^{*}_{\mathcal{R}} t \updownarrow_{\beta}^{\eta}$.

The ($\Leftarrow$) direction follows from observing that $\to_{\beta,\overline{\eta}}$ and $\leftrightarrow^{*}_{\mathcal{R}}$ are all included in $\doteq_{\mathcal{R}}$. The ($\Rightarrow$) direction is by induction on the derivation of $\Gamma \vdash s \doteq_{\mathcal{R}} t : A$.

Higher-Order Rewriting Logic results from dropping ESymm in Def. 5 and adding a proof witness. Its judgments take the form $\Gamma \vdash \rho : s \mathrel{-\!\!\triangleright} t : A$ where the proof witness ρ is called a *rewrite*. Given a set of *rule symbols* $(\varrho, \vartheta, \ldots)$, the set of *rewrites* $(\rho, \sigma, \ldots)$ is given by:

$$\rho ::= x \mid \mathbf{c} \mid \varrho \mid \lambda x.\rho \mid \rho\, \rho \mid \rho \,\mathbf{;}\, \rho$$

A rewrite can either be a variable, a constant, a rule symbol, an abstraction congruence, an application congruence, or a composition. Note that composition may occur anywhere inside a rewrite. For the sake of clarity we present the full system for Higher-Order Rewriting Logic next. We assume given an HRS $\mathcal{R}$ such that each rewrite rule $\langle \ell, r\rangle \in \mathcal{R}$ has been assigned a unique rule symbol ϱ and shall write $\langle \varrho, \ell, r\rangle \in \mathcal{R}$ and also use the same notation for $\mathcal{R}^c$. HORL consists of two forms of typing judgments:

1. $\Gamma \vdash s =_{\beta\eta} t : A$, meaning that s and t are $\beta\eta$-equivalent terms of type A under Γ; and
2. $\Gamma \vdash \rho : s \mathrel{-\!\!\triangleright}_{\mathcal{R}} t : A$, meaning that ρ is a rewrite with source s and target t, which are terms of type A under Γ.

▶ **Definition 7** (Higher-Order Rewriting Logic). *Term equivalence is defined as the reflexive, symmetric, transitive, and contextual closure of:*

$$\frac{\Gamma, x : A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash (\lambda x.s)\, t =_{\beta\eta} s\{x\backslash t\} : B}\,\textsf{EqBeta} \qquad \frac{\Gamma, x : A \vdash s : B \qquad x \notin \mathsf{fv}(s)}{\Gamma \vdash \lambda x.s\, x =_{\beta\eta} s : B}\,\textsf{EqEta}$$

Typing rules for rewrites are as follows:

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : x \rightarrowtriangle_{\mathcal{R}} x : A}\,\textsf{RVar} \qquad \frac{(\mathbf{c} : A) \in \mathcal{C}}{\Gamma \vdash \mathbf{c} : \mathbf{c} \rightarrowtriangle_{\mathcal{R}} \mathbf{c} : A}\,\textsf{RCon} \qquad \frac{\Gamma, x : A \vdash \rho : s_0 \rightarrowtriangle_{\mathcal{R}} s_1 : B}{\Gamma \vdash \lambda x.\rho : \lambda x.s_0 \rightarrowtriangle_{\mathcal{R}} \lambda x.s_1 : A \to B}\,\textsf{RAbs}$$

$$\frac{\Gamma \vdash \rho : s_0 \rightarrowtriangle_{\mathcal{R}} s_1 : A \to B \qquad \Gamma \vdash \sigma : t_0 \rightarrowtriangle_{\mathcal{R}} t_1 : A}{\Gamma \vdash \rho\,\sigma : s_0\,t_0 \rightarrowtriangle_{\mathcal{R}} s_1\,t_1 : B}\,\textsf{RApp}$$

$$\frac{\langle \varrho, s, t\rangle \in \mathcal{R}^c \qquad \cdot \vdash s : A \qquad \cdot \vdash t : A}{\Gamma \vdash \varrho : s \rightarrowtriangle_{\mathcal{R}} t : A}\,\textsf{RRule} \qquad \frac{\Gamma \vdash \rho : s_0 \rightarrowtriangle_{\mathcal{R}} s_1 : A \qquad \Gamma \vdash \sigma : s_1 \rightarrowtriangle_{\mathcal{R}} s_2 : A}{\Gamma \vdash \rho \mathbin{;} \sigma : s_0 \rightarrowtriangle_{\mathcal{R}} s_2 : A}\,\textsf{RTrans}$$

$$\frac{\Gamma \vdash \rho : s' \rightarrowtriangle_{\mathcal{R}} t' : A \qquad \Gamma \vdash s =_{\beta\eta} s' : A \qquad \Gamma \vdash t' =_{\beta\eta} t : A}{\Gamma \vdash \rho : s \rightarrowtriangle_{\mathcal{R}} t : A}\,\textsf{RConv}$$

The RVar and RCon rules express that variables and constants represent identity rewrites. The RAbs and RApp rules express congruence below abstraction and application. The RRule rule allows us to use a rule symbol to stand for a rewrite between its source and its target, which must be closed terms of the same type. The RConv rule states that the source and the target of a rewrite are regarded up to $\beta\eta$-equivalence. Note that there are no rules equating rewrites; such rules are the purpose of Section 4 which introduces permutation equivalence.

▶ **Example 8.** Suppose we assign the following rule symbols to the rewriting rules of Ex. 4: $\langle \varrho, \mathbf{mu}(\lambda y.x\,y), x\,(\mathbf{mu}(\lambda y.x\,y))\rangle$ and $\langle \vartheta, \mathbf{f}\,x, \mathbf{g}\,x\rangle$. Recall that $\mathcal{C} \overset{\text{def}}{=} \{\mathbf{mu} : (\iota \to \iota) \to \iota, \mathbf{f} : \iota \to \iota\}$. The reduction of Ex. 4 can be represented as a rewrite:

$$\cdot \vdash \varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) : \mathbf{mu}\,(\lambda x.\mathbf{f}\,x) \rightarrowtriangle_{\mathcal{R}} \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) : \iota$$

Inspection of the proof of Thm. 6 in [12] reveals that β and η are only needed for substitutions in rewrite rules. As a consequence:

▶ **Theorem 9.** *There is a rewrite ρ such that $\Gamma \vdash \rho : s \rightarrowtriangle_{\mathcal{R}} t : A$ if and only if $s \updownarrow^{\eta}_{\beta} \xrightarrow{*}_{\mathcal{R}} t \updownarrow^{\eta}_{\beta}$.*

Now that we know that rewrites over an HRS $\mathcal{R}$ are sound and complete with respect to reduction sequences in $\mathcal{R}$, we review some basic properties of rewrites and then focus, in the remaining sections, on equivalences between rewrites. In the sequel we will omit $\mathcal{R}$ in $\Gamma \vdash \rho : s \rightarrowtriangle_{\mathcal{R}} t : A$ and write $\Gamma \vdash \rho : s \rightarrowtriangle t : A$.

▶ **Definition 10** (Source and target of a rewrite)**.** *For each rewrite ρ we define the* source ρ^{src} *and the* target ρ^{tgt} *as the following terms:*

$$\begin{array}{rcl@{\qquad}rcl}
x^{\mathsf{src}} & \overset{\text{def}}{=} & x & x^{\mathsf{tgt}} & \overset{\text{def}}{=} & x \\
\mathbf{c}^{\mathsf{src}} & \overset{\text{def}}{=} & \mathbf{c} & \mathbf{c}^{\mathsf{tgt}} & \overset{\text{def}}{=} & \mathbf{c} \\
\varrho^{\mathsf{src}} & \overset{\text{def}}{=} & s \quad \textit{if } (\varrho : s \rightarrowtriangle t : A) \in \mathcal{R} & \varrho^{\mathsf{tgt}} & \overset{\text{def}}{=} & t \quad \textit{if } (\varrho : s \rightarrowtriangle t : A) \in \mathcal{R} \\
(\lambda x.\rho)^{\mathsf{src}} & \overset{\text{def}}{=} & \lambda x.\rho^{\mathsf{src}} & (\lambda x.\rho)^{\mathsf{tgt}} & \overset{\text{def}}{=} & \lambda x.\rho^{\mathsf{tgt}} \\
(\rho\,\sigma)^{\mathsf{src}} & \overset{\text{def}}{=} & \rho^{\mathsf{src}}\,\sigma^{\mathsf{src}} & (\rho\,\sigma)^{\mathsf{tgt}} & \overset{\text{def}}{=} & \rho^{\mathsf{tgt}}\,\sigma^{\mathsf{tgt}} \\
(\rho \mathbin{;} \sigma)^{\mathsf{src}} & \overset{\text{def}}{=} & \rho^{\mathsf{src}} & (\rho \mathbin{;} \sigma)^{\mathsf{tgt}} & \overset{\text{def}}{=} & \rho^{\mathsf{tgt}}
\end{array}$$

