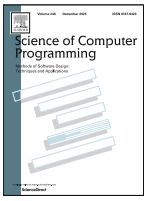
Science of Computer Programming 254 (2026) 103522

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico



Typed term evaluation systems with refinements for contextual improvement

Koko Muroya [a], Makoto Hamana [b,*]

[a] Ochanomizu University, Japan
[b] Kyushu Institute of Technology, Japan

ARTICLE INFO

Keywords:
Term rewriting
Evaluation contexts
Critical pair analysis

ABSTRACT

For a programming language, term rewriting appears in two distinct roles: run-time rewriting, referred to as evaluation, and compile-time rewriting, referred to as refinement. While evaluation specifies operational semantics, refinement models program transformations and optimisations and must therefore be validated with respect to evaluation.

This paper develops a rewriting-theoretic proof methodology for establishing contextual improvement, a quantitative correctness criterion that ensures that refinement preserves observable results and does not increase the number of evaluation steps. The methodology is centred around the notion of local coherence, a sufficient condition that enables local reasoning about the interaction between evaluation and refinement.

To formulate and analyse this interaction, we introduce Term Evaluation Systems (TES) and Term Evaluation and Refinement Systems (TERS), which make evaluation contexts, refinement rules, and values explicit within a uniform rewriting-theoretic framework. We show that local coherence can be established by critical pair analysis, thereby reducing contextual improvement to a finite and systematic analysis of rewrite rules.

The theory is developed both in a first-order setting and in a simply-typed second-order setting. The first-order setting isolates the core ideas, while the second-order case addresses additional technical challenges arising from variable binding, metavariables, and higher-order critical pairs. The applicability of the methodology is demonstrated through a range of examples from functional programming, including calculi with effects and sharing.

1. Introduction

Term rewriting provides a general and well-established model of computation. In the context of programming languages, rewriting typically appears in two distinct roles: run-time rewriting, which we refer to as evaluation, and compile-time rewriting, which we refer to as refinement.

The difference between evaluation and refinement can be characterised by the way rewrite rules are applied inside contexts. Evaluation is constrained by evaluation contexts, whereas refinement can be applied under arbitrary contexts. This distinction can be summarised by the following inference rules:

$$\frac{(l \to r) \in \mathcal{E} \quad E \in Ectx}{E[l\theta] \to_{\mathcal{E}} E[r\theta]} \qquad \frac{(l \Rightarrow r) \in \mathcal{R} \quad C \in Ctx}{C[l\theta] \Rightarrow_{\mathcal{R}} C[r\theta]}$$

* Corresponding author.
E-mail addresses: kmuroya@is.ocha.ac.jp (K. Muroya), hamana@csn.kyutech.ac.jp (M. Hamana).

https://doi.org/10.1016/j.scico.2026.103522
Received 27 December 2024; Received in revised form 22 May 2026; Accepted 2 June 2026
Available online 10 June 2026

Here, θ is a substitution, and evaluation contexts $E \in Ectx$ restrict where evaluation rules may be applied, typically enforcing a deterministic evaluation order, while refinement rules are applicable under any context $C \in Ctx$ and model compile-time program transformations.

When refinement is intended to preserve or improve the behaviour of programs, it must be validated with respect to evaluation. In this paper, we address this validation problem using the quantitative notion of contextual improvement introduced by Sands [1,2]. Contextual improvement not only requires preservation of observable results, but also guarantees that the number of evaluation steps is not increased by refinement.

Our main contribution is a rewriting-theoretic proof methodology for establishing contextual improvement. The central concept of this methodology is local coherence, a sufficient condition that enables local reasoning about the interaction between evaluation and refinement. We show that local coherence can be systematically analysed using critical pair analysis, thereby providing a concrete and effective method for proving contextual improvement.

To formulate this methodology precisely, we introduce the frameworks of Term Evaluation Systems (TES) and Term Evaluation and Refinement Systems (TERS). These frameworks make evaluation contexts, refinement rules, and values explicit, and allow evaluation and refinement to be treated within a uniform rewriting-theoretic setting. The role of TES and TERS in this work is not to propose a new rewriting paradigm, but to provide a formal foundation for the analysis of contextual improvement.

The development proceeds stepwise. We first present a first-order setting to isolate the core ideas and proof techniques. We then extend the framework to a simply-typed second-order setting, where genuinely new technical issues arise due to variable binding, metavariables, and second-order critical pairs. This extension demonstrates that the proposed methodology scales to typed second-order abstract syntax, while keeping the focus on contextual improvement rather than on the expressiveness of the type system itself.