The free variables of an expression X (which may be a term or a rewrite) are written $\mathsf{fv}(X)$, and defined as expected, with lambdas binding variables in their bodies. For any given term or rewrite X, we write $X\{x\backslash t\}$ for the capture-avoiding substitution of the variable x in X by t. The operation $\rho\{x\backslash t\}$ is called *rewrite/term substitution.*

We mention a few important syntactic properties of terms and rewrites (detailed statements and proofs can be found in Section A of [2]). First, some basic properties hold, such as weakening (*e.g.* if $\Gamma \vdash \rho : s \rightarrow t : A$ then $\Gamma, x : B \vdash \rho : s \rightarrow t : A$) and commuting substitution with the source and target operators (*e.g.* $\rho\{x\backslash s\}^{\mathsf{src}} = \rho^{\mathsf{src}}\{x\backslash s\}$). Terms appearing in valid equality and rewriting judgments can always be shown to be typable, that is, if either $\Gamma \vdash s =_{\beta\eta} t : A$ or $\Gamma \vdash \rho : s \rightarrow t : A$, then $\Gamma \vdash s : A$ and $\Gamma \vdash t : A$. Second, given a typable rewrite, $\Gamma \vdash \rho : s \rightarrow t : A$, the source of ρ and s are not necessarily equal, but they are interconvertible, that is $\Gamma \vdash s =_{\beta\eta} \rho^{\mathsf{src}} : A$, and similarly for the target, *i.e.* $\Gamma \vdash t =_{\beta\eta} \rho^{\mathsf{tgt}} : A$. For example, if $\varrho : \lambda x.\mathbf{c}\,x \rightarrow \lambda x.\mathbf{d} : A \to A$ then it can be shown that $\vdash \varrho\,\mathbf{d} : \mathbf{c}\,\mathbf{d} \rightarrow \mathbf{d} : A$, and indeed $\mathbf{c}\,\mathbf{d} =_{\beta\eta} (\lambda x.\mathbf{c}\,x)\,\mathbf{d} = (\varrho\,\mathbf{d})^{\mathsf{src}}$. Third, any typable term s can be understood as an empty or *unit* rewrite $\underline{s}$, without occurrences of rule symbols, between s and itself: if $\Gamma \vdash s : A$ then $\Gamma \vdash \underline{s} : s \rightarrow s : A$. We usually coerce terms to rewrites implicitly if there is little danger of confusion. Substitution of a variable for a term is functorial, that is, given a rewrite $\Gamma, x : A \vdash \rho : s \rightarrow t : B$ and a term $\Gamma \vdash r : A$, then $\Gamma \vdash \rho\{x\backslash r\} : s\{x\backslash r\} \rightarrow t\{x\backslash r\} : B$.

Term/rewrite substitution generalizes term/term substitution $s\{x\backslash t\}$ when t is a rewrite, *i.e.* $s\{x\backslash\!\backslash\rho\}$. Sometimes we also call this notion *lifting substitution*, as $s\{x\backslash\!\backslash\rho\}$ "lifts" the expression s from the level of terms to the level of rewrites.

▶ **Definition 11** (Term/rewrite substitution).

$$\begin{array}{rclcrcl}
y\{x\backslash\!\backslash\rho\} & \overset{\text{def}}{=} & \begin{cases}\rho & \textit{if } x = y\\ y & \textit{if } x \neq y\end{cases} & & \mathbf{c}\{x\backslash\!\backslash\rho\} & \overset{\text{def}}{=} & \mathbf{c}\\
(\lambda y.s)\{x\backslash\!\backslash\rho\} & \overset{\text{def}}{=} & \lambda y.s\{x\backslash\!\backslash\rho\} & \textit{if } x \neq y & (s\,t)\{x\backslash\!\backslash\rho\} & \overset{\text{def}}{=} & s\{x\backslash\!\backslash\rho\}\,t\{x\backslash\!\backslash\rho\}
\end{array}$$

We mention some important properties of term/rewrite substitution. First, term/rewrite substitution is a kind of *horizontal composition*, in the sense that if $\Gamma, x : A \vdash s : B$ and $\Gamma \vdash \rho : t \rightarrow t' : A$ then $\Gamma \vdash s\{x\backslash\!\backslash\rho\} : s\{x\backslash t\} \rightarrow s\{x\backslash t'\} : B$. Second, term/rewrite and rewrite/term substitution commute according to the equation $s\{x\backslash\!\backslash\rho\}\{y\backslash t\} = s\{y\backslash t\}\{x\backslash\!\backslash\rho\{y\backslash t\}\}$, assuming that $\Gamma, x : A, y : B \vdash s : C$ and $\Gamma, y : B \vdash \rho : r \rightarrow r' : A$ and $\Gamma \vdash t : B$ (where, by convention, $x \notin \mathsf{fv}(t)$). Note that, in particular, if y does not occur free in ρ, this means that $s\{x\backslash\!\backslash\rho\}\{y\backslash t\} = s\{y\backslash t\}\{x\backslash\!\backslash\rho\}$. Third, term/rewrite substitution commutes with reflexivity in the sense that $s\{x\backslash\!\backslash\underline{t}\} = \underline{s\{x\backslash t\}}$ holds whenever $\Gamma, x : A \vdash s : B$ and $\Gamma \vdash t : A$. It also commutes with the source and target operators, in the sense that $s\{x\backslash\!\backslash\rho\}^{\mathsf{src}} = s\{x\backslash\rho^{\mathsf{src}}\}$ and $s\{x\backslash\!\backslash\rho\}^{\mathsf{tgt}} = s\{x\backslash\rho^{\mathsf{tgt}}\}$ hold whenever $\Gamma, x : A \vdash s : B$ and $\Gamma \vdash \rho : t \rightarrow t' : A$.

4 Permutation equivalence

This section presents *permutation equivalence* (Def. 12), a relation over (typed) rewrites $\rho \approx \sigma$ that identifies any two rewrites ρ and σ denoting computations in a given HRS $\mathcal{R}$ that are equivalent up to permutation of steps.

Towards Permutation Equivalence for Rewrites. Equipped with the self-evident operations of *term/rewrite substitution* $s\{x\backslash\!\backslash\rho\}$, *rewrite/term substitution* $\rho\{x\backslash t\}$ and the fact that rewrites may be freely composed, we set out to synthesize a definition of permutation equivalence by attempting to assign a meaning for $(\lambda x.\rho)\,\sigma$, where $\Gamma \vdash \rho : s_0 \rightarrow s_1 : A$ and $\Gamma \vdash \sigma : t_0 \rightarrow t_1 : A$. We begin by assuming we have equations that allow rewrites to be post-composed with their targets ($\approx$-IdR) and pre-composed with their source ($\approx$-IdL) and reason as follows:

$$(\lambda x.\rho)\,\sigma \quad \approx^{(\mathsf{IdR})} \quad ((\lambda x.\rho)\,;(\lambda x.s_1))\,\sigma \quad \approx^{(\mathsf{IdL})} \quad ((\lambda x.\rho)\,;(\lambda x.s_1))\,(t_0\,;\sigma)$$

These rewrites are syntactically valid since we allow composition inside an application. Next, we allow application to commute with composition by introducing a rule $\approx$-App: $(\rho_1\rho_2)\,;\,(\sigma_1\sigma_2) \approx (\rho_1\,;\,\sigma_1)(\rho_2\,;\,\sigma_2)$. Applying this equation leads us to:

$$((\lambda x.\rho)\,;\,(\lambda x.s_1))\,(t_0\,;\,\sigma) \quad \approx^{(\mathsf{App})} \quad (\lambda x.\rho)\,t_0\,;\,(\lambda x.s_1)\,\sigma$$

Finally, we introduce β-equality on rewrites. Arbitrary β-reduction of rewrites is not allowed *a priori*. It is only allowed when either the abstraction or the argument are unit rewrites, for which the substitution operators mentioned above can be used. These equations take the form $(\lambda x.\underline{s})\,\rho \approx s\{x\backslash\!\backslash\rho\}$ and $(\lambda x.\rho)\,\underline{s} \approx \rho\{x\backslash s\}$ and are called, $\approx$-BetaTR and $\approx$-BetaRT.

$$(\lambda x.\rho)\,t_0\,;\,(\lambda x.s_1)\,\sigma \quad \approx^{(\mathsf{BetaRT})} \quad \rho\{x\backslash t_0\}\,;\,(\lambda x.s_1)\,\sigma \quad \approx^{(\mathsf{BetaTR})} \quad \rho\{x\backslash t_0\}\,;\,s_1\{x\backslash\!\backslash\sigma\}$$

In summary we have $(\lambda x.\rho)\,\sigma \approx \rho\{x\backslash t_0\}\,;\,s_1\{x\backslash\!\backslash\sigma\}$. We could equally well have deduced $(\lambda x.\rho)\,\sigma \approx s_0\{x\backslash\!\backslash\sigma\}\,;\,\rho\{x\backslash t_1\}$. As it turns out, however, $\rho\{x\backslash t_0\}\,;\,s_1\{x\backslash\!\backslash\sigma\}$ and $s_0\{x\backslash\!\backslash\sigma\}\,;\,\rho\{x\backslash t_1\}$ are permutation equivalent in our theory.

Permutation Equivalence for Rewrites: Definition and Properties. We collect the observations above in the following definition.

▶ **Definition 12** (Permutation equivalence). *Suppose $\Gamma \vdash \rho : s \rightarrow t : A$ and $\Gamma \vdash \rho' : s' \rightarrow t' : A$ are derivable. Permutation equivalence, written $\Gamma \vdash (\rho : s \rightarrow t) \approx (\rho' : s' \rightarrow t') : A$ (or simply $\rho \approx \rho'$ if Γ, s, t, s', t', A are clear from the context), is defined as the reflexive, symmetric, transitive, and contextual closure of the following axioms:*

$$\begin{array}{rcll}
\underline{\rho^{\mathsf{src}}}\,;\,\rho & \approx & \rho & & \approx\text{-IdL} \\
\rho\,;\,\underline{\rho^{\mathsf{tgt}}} & \approx & \rho & & \approx\text{-IdR} \\
(\rho\,;\,\sigma)\,;\,\tau & \approx & \rho\,;\,(\sigma\,;\,\tau) & & \approx\text{-Assoc} \\
(\lambda x.\rho)\,;\,(\lambda x.\sigma) & \approx & \lambda x.(\rho\,;\,\sigma) & & \approx\text{-Abs} \\
(\rho_1\rho_2)\,;\,(\sigma_1\sigma_2) & \approx & (\rho_1\,;\,\sigma_1)(\rho_2\,;\,\sigma_2) & & \approx\text{-App} \\
(\lambda x.\underline{s})\,\rho & \approx & s\{x\backslash\!\backslash\rho\} & & \approx\text{-BetaTR} \\
(\lambda x.\rho)\,\underline{s} & \approx & \rho\{x\backslash s\} & & \approx\text{-BetaRT} \\
\lambda x.\rho\,x & \approx & \rho & \textit{if } x \notin \mathsf{fv}(\rho) & \approx\text{-Eta}
\end{array}$$

Rules $\approx$-IdL, $\approx$-IdR and $\approx$-Assoc, state that rewrites together with rewrite composition have a monoidal structure. Recall from Section 3 that ρ^{src} is a term and $\underline{\rho^{\mathsf{src}}}$ is its corresponding rewrite. Rules $\approx$-Abs and $\approx$-App state that rewrite composition commutes with abstraction and application. An important thing to be wary of is that rules may be applied only if both the left and the right-hand sides are well-typed. In particular, the right-hand side of the $\approx$-App rule may not be well-typed even if the left-hand side is; for example given rule symbols $\mathbf{c} : A \to B$ and $\mathbf{d} : A$, the expression $((\lambda x.x)(\mathbf{c}\,\mathbf{d}))\,;\,(\mathbf{c}\,\mathbf{d})$ is well-typed, with source and target $\mathbf{c}\,\mathbf{d}$, while $((\lambda x.x)\,;\,\mathbf{c})\,((\mathbf{c}\,\mathbf{d})\,;\,\mathbf{d})$ is not well-typed.

Finally, rules $\approx$-BetaTR, $\approx$-BetaRT and $\approx$-Eta introduce $\beta\eta$-equivalence for rewrites. Note that $\approx$-BetaTR and $\approx$-BetaRT restrict either the body of the abstraction or the argument to a unit rewrite, thus avoiding the issue mentioned in the introduction where a naive combination of composition and $\beta\eta$-equivalence can lead to invalid rewrites.

Note that there are no explicit sequencing equations such as the I/O equations[2] defining permutation equivalence in the first-order case [15] and the corresponding equations flat-l and flat-r of [4] for the higher-order case. Nonetheless, we can derive the following coherence equation (see Lem. 63 and Lem. 64 in [2] for the proof):

$$\rho\{x\backslash s'\} \mathbin{;} t\{x\backslash\!\!\backslash\sigma\} \quad\approx\quad s\{x\backslash\!\!\backslash\sigma\} \mathbin{;} \rho\{x\backslash t'\} \qquad (\approx\text{-Perm})$$

where $\Gamma, x : A \vdash \rho : s \twoheadrightarrow t : B$ and $\Gamma \vdash \sigma : s' \twoheadrightarrow t' : A$.

▶ **Example 13.** Consider the HRS of Ex. 4 and the reduction of Ex. 8. We recall the latter below (R_2) and present a second one (R_1).

$$\begin{array}{lcl} R_1 & : & \mathbf{mu}\,(\lambda x.\mathbf{f}\,x) \twoheadrightarrow \mathbf{mu}\,(\lambda x.\mathbf{g}\,x) \twoheadrightarrow \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \\ R_2 & : & \mathbf{mu}\,(\lambda x.\mathbf{f}\,x) \twoheadrightarrow \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{f}\,x)) \twoheadrightarrow \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \twoheadrightarrow \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \end{array}$$

Reduction sequence R_1 can be encoded as the rewrite $\mathbf{mu}\,(\lambda x.\vartheta\,x); \varrho\,(\lambda x.\mathbf{g}\,x)$ and R_2 as $\varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x))$. These two rewrites are permutation equivalent:

$$\begin{array}{ll} & \mathbf{mu}\,(\lambda x.\vartheta\,x) \mathbin{;} \varrho\,(\lambda x.\mathbf{g}\,x) \\ \approx^{(\mathsf{Eta})} & \mathbf{mu}\,\vartheta \mathbin{;} \varrho\,\mathbf{g} \\ = & (\mathbf{mu}\,y)\{y\backslash\!\!\backslash\vartheta\} \mathbin{;} (\varrho\,y)\{y\backslash\mathbf{g}\} \\ \approx^{(\mathsf{Perm})} & (\varrho\,y)\{y\backslash\mathbf{f}\} \mathbin{;} (y\,(\mathbf{mu}\,y))\{y\backslash\!\!\backslash\vartheta\} \\ = & \varrho\,\mathbf{f} \mathbin{;} \vartheta\,(\mathbf{mu}\,\vartheta) \\ \approx^{(\mathsf{IdL})} & \varrho\,\mathbf{f} \mathbin{;} (\mathbf{f} \mathbin{;} \vartheta)\,(\mathbf{mu}\,\vartheta) \\ \approx^{(\mathsf{IdR})} & \varrho\,\mathbf{f} \mathbin{;} (\mathbf{f} \mathbin{;} \vartheta)\,((\mathbf{mu}\,\vartheta) \mathbin{;} (\mathbf{mu}\,\mathbf{g})) \\ \approx^{(\mathsf{App})} & \varrho\,\mathbf{f} \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,\vartheta) \mathbin{;} \vartheta\,(\mathbf{mu}\,\mathbf{g}) \\ \approx^{(\mathsf{Eta})} & \varrho\,(\lambda x.\mathbf{f}\,x) \mathbin{;} \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \end{array}$$

The $\approx$-Perm rule motivates the definition of *rewrite/rewrite substitution*, $\rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\} \stackrel{\text{def}}{=} \rho\{x\backslash s'\} \mathbin{;} t\{x\backslash\!\!\backslash\sigma\}$, which defines a rewrite $s\{x\backslash s'\} \twoheadrightarrow t\{x\backslash t'\}$. Note that $\rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\}$ depends on t and s', and hence on the particular typing derivations for ρ and σ. Congruence results (Lem. 63 and Lem. 64 in [2]) ensure that the value of $\rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\}$ does not depend, up to permutation equivalence, on those typing derivations. Rewrite/rewrite substitution generalizes rewrite/term and term/rewrite substitution, in the sense that $\rho\{x\backslash t\} \approx \rho\{x\backslash\!\!\backslash\!\!\backslash\underline{t}\}$ and $s\{x\backslash\!\!\backslash\rho\} \approx \underline{s}\{x\backslash\!\!\backslash\!\!\backslash\rho\}$.