1.1. Overview

First we provide an overview of the new frameworks of *Term Evaluation Systems (TES)* and *Term Evaluation and Refinement Systems (TERS)* we formulate in this paper, using examples to illustrate their structure. We also demonstrate our main result: a method for deriving contextual improvement through local coherence.

The standard left-to-right call-by-value lambda-calculus is a TES. Terms t, t' including values v are defined as below, and the call-by-value evaluation strategy is specified using evaluation contexts E and one evaluation rule $\to$:

$$v ::= \lambda x.t \qquad t, t' ::= x \mid v \mid t\ t' \qquad E ::= \Box \mid E\ t \mid v\ E \qquad (\lambda x.t)\ v \to t[v/x].$$

Values v appearing in this specification play a significant role. The definition of evaluation contexts notably includes the clause $v\ E$ where the left sub-term v is restricted to values. This ensures the left-to-right evaluation of application $t\ t'$; the right sub-term t' can be evaluated only after the left sub-term t has been evaluated to a value. Additionally, the redex $(\lambda x.t)\ v$ restricts the right sub-term v to values. This ensures the call-by-value evaluation of application.

A simplified computational lambda-calculus λ_{ml*} [3] is a TERS. Its terms are either values v, v' or computations p, p', and its evaluation (which has been studied [4]) is specified using evaluation contexts E and two evaluation rules $\to$:

$$v, v' ::= x \mid \lambda x.p \qquad p, p' ::= \mathtt{return}(v) \mid \mathtt{let}\ x = p\ \mathtt{in}\ p' \mid v\ v' \qquad E ::= \Box \mid \mathtt{let}\ x = E\ \mathtt{in}\ p$$

$$(\lambda x.p)\ v \to p[v/x] \qquad \mathtt{let}\ x = \mathtt{return}(v)\ \mathtt{in}\ p \to p[v/x].$$

We can observe that evaluation contexts constrain where evaluation rules can be applied, namely in the sub-term p of $\mathtt{let}\ x = p\ \mathtt{in}\ p'$. Again, values in evaluation rules assure the call-by-value evaluation of application and let-binding.

Originally, the calculus λ_{ml*} is specified by equations rather than evaluation. Directed equations can be seen as the following five refinement rules $\Rightarrow$:

$$(\lambda x.p)\ v \Rightarrow p[v/x] \qquad \mathtt{let} x = \mathtt{return}(v)\ \mathtt{in} p \Rightarrow p[v/x]$$

$$\lambda x.v\ x \Rightarrow v (x \text{is not free in} v) \qquad \mathtt{let} x = p\ \mathtt{in} \mathtt{return}(x) \Rightarrow p$$

$$\mathtt{let}\ x_1 = (\mathtt{let}\ x_2 = p_2\ \mathtt{in}\ p_1)\ \mathtt{in}\ p_3 \Rightarrow \mathtt{let}\ x_2 = p_2\ \mathtt{in}\ \mathtt{let}\ x_1 = p_1\ \mathtt{in}\ p_3.$$

While the first two rules represent β-conversion, the third one represents η-conversion. The fourth one removes the trivial let-binding, and the last one flattens let-bindings. We can observe that the last three rules *simplify* terms.

We now have a TERS of λ_{ml*} that has both evaluation and refinement. We are now interested in whether refinement is *valid* with respect to evaluation. Our goal here is namely to prove contextual improvement: that is, for any refinement $t \Rightarrow_{\mathcal{R}} u$ and any context $C \in Ctx$, if the evaluation of $C[t]$ terminates, then the evaluation of $C[u]$ terminates with no more evaluation steps.

To prove contextual improvement, we would need to analyse how each evaluation step interferes with the refinement $t \Rightarrow_{\mathcal{R}} u$. This amounts to analysing how each evaluation rule $l \to r$ can *conflict with* each refinement rule $l' \Rightarrow r'$. This is exactly what critical pair analysis is targeted at.

Critical pair analysis is usually for proving local confluence, which is a fundamental property of term rewriting. It firstly enumerates the situation where two rewrite rules conflict with each other. It then checks if the two conflicting rewritings can be *joined*. This is illustrated in Fig. 1 (left), where the joining part is depicted in dashed arrows, and '$*$' means an arbitrary number of rewriting steps.