Other important facts involving rewrite/rewrite substitution are the following. First, it commutes with abstraction, application, and composition, that is $(\lambda y.\rho)\{x\backslash\!\!\backslash\!\!\backslash\sigma\} \approx \lambda y.\rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\}$, $(\rho_1\,\rho_2)\{x\backslash\!\!\backslash\!\!\backslash\sigma\} \approx \rho_1\{x\backslash\!\!\backslash\!\!\backslash\sigma\}\,\rho_2\{x\backslash\!\!\backslash\!\!\backslash\sigma\}$, and $(\rho_1 \mathbin{;} \rho_2)\{x\backslash\!\!\backslash\!\!\backslash\sigma_1 \mathbin{;} \sigma_2\} \approx \rho_1\{x\backslash\!\!\backslash\!\!\backslash\sigma_1\} \mathbin{;} \rho_2\{x\backslash\!\!\backslash\!\!\backslash\sigma_2\}$. Second, permutation equivalence is a congruence with respect to rewrite/rewrite substitution, that is, if $\rho \approx \rho'$ and $\sigma \approx \sigma'$ then $\rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\} \approx \rho'\{x\backslash\!\!\backslash\!\!\backslash\sigma'\}$. Third, an analog of the substitution lemma holds, namely $\rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\}\{y\backslash\!\!\backslash\!\!\backslash\tau\} \approx \rho\{y\backslash\!\!\backslash\!\!\backslash\tau\}\{x\backslash\!\!\backslash\!\!\backslash\sigma\{y\backslash\!\!\backslash\!\!\backslash\tau\}\}$. Finally, as discussed above, a β-rule for arbitrary rewrites holds in the form $(\lambda x.\rho)\,\sigma \approx \rho\{x\backslash\!\!\backslash\!\!\backslash\sigma\}$. The full theory of rewrite/rewrite substitution is not developed here for lack of space (but see Section B.2 in [2]).

5 Flattening

Allowing composition to be nested within application and abstraction can give rise to rewrites in which it is not obvious what reduction sequences of steps are being denoted. An example from the previous section might be the rewrite $((\lambda x.\mathbf{f}\,x) \mathbin{;} \vartheta)\,((\mathbf{mu}\,(\lambda x.\vartheta\,x)) \mathbin{;} (\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)))$

[2] $I : \varrho(\sigma_1, ..., \sigma_n) \approx l(\sigma_1, ..., \sigma_n) \cdot \varrho(t_1, ..., t_n)$ and $O : \varrho(\sigma_1, ..., \sigma_n) \approx \varrho(s_1, ..., s_n) \cdot r(\sigma_1, ..., \sigma_n)$

which denotes the reduction sequence $\mathbf{f}\,(\mathbf{mu}\,(\lambda x.\mathbf{f}\,x)) \rightarrowtriangle \mathbf{g}\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x))$ that replaces both occurrences of $\mathbf{f}$ with $\mathbf{g}$ simultaneously. This section shows how rewrites can be "flattened" so as to expose an underlying reduction sequence, expressed as a canonical (*flat*) rewrite. One additional use of flattening will be to use it to show that permutation equivalence is decidable (*cf.* end of Sec. Section 6). Before introducing flat rewrites we define *multisteps.*

A *multistep* is a rewrite without any occurrences of the composition operator. We use $\mu, \nu, \xi, \ldots$ to range over multisteps. The capture-avoiding substitution of the free occurrences of x in μ by ν is written $\mu\{x\backslash\nu\}$, which is in turn a multistep. A *flat multistep* $(\hat{\mu}, \hat{\nu}, \ldots)$, is a multistep in β-normal form, *i.e.* without subterms of the form $(\lambda x.\mu)\,\nu$. A *flat rewrite* $(\hat{\rho}, \hat{\sigma}, \ldots)$, is a rewrite given by the grammar $\hat{\rho} ::= \hat{\mu} \mid \hat{\rho}\,;\,\hat{\sigma}$. Flat rewrites use the composition operator "$;$" at the top level, that is they are of the form $\hat{\mu}_1\,;\,\ldots\,;\,\hat{\mu}_n$ (up to associativity of "$;$"), where each $\hat{\mu}_i$ is a flat multistep. Note that we do not require the $\hat{\mu}_i$ to be in $\beta\eta$-normal form nor in $\beta\overline{\eta}$-normal form. As mentioned in the introduction, flattening is achieved by means of a *rewriting system whose objects are themselves rewrites* (Def. 15) which is shown to be *confluent and terminating* (Prop. 17).

We also formulate an equational theory defining a relation $\rho \sim \sigma$ of *flat permutation equivalence* between flat rewrites (Def. 19). The main result of this section is that permutation equivalence is *sound and complete* with respect to flat permutation equivalence (Thm. 20).

▶ Remark 14. A substitution $\mu\{x\backslash\nu\}$ in which μ is a term is a term/rewrite substitution, *i.e.* $s\{x\backslash\nu\} = s\{x\backslash\!\backslash\nu\}$. A substitution in which ν is a term is a rewrite/term substitution, *i.e.* $\mu\{x\backslash s\} = \mu\{x\backslash s\}$.

▶ **Definition 15** (Flattening Rewrite System $\mathcal{F}$)**.** *The flattening system $\mathcal{F}$ is given by the following rules, closed under arbitrary contexts, defined between* **typable** *rewrites:*

$\lambda x.(\rho\,;\,\sigma)$	$\overset{\flat}{\mapsto}$	$(\lambda x.\rho)\,;\,(\lambda x.\sigma)$		$\mathcal{F}$-Abs
$(\rho\,;\,\sigma)\,\mu$	$\overset{\flat}{\mapsto}$	$(\rho\,\underline{\mu^{\mathsf{src}}})\,;\,(\sigma\,\mu)$		$\mathcal{F}$-App1
$\mu\,(\rho\,;\,\sigma)$	$\overset{\flat}{\mapsto}$	$(\mu\,\rho)\,;\,(\underline{\mu^{\mathsf{tgt}}}\,\sigma)$		$\mathcal{F}$-App2
$(\rho_1\,;\,\rho_2)\,(\sigma_1\,;\,\sigma_2)$	$\overset{\flat}{\mapsto}$	$((\rho_1\,;\,\rho_2)\,\underline{\sigma_1^{\mathsf{src}}})\,;\,(\underline{\rho_2^{\mathsf{tgt}}}\,(\sigma_1\,;\,\sigma_2))$		$\mathcal{F}$-App3
$(\lambda x.\mu)\,\nu$	$\overset{\flat}{\mapsto}$	$\mu\{x\backslash\nu\}$		$\mathcal{F}$-BetaM
$\lambda x.\mu\,x$	$\overset{\flat}{\mapsto}$	μ	*if* $x \notin \mathsf{fv}(\mu)$	$\mathcal{F}$-EtaM

*Note that rules $\mathcal{F}$-*BetaM *and $\mathcal{F}$-*EtaM *apply to multisteps only. The reduction relation $\overset{\flat}{\mapsto}$ is the union of all these rules, closed by compatibility under arbitrary contexts. We write $\rho^\flat$ for the unique $\overset{\flat}{\mapsto}$-normal form of ρ.*

▶ **Example 16.** Consider a rewriting rule $\varrho : \mathbf{c} \rightarrowtriangle \mathbf{d} : A$. The rewrite $(\lambda x.(x\,;\,x))\,\varrho$, whose meaning (as previously mentioned) is not obvious, can be flattened as follows:

$$\begin{array}{lllll}
(\lambda x.(x\,;\,x))\,\varrho & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-Abs}} & ((\lambda x.x)\,;\,(\lambda x.x))\,\varrho & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-App1}} & (\lambda x.x)\,\mathbf{c}\,;\,(\lambda x.x)\,\varrho \\
& \overset{\flat}{\mapsto}_{\mathcal{F}\text{-BetaM}} & \mathbf{c}\,;\,(\lambda x.x)\,\varrho & \overset{\flat}{\mapsto}_{\mathcal{F}\text{-BetaM}} & \mathbf{c}\,;\,\varrho
\end{array}$$

The following result is proved by noting that $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM steps can be postponed after steps of other kinds and then providing a well-founded measure for steps in $\mathcal{F}$ without $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM to prove it is SN. Confluence of $\mathcal{F}$ follows from Newman's lemma.

▶ **Proposition 17.** *The flattening system $\mathcal{F}$ is strongly normalizing and confluent.*

Flat Permutation Equivalence. We now turn to the definition of the relation $\rho \sim \sigma$ of flat permutation equivalence. The key notion to define is the following ternary relation:

▶ **Definition 18** (Splitting). *Let* $\Gamma \vdash \mu : s \twoheadrightarrow t : A$ *and* $\Gamma \vdash \mu_1 : s' \twoheadrightarrow r_1 : A$ *and* $\Gamma \vdash \mu_2 : r_2 \twoheadrightarrow t' : A$ *be multisteps. We say that* μ splits *into* μ_1 *and* μ_2 *if the following inductively defined ternary relation, written* $\mu \Leftrightarrow \mu_1 \,;\, \mu_2$*, holds:*

$$\frac{}{x \Leftrightarrow x \,;\, x}\,\mathsf{SVar} \qquad \frac{}{\mathbf{c} \Leftrightarrow \mathbf{c} \,;\, \mathbf{c}}\,\mathsf{SCon} \qquad \frac{}{\varrho \Leftrightarrow \varrho \,;\, \underline{\varrho^{\mathsf{tgt}}}}\,\mathsf{SRuleL} \qquad \frac{}{\varrho \Leftrightarrow \underline{\varrho^{\mathsf{src}}} \,;\, \varrho}\,\mathsf{SRuleR}$$

$$\frac{\mu \Leftrightarrow \mu_1 \,;\, \mu_2}{\lambda x.\mu \Leftrightarrow \lambda x.\mu_1 \,;\, \lambda x.\mu_2}\,\mathsf{SAbs} \qquad \frac{\mu \Leftrightarrow \mu_1 \,;\, \mu_2 \qquad \nu \Leftrightarrow \nu_1 \,;\, \nu_2}{\mu\,\nu \Leftrightarrow \mu_1\,\nu_1 \,;\, \mu_2\,\nu_2}\,\mathsf{SApp}$$

▶ **Definition 19** (Flat permutation equivalence). *Flat permutation equivalence judgments are of the form:* $\Gamma \vdash (\rho : s \twoheadrightarrow t) \sim (\rho' : s' \twoheadrightarrow t') : A$*, meaning that* ρ *and* ρ' *are equivalent rewrites, with sources* s *and* s' *respectively, and targets* t *and* t' *respectively. The rewrites* ρ *and* ρ' *are* **assumed to be in** $\overset{\flat}{\mapsto}$**-normal form**, *which in particular means that they must be flat rewrites. Sometimes we write* $\rho \sim \rho'$ *if* Γ, s, t, s', t', A *are irrelevant or clear from the context. Derivability is defined by the two following axioms, which are closed by reflexivity, symmetry, transitivity, and closure under* composition contexts *(given by* $\mathsf{S} ::= \Box \mid \mathsf{S} \,;\, \rho \mid \rho \,;\, \mathsf{S}$*):*

$$\begin{array}{lll} (\rho \,;\, \sigma) \,;\, \tau \sim \rho \,;\, (\sigma \,;\, \tau) & & \sim\text{-}\mathsf{Assoc} \\ \qquad \mu \sim \mu_1^\flat \,;\, \mu_2^\flat & \text{if } \mu \Leftrightarrow \mu_1 \,;\, \mu_2 & \sim\text{-}\mathsf{Perm} \end{array}$$

Note that in $\sim$-Perm, $-^\flat$ operates over multisteps. So the only rules of $\mathcal{F}$ that are applied here are the $\mathcal{F}$-BetaM and $\mathcal{F}$-EtaM rules.

▶ **Theorem 20** (Soundness and completeness of flat permutation equivalence). *Let* $\Gamma \vdash \rho : s \twoheadrightarrow t : A$ *and* $\Gamma \vdash \sigma : s' \twoheadrightarrow t' : A$*. Then* $\rho \approx \sigma$ *if and only if* $\rho^\flat \sim \sigma^\flat$*.*

Proof. The ($\Leftarrow$) direction is immediate, given that reduction $\overset{\flat}{\mapsto}$ in the flattening system $\mathcal{F}$ is included in permutation equivalence ($\rho \overset{\flat}{\mapsto} \sigma$ implies $\rho \approx \sigma$) and, similarly, flat permutation equivalence is included in permutation equivalence ($\rho \sim \sigma$ implies $\rho \approx \sigma$).

The ($\Rightarrow$) direction is by induction on the derivation of $\rho \approx \sigma$. It is subtle and requires numerous auxiliary results (see Section D.8 in [2]). ◀

▶ **Example 21.** With the same notation as in Ex. 13, it can be checked that the rewrites $\mathbf{mu}\,(\lambda x.\vartheta\,x) \,;\, \varrho\,(\lambda x.\mathbf{g}\,x)$ and $\varrho\,(\lambda x.\mathbf{f}\,x) \,;\, \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \,;\, \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x))$ are permutation equivalent by means of flattening. Indeed, using the $\sim$-Perm rule three times:

$$\begin{array}{llll} \mathbf{mu}\,\vartheta \,;\, \varrho\,\mathbf{g} & \sim & \varrho\,\vartheta & \text{as } \varrho\,\vartheta \Leftrightarrow (\lambda x.\mathbf{mu}\,(\lambda y.x\,y))\,\vartheta \,;\, \varrho\,(\lambda x.\mathbf{g}\,x) \\ & \sim & \varrho\,\mathbf{f} \,;\, \vartheta\,(\mathbf{mu}\,\vartheta) & \text{as } \varrho\,\vartheta \Leftrightarrow \varrho\,(\lambda x.\mathbf{f}\,x) \,;\, (\lambda x.x\,(\mathbf{mu}\,(\lambda y.x\,y)))\,\vartheta \\ & \sim & \varrho\,\mathbf{f} \,;\, (\mathbf{f}(\mathbf{mu}\,\vartheta) \,;\, \vartheta(\mu\,\mathbf{g})) & \text{as } \vartheta\,(\mathbf{mu}\,\vartheta) \Leftrightarrow (\lambda x.\mathbf{f}\,x)\,(\mathbf{mu}\,\vartheta) \,;\, \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)) \end{array}$$

Note that $\varrho\,\vartheta \Leftrightarrow (\lambda x.\mathbf{mu}\,(\lambda y.x\,y))\,\vartheta \,;\, \varrho\,(\lambda x.\mathbf{g}\,x)$ follows from SApp, SRuleR for the upper left hypothesis and SRuleL for the upper right one. Hence

$$\begin{array}{lll} (\mathbf{mu}\,(\lambda x.\vartheta\,x) \,;\, \varrho\,(\lambda x.\mathbf{g}\,x))^\flat & = & \mathbf{mu}\,\vartheta \,;\, \varrho\,\mathbf{g} \\ & \sim & \varrho\,\mathbf{f} \,;\, (\mathbf{f}(\mathbf{mu}\,\vartheta) \,;\, \vartheta(\mu\,\mathbf{g})) \\ & = & (\varrho\,(\lambda x.\mathbf{f}\,x) \,;\, \mathbf{f}\,(\mathbf{mu}\,(\lambda x.\vartheta\,x)) \,;\, \vartheta\,(\mathbf{mu}\,(\lambda x.\mathbf{g}\,x)))^\flat \end{array}$$

6 Projection

This section presents projection equivalence. Two rewrites ρ and σ are said to be projection equivalent if the steps performed by ρ are included in those performed by σ and vice-versa. We proceed in stages as follows. First, we define *projection* of multisteps over multisteps (Def. 25) and prove some of its properties (Prop. 26). Second, we extend projection to flat rewrites (Def. 28). Third, we extend projection to arbitrary rewrites (Def. 29) and, again, we prove some of its properties (Prop. 30). Finally, we show that the induced notion of *projection equivalence* turns out to coincide with permutation equivalence (Thm. 31).

Projection for Multisteps. Consider the rewrites $\mathbf{mu}\,\vartheta$ and $\varrho\,\mathbf{f}$, using the notation of Ex. 13, each representing one step. Since rewrites are subject to $\beta\eta$-equivalence, to define projection one must "line up" rule symbols with the left-hand side of the rewrite rules they witness[3]. For example, if the above two multisteps were rewritten as $(\lambda y.\mathbf{mu}\,(\lambda x.y\,x))\,\vartheta$ and $\varrho\,(\lambda x.\mathbf{f}\,x)$, respectively, then one can reason inductively as follows to compute the projection of the former over the latter (the inference rules themselves are introduced in Def. 22):

$$\dfrac{\dfrac{}{\lambda y.\mathbf{mu}\,(\lambda x.y\,x) \mathbin{/\!\!/\!\!/} \varrho \Rightarrow \lambda y.y\,(\mathbf{mu}\,(\lambda x.y\,x))}\,\textsf{ProjRuleR} \qquad \dfrac{}{\vartheta \mathbin{/\!\!/\!\!/} \lambda x.\mathbf{f}\,x \Rightarrow \vartheta}\,\textsf{ProjRuleL}}{(\lambda y.\mathbf{mu}\,(\lambda x.y\,x))\,\vartheta \mathbin{/\!\!/\!\!/} \varrho\,(\lambda x.\mathbf{f}\,x) \Rightarrow (\lambda y.y\,(\mathbf{mu}\,(\lambda x.y\,x)))\,\vartheta}\,\textsf{ProjApp}$$

The flat normal form of $(\lambda y.y\,(\mathbf{mu}\,(\lambda x.y\,x)))\,\vartheta$ is the rewrite $\vartheta\,(\mathbf{mu}\,\vartheta)$. Hence we would deduce $\mathbf{mu}\,\vartheta \mathbin{/\!\!/\!\!/} \varrho\,\mathbf{f} \Rightarrow \vartheta\,(\mathbf{mu}\,\vartheta)$. We begin by introducing an auxiliary notion of projection on coinitial multisteps that may not be flat (*i.e.* may not be in $\mathcal{F}$-BetaM, $\mathcal{F}$-EtaM-normal form) called *weak projection*. We then make use of this notion, to define projection for flat multisteps (Def. 25).

▶ **Definition 22** (Weak projection and compatibility). *Let $\Gamma \vdash \mu : s \twoheadrightarrow t : A$ and $\Gamma \vdash \nu : s' \twoheadrightarrow r : A$ be multisteps, not necessarily in normal form, such that $s =_{\beta\eta} s'$. The judgment $\mu \mathbin{/\!\!/\!\!/} \nu \Rightarrow \xi$ is defined as follows:*

$$\dfrac{}{x \mathbin{/\!\!/\!\!/} x \Rightarrow x}\,\textsf{ProjVar} \qquad \dfrac{}{\mathbf{c} \mathbin{/\!\!/\!\!/} \mathbf{c} \Rightarrow \mathbf{c}}\,\textsf{ProjCon} \qquad \dfrac{}{\varrho \mathbin{/\!\!/\!\!/} \varrho \Rightarrow \varrho^{\textsf{tgt}}}\,\textsf{ProjRule} \qquad \dfrac{}{\varrho \mathbin{/\!\!/\!\!/} \varrho^{\textsf{src}} \Rightarrow \varrho}\,\textsf{ProjRuleL}$$

$$\dfrac{}{\varrho^{\textsf{src}} \mathbin{/\!\!/\!\!/} \varrho \Rightarrow \varrho^{\textsf{tgt}}}\,\textsf{ProjRuleR} \qquad \dfrac{\mu \mathbin{/\!\!/\!\!/} \nu \Rightarrow \xi}{\lambda x.\mu \mathbin{/\!\!/\!\!/} \lambda x.\nu \Rightarrow \lambda x.\xi}\,\textsf{ProjAbs} \qquad \dfrac{\mu_1 \mathbin{/\!\!/\!\!/} \nu_1 \Rightarrow \xi_1 \qquad \mu_2 \mathbin{/\!\!/\!\!/} \nu_2 \Rightarrow \xi_2}{\mu_1\,\mu_2 \mathbin{/\!\!/\!\!/} \nu_1\,\nu_2 \Rightarrow \xi_1\,\xi_2}\,\textsf{ProjApp}$$

We say that μ and ν are compatible*, written $\mu \uparrow \nu$ if, intuitively speaking, μ and ν are coinitial, and are "almost" η-expanded and β-normal forms, with the exception that the head of the term may be the source of a rule,* i.e. *a term of the form $\varrho^{\textsf{src}}$. Compatibility is defined as follows:*

$$\dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.y\,\overline{\mu} \uparrow \lambda\overline{x}.y\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\mathbf{c}\,\overline{\mu} \uparrow \lambda\overline{x}.\mathbf{c}\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\varrho\,\overline{\mu} \uparrow \lambda\overline{x}.\varrho\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\varrho\,\overline{\mu} \uparrow \lambda\overline{x}.\varrho^{\textsf{src}}\,\overline{\nu}} \qquad \dfrac{(\mu_i \uparrow \nu_i)_{i=1}^m}{\lambda\overline{x}.\varrho^{\textsf{src}}\,\overline{\mu} \uparrow \lambda\overline{x}.\varrho\,\overline{\nu}}$$

[3] See also the discussion on pg. 120 of [4].

The interesting cases are the two last rules, which state essentially that a rule symbol is compatible with its source term. Clearly if $\mu \uparrow \nu$, then there exists a unique ξ such that $\mu /\!\!/\!\!/ \nu \Rrightarrow \xi$. Moreover, weak projection is coherent with respect to flattening:

▶ **Lemma 23** (Coherence of projection). *Let $\mu_1, \nu_1, \mu_2, \nu_2$ be multisteps such that the following are satisfied:*
1. *$\mu_1 \uparrow \nu_1$ and $\mu_2 \uparrow \nu_2$;*
2. *$\mu_1^\flat = \mu_2^\flat$ and $\nu_1^\flat = \nu_2^\flat$; and*
3. *$\mu_1 /\!\!/\!\!/ \nu_1 \Rrightarrow \xi_1$ and $\mu_2 /\!\!/\!\!/ \nu_2 \Rrightarrow \xi_2$.*

Then $\xi_1^\flat = \xi_2^\flat$.

Thus for arbitrary, coinitial multisteps μ and ν, it suffices to show that we can always find corresponding *compatible* "almost" η-expanded and β-normal forms, as mentioned above.

▶ **Proposition 24** (Existence and uniqueness of projection). *Let μ, ν be such that $\mu^{\mathsf{src}} =_{\beta\eta} \nu^{\mathsf{src}}$. Then:*
1. Existence. *There exist multisteps $\dot{\mu}, \dot{\nu}, \dot{\xi}$ such that $\dot{\mu}^\flat = \mu^\flat$ and $\dot{\nu}^\flat = \nu^\flat$ and $\dot{\mu} /\!\!/\!\!/ \dot{\nu} \Rrightarrow \dot{\xi}$.*
2. Compatibility. *Furthermore, $\dot{\mu}$ and $\dot{\nu}$ can be chosen in such a way that $\dot{\mu} \uparrow \dot{\nu}$.*
3. Uniqueness. *If $(\dot{\mu}')^\flat = \mu^\flat$ and $(\dot{\nu}')^\flat = \nu^\flat$ and $\dot{\mu}' /\!\!/\!\!/ \dot{\nu}' \Rrightarrow \dot{\xi}'$ then $(\dot{\xi}')^\flat = \xi^\flat$.*

Prop. 24 relies on the left-hand side of the rewrite rules of the HRS being patterns. This ensures, among other things, that flattening is injective when applied to left-hand sides of rewrite rules in the sense that if $(\varrho^{\mathsf{src}}\, \mu_1 \ldots \mu_n)^\flat = (\varrho^{\mathsf{src}}\, \nu_1 \ldots \nu_n)^\flat$ then $\mu_i^\flat = \nu_i^\flat$ for all $1 \leq i \leq n$. We can now define projection on arbitrary coinitial rewrites as follows.

▶ **Definition 25** (Projection operator for multisteps). *Let μ, ν be such that $\mu^{\mathsf{src}} =_{\beta\eta} \nu^{\mathsf{src}}$. We write μ/ν for the unique multistep of the form $\dot{\xi}^\flat$ such that there exist $\dot{\mu}, \dot{\nu}$ such that $\dot{\mu}^\flat = \mu^\flat$ and $\dot{\nu}^\flat = \nu^\flat$ and $\dot{\mu} /\!\!/\!\!/ \dot{\nu} \Rrightarrow \dot{\xi}$, as guaranteed by Prop. 24. The proof is constructive (this relies on the HRS being orthogonal), thus providing an effective method to compute μ/ν.*

▶ **Proposition 26** (Properties of projection for multisteps).
1. $\mu/\nu = (\mu/\nu)^\flat = \mu^\flat/\nu^\flat$
2. *Projection commutes with abstraction and application, that is, $(\lambda x.\mu)/(\lambda x.\nu) = (\lambda x.(\mu/\nu))^\flat$ and $(\mu_1\, \mu_2)/(\nu_1\, \nu_2) = ((\mu_1/\nu_1)\, (\mu_2/\nu_2))^\flat$, provided that μ_1/ν_1 and μ_2/ν_2 are defined.*
3. *The set of multisteps with the projection operator form a residual system [15, Def. 8.7.2]:*
   - **3.1** *$(\mu/\nu)/(\xi/\nu) = (\mu/\xi)/(\nu/\xi)$, known as the* Cube Lemma.
   - **3.2** *$\mu/\mu = (\mu^{\mathsf{tgt}})^\flat$ and, as particular cases: $\underline{s}/\underline{s} = \underline{s}^\flat$, $x/x = x$, $\mathbf{c}/\mathbf{c} = \mathbf{c}$, and $\varrho/\varrho = (\varrho^{\mathsf{tgt}})^\flat$.*
   - **3.3** *$(\mu^{\mathsf{src}})^\flat/\mu = (\mu^{\mathsf{tgt}})^\flat$ and, as a particular case, $(\varrho^{\mathsf{src}})^\flat/\varrho = (\varrho^{\mathsf{tgt}})^\flat$.*
   - **3.4** *$\mu/(\mu^{\mathsf{src}})^\flat = \mu^\flat$ and, as a particular case, $\varrho/(\varrho^{\mathsf{src}})^\flat = \varrho$.*

▶ **Example 27.** Let $\vartheta : \lambda x.\mathbf{f}\, x \to \lambda x.\mathbf{g}\, x$. Then:

$$\begin{array}{rlclcl} & (\lambda x.(\lambda x.\mathbf{f}\, x)\, x)/(\lambda x.\vartheta\, x) & = & (\lambda x.((\lambda x.\mathbf{f}\, x)\, x)/(\vartheta\, x))^\flat & = & (\lambda x.(((\lambda x.\mathbf{f}\, x)/\vartheta)(x/x))^\flat)^\flat \\ = & (\lambda x.((\lambda x.\mathbf{g}\, x)\, x)^\flat)^\flat & = & (\lambda x.\mathbf{g}\, x)^\flat & = & \mathbf{g} \end{array}$$

Projection for Flat Rewrites. The projection operator from Def. 25 is extended to operate on flat rewrites. One may try to define ρ/σ using equations such as $(\rho_1 \mathbin{;} \rho_2)/\sigma = (\rho_1/\sigma) \mathbin{;} (\rho_2/(\sigma/\rho_1))$. However, it is not *a priori* clear that this recursive definition is well-founded[4]. This is why the following definition proceeds in three stages:

[4] Another way to prove well-foundedness is by interpretation, as done in [15, Example 6.5.43].

▶ **Definition 28** (Projection operator for flat rewrites). *We define:*

1. *projection of a flat multistep over a coinitial flat rewrite* $(\mu \,/^1\, \rho)$*, by induction on* ρ*;*
2. *projection of a flat rewrite over a coinitial flat multistep* $(\rho \,/^2\, \mu)$*, by induction on* ρ*; and*
3. *projection of a flat rewrite over a coinitial flat rewrite* $(\rho \,/^3\, \sigma)$ *by induction on* σ*, as follows:*

$$\begin{array}{lcllcl}
\mu \,/^1\, \nu & \stackrel{\text{def}}{=} & \mu/\nu & \mu \,/^1\, (\rho_1 \,;\, \rho_2) & \stackrel{\text{def}}{=} & (\mu \,/^1\, \rho_1) \,/^1\, \rho_2 \\
\nu \,/^2\, \mu & \stackrel{\text{def}}{=} & \nu/\mu & (\rho_1 \,;\, \rho_2) \,/^2\, \mu & \stackrel{\text{def}}{=} & (\rho_1 \,/^2\, \mu) \,;\, (\rho_2 \,/^2\, (\mu \,/^1\, \rho_1)) \\
\rho \,/^3\, \mu & \stackrel{\text{def}}{=} & \rho \,/^2\, \mu & \rho \,/^3\, (\sigma_1 \,;\, \sigma_2) & \stackrel{\text{def}}{=} & (\rho \,/^3\, \sigma_1) \,/^3\, \sigma_2
\end{array}$$

Note that $/^3$ generalizes $/^2$ and $/^1$ in the sense that $\mu \,/^1\, \rho = \mu \,/^3\, \rho$ and $\rho \,/^2\, \mu = \rho \,/^3\, \mu$. With these definitions, the key equation $(\rho_1 \,;\, \rho_2) \,/^3\, \sigma = (\rho_1 \,/^3\, \sigma) \,;\, (\rho_2 \,/^3\, (\sigma \,/^3\, \rho_1))$ can be shown to hold.

From this point on, we overload ρ/σ to stand for either of these projection operators. The key equation ensures that this abuse of notation is harmless. In the following, we mention some important properties of projection for flat rewrites. First, projection of a rewrite over a sequence, and of a sequence over a rewrite, obey the expected equations $\rho/(\sigma_1 \,;\, \sigma_2) = (\rho/\sigma_1)/\sigma_2$ and $(\rho_1 \,;\, \rho_2)/\sigma = (\rho_1/\sigma) \,;\, (\rho_2/(\sigma/\rho_1))$. Second, flat permutation equivalence is a congruence with respect to projection: more precisely, if $\rho \sim \sigma$ then $\tau/\rho = \tau/\sigma$ and $\rho/\tau \sim \sigma/\tau$. Third, the projection of a rewrite over itself is always empty; specifically $\rho/\rho \sim (\rho^{\mathsf{tgt}})^\flat$. Finally, an important property is that $\rho \,;\, (\sigma/\rho) \sim \sigma \,;\, (\rho/\sigma)$, corresponding to a strong form of confluence. The proof of these properties is technical, by induction on the structure of the rewrites. We do not develop the full theory of projection for flat rewrites here for lack of space (see Section E in [2] for more details).

Projection for Arbitrary Rewrites. As a final step, the projection operator of Def. 28 may be extended to arbitrary rewrites by flattening first. The proof of Prop. 30 relies crucially on the properties of projection for flat rewrites and on Thm. 20; it may be found in Section G in [2].

▶ **Definition 29** (Projection operator for arbitrary rewrites). *Let* ρ, σ *be arbitrary coinitial rewrites. Their projection is defined as* $\rho /\!\!/ \sigma \stackrel{\text{def}}{=} \rho^\flat/\sigma^\flat$*.*

▶ **Proposition 30** (Properties of projection for arbitrary rewrites).

1. *Projection of a rewrite over a sequence and of a sequence over a rewrite obey the expected equations* $\rho /\!\!/ (\sigma_1 \,;\, \sigma_2) = (\rho /\!\!/ \sigma_1) /\!\!/ \sigma_2$ *and* $(\rho_1 \,;\, \rho_2) /\!\!/ \sigma = (\rho_1 /\!\!/ \sigma) \,;\, (\rho_2 /\!\!/ (\sigma /\!\!/ \rho_1))$*.*
2. *Projection commutes with abstraction and application, that is:*
   - **2.1** $(\lambda x.\rho) /\!\!/ (\lambda x.\sigma) \approx \lambda x.(\rho /\!\!/ \sigma)$*, and more precisely* $(\lambda x.\rho) /\!\!/ (\lambda x.\sigma) \overset{\flat}{\hookleftarrow}{}^* \lambda x.(\rho /\!\!/ \sigma)$*.*
   - **2.2** *If* ρ_1, σ_1 *are coinitial and* ρ_2, σ_2 *are coinitial, then* $(\rho_1\, \rho_2) /\!\!/ (\sigma_1\, \sigma_2) \approx (\rho_1 /\!\!/ \sigma_1)\, (\rho_2 /\!\!/ \sigma_2)$*, and more precisely* $(\rho_1\, \rho_2) /\!\!/ (\sigma_1\, \sigma_2) \overset{\flat}{\hookleftarrow}{}^* (\rho_1 /\!\!/ \sigma_1)\, (\rho_2 /\!\!/ \sigma_2)$*.*
3. *The projection of a rewrite over itself is always empty,* $\rho /\!\!/ \rho \approx \rho^{\mathsf{tgt}}$*.*
4. *Permutation equivalence is a congruence with respect to projection, namely if* $\rho \approx \sigma$ *then* $\tau /\!\!/ \rho = \tau /\!\!/ \sigma$ *and* $\rho /\!\!/ \tau \approx \sigma /\!\!/ \tau$*.*
5. *The key equation* $\rho \,;\, (\sigma /\!\!/ \rho) \approx \sigma \,;\, (\rho /\!\!/ \sigma)$ *holds.*

Characterization of Permutation Equivalence in Terms of Projection. Finally, we are able to characterize permutation equivalence $\rho \approx \sigma$ as the condition that the projections $\rho /\!\!/ \sigma$ and $\sigma /\!\!/ \rho$ are both empty. Indeed:

▶ **Theorem 31** (Projection equivalence). *Let ρ, σ be arbitrary coinitial rewrites. Then $\rho \approx \sigma$ if and only if $\rho /\!/ \sigma \approx \sigma^{\mathsf{tgt}}$ and $\sigma /\!/ \rho \approx \rho^{\mathsf{tgt}}$.*

Proof. ($\Rightarrow$) Suppose that $\rho \approx \sigma$. Then, by Prop. 30, $\rho /\!/ \sigma \approx \sigma /\!/ \sigma \approx \sigma^{\mathsf{tgt}}$. Symmetrically, $\sigma /\!/ \rho \approx \rho^{\mathsf{tgt}}$. ($\Leftarrow$) Let $\rho /\!/ \sigma \approx \sigma^{\mathsf{tgt}}$ and $\sigma /\!/ \rho \approx \rho^{\mathsf{tgt}}$. Then, by Prop. 30, $\rho \approx \rho\,;\rho^{\mathsf{tgt}} \approx \rho\,;(\sigma /\!/ \rho) \approx \sigma\,;(\rho /\!/ \sigma) \approx \sigma\,;\sigma^{\mathsf{tgt}} \approx \sigma$. ◀

Since flattening and projection are computable, Thm. 20 and Thm. 31 together provide an **effective method to decide permutation equivalence** $\rho \approx \sigma$ for arbitrary rewrites. Indeed, to test whether $\rho /\!/ \sigma \approx \sigma^{\mathsf{tgt}}$, note by Thm. 20 that this is equivalent to testing whether $\rho /\!/ \sigma \sim (\sigma^{\mathsf{tgt}})^{\flat}$, so it suffices to check that $\rho /\!/ \sigma$ is *empty*, *i.e.* it contains no rule symbols. This is justified by the fact that if μ has no rule symbols and $\mu \sim \rho$, then ρ has no rule symbols (See Lem. 162 in [2]).

7 Related Work and Conclusions

As mentioned in the introduction, proof terms were introduced by van Oostrom and de Vrijer for first-order left-linear rewrite systems to study equivalence of reductions in [17] and [15, Chapter 9]. They are inspired in Rewriting Logic [13]. In the setting of HORs, Hilken [6] introduces rewrites for $\beta\eta$-reduction together with a notion of permutation equivalence for those rewrites. He does not study permutation equivalence for arbitrary HORs nor formulate notions of projection. Hilken does, however, justify his equations through a categorical semantics. We have already discussed Bruggink's work extensively [4, 3]. Another attempt at devising proof terms for HOR by the authors of the present paper is [1]. The latter uses a term assignment for a minimal modal logic called Logic of Proofs (LP), to model rewrites. LP is a refinement of S4 in which the modality $\Box A$ is refined to $[s]A$, where s is said to be a witness to the proof of A. The intuition is that terms and rewrites may be seen to belong to different stages of discourse; rewrites verse about terms. Terms are typed with simple types and rewrites are typed with a modal type $[s]A$ where the term s is the source term of the rewrite. However, the notion of substitution that is required for subject reduction is arguably ad-hoc. In particular, substitution of a rewrite $\rho : s \twoheadrightarrow s' : A$ for x in another rewrite $\sigma : t \twoheadrightarrow t' : A$ is *defined* as the composed rewrite $\rho\{x\backslash t\}\ ;\ s'\{x\backslash\!\backslash\sigma\}$, where ρ is substituted for x in $\underline{t}$ followed by σ where s' is substituted for x.

Future work. It would be of interest to develop tools based on the work presented here for reasoning about computations in higher-order rewriting, as has recently been explored for first-order rewriting [9, 10]. One downside is that our rewrites cannot be treated as terms in a higher-order rewrite system. Indeed, rewrites are not defined modulo $\beta\eta$ (for good reason since an expression such as $(\lambda x.\rho)\,\sigma$ should not be subject to β reduction).

One problem that should be addressed is that of formulating *standardization* (see *e.g.* [15, Section 8.5]) using rewrites. This amounts to giving a procedure that reorders the steps of a rewrite ρ, yielding a rewrite ρ^* in which outermost steps are performed before innermost ones. Standardization finds canonical representatives of $\approx$-equivalence classes, in the sense that $\rho \approx \sigma$ if and only if $\rho^* = \sigma^*$. The flattening rewrite system of Section 5 is a first approximation to standardization, since $\rho \approx \sigma$ if and only if $\rho^{\flat} \sim \sigma^{\flat}$. In a preliminary version of this work, we proposed a procedure to compute canonical representatives of $\approx$-equivalence classes, based on the idea of repeatedly converting $\mu\,;\nu$ into $\mu'\,;\nu'$ whenever $\nu \Leftrightarrow \xi\,;\nu'$ and $\mu' \Leftrightarrow \mu\,;\xi$, an idea reminiscent of *greedy decompositions* [5]. Unfortunately, this procedure does not always terminate, due to the fact that rewrites may have infinitely long “unfoldings”;

for instance, if $\varrho : \mathbf{c} \rightarrowtriangle \mathbf{c}$ and $\vartheta : \mathbf{f}(x) \rightarrowtriangle \mathbf{d}$ then $\vartheta(\mathbf{c}) : \mathbf{f}(\mathbf{c}) \rightarrowtriangle \mathbf{d}$ is equivalent to arbitrarily long rewrites of the form $\mathbf{f}(\varrho) \mathbin{;} \ldots \mathbin{;} \mathbf{f}(\varrho) \mathbin{;} \vartheta(\mathbf{c})$. A terminating procedure should probably rely on a measure based on the notion of *essential development* [16, Definition 11].

Another avenue to pursue is to characterize permutation equivalence via *labelling*. The application of a rewrite step leaves a witness in the term itself, manifested as a decoration (a label). These labels thus collect and record the history of a computation. By comparing them one can determine whether two computations are equivalent. Labelling equivalence for first-order rewriting is studied by van Oostrom and de Vrijer in [17] and [15, Chapter 9].

We have given semantics to rewrites via Higher-Order Rewriting Logic. A categorical semantics for a similar notion of rewrite and permutation equivalence was presented by Hirshowitz [7] (projection equivalence and flattening are not studied though). Our $s\{x\backslash\!\backslash\rho\}$ is called *left whiskering* and $\rho\{x\backslash s\}$ *right whiskering*, using the terminology of 2-category theory. These are then used to define $\rho\{x\backslash\!\backslash\!\backslash\sigma\}$. A precise relation between the two notions of rewrite should be investigated.

References

1 Pablo Barenbaum and Eduardo Bonelli. Rewrites as terms through justification logic. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 11:1–11:13. ACM, 2020.

2 Pablo Barenbaum and Eduardo Bonelli. Reductions in higher-order rewriting and their equivalence. *CoRR*, 2022.

3 Sander Bruggink. Residuals in higher-order rewriting. In Robert Nieuwenhuis, editor, *Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings*, volume 2706 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2003.

4 Sander Bruggink. *Equivalence of reductions in higher-order rewriting*. PhD thesis, Utrecht University, 2008. Available from: `http://www.ti.inf.uni-due.de/publications/bruggink/thesis.pdf`.

5 P. Dehornoy, F. Digne, E. Godelle, D. Krammer, and J. Michel. *Foundations of Garside Theory*. EMS tracts in mathematics. European Mathematical Society, 2015. Available from: `https://books.google.com.ar/books?id=7ec_SGVzNhEC`.

6 Barney P. Hilken. Towards a proof theory of rewriting: The simply typed 2λ-calculus. *Theor. Comput. Sci.*, 170(1-2):407–444, 1996.

7 Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *Log. Methods Comput. Sci.*, 9(3), 2013.

8 Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, Utrecht University, 1980.

9 Christina Kohl and Aart Middeldorp. Protem: A proof term manipulator (system description). In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 31:1–31:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

10 Christina Kohl and Aart Middeldorp. Composing proof terms. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 337–353. Springer, 2019.

11 Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul*. PhD thesis, Université de Paris 7, 1978.

12 Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

13 José Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

14 Tobias Nipkow. Higher-order critical pairs. In *Proceedings 1991 Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–343. IEEE Computer Society, 1991.

15 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

16 Vincent van Oostrom. Normalisation in weakly orthogonal rewriting. In Paliath Narendran and Michaël Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings*, volume 1631 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1999.

17 Vincent van Oostrom and Roel C. de Vrijer. Four equivalent equivalences of reductions. *Electron. Notes Theor. Comput. Sci.*, 70(6):21–61, 2002.