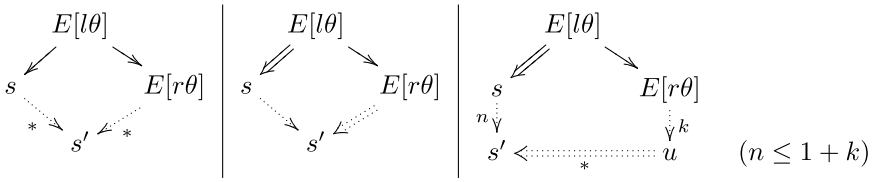


Fig. 1. Joinability for local confluence, commutation and local coherence.

In our development, we exploit critical pair analysis for proving contextual improvement, and more specifically for proving ***local coherence***. The analysis is targeted at conflicts between evaluation $\to$ and refinement $\Rightarrow$. We analyse if these conflicts can be *joined* using evaluation and refinement; see Fig. 1 (right). To ensure improvement, our notion of local coherence asserts that the joining part satisfies the inequality $1 + k \geq n$ about the number of evaluation steps.

To prove the joinability for local coherence, we need to be careful with evaluation contexts. We need to show that the $1 + k$ evaluation steps $E[l\theta] \to_{\mathcal{E}} E[r\theta] \overset{k}{\to}_{\mathcal{E}} u$ can be *simulated* by the n evaluation steps $s \overset{n}{\to}_{\mathcal{E}} s'$. Naively, this can be done by showing that the evaluation rule $l \to r$ can also be applied to the term s. This, however, involves making sure that the rule $l \to r$ can be applied *inside an evaluation context*. This is not a trivial issue; the evaluation context E might be modified by the refinement $E[l\theta] \Rightarrow_{\mathcal{R}} s$. The refinement should not turn an evaluation context into a non-evaluation context (see Definition 2.9 ((ii))).

Note that local coherence can be seen as a generalisation of *commutation* [5]; see Fig. 1 (middle). Commutation is the case where $k = 0$, $n = 1$, and allowing only one step of refinement $\Rightarrow_{\mathcal{R}}$ instead of $\overset{*}{\Rightarrow}_{\mathcal{R}}$.

1.2. Contributions

The contributions of this paper can be summarised as follows.

1. We introduce the frameworks of Term Evaluation Systems (TESs) and Term Evaluation and Refinement Systems (TERSs), which make the distinction between evaluation and refinement explicit and provide a uniform rewriting-theoretic setting for reasoning about their interaction.
2. We identify local coherence as a sufficient condition for contextual improvement, thereby providing a clear and modular criterion for validating refinement with respect to evaluation.
3. We show that local coherence can be established by critical pair analysis, yielding a concrete and systematic proof methodology for contextual improvement.
4. We develop the theory both in a first-order setting and in a simply-typed second-order setting. The first-order setting isolates the core ideas, while the second-order setting addresses additional technical challenges arising from variable binding, metavariables, and higher-order critical pairs.
5. We demonstrate the applicability of the methodology through a variety of examples from functional programming, including calculi with effects and sharing, showing that contextual improvement can be verified in practice by local reasoning.

1.3. Organisation

This paper is the fully reworked and extended version of the conference paper [6]. The paper is organised as follows.

Section 2 develops the theory of first-order Term Evaluation and Refinement Systems. This setting serves to introduce the core definitions and proof techniques in a simple form, closely related to standard first-order term rewriting.

Section 3 extends the framework to a simply-typed second-order setting. This extension incorporates variable binding, metavariables, and types, and addresses the additional technical issues that arise in higher-order rewriting, including higher-order critical pairs. The main results on contextual improvement and local coherence continue to hold in this setting.

Section 4 presents a further example, namely the call-by-need lambda-calculus, illustrating how the methodology applies to calculi with sharing and non-trivial evaluation strategies.

Section 5 discusses the relationship between TES-based evaluation and existing approaches to rewriting strategies, including innermost and context-sensitive rewriting. The purpose of this section is to clarify the scope of the proposed framework rather than to provide an exhaustive comparison.

Finally, Sections 6 and 7 discuss related work, conclusions, and directions for future research.

2. First-order term evaluation and refinement systems

2.1. Preliminaries

Let $\mathbb{N}$ be the set of natural numbers. For any $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$ (mind the starting point); for example, $[0] = \emptyset$, $[1] = \{1\}, [2] = \{1, 2\}$. We write $\overline{A}$ for a sequence $A_1, \ldots, A_n$, and $|\overline{A}|$ for its length (i.e. n). The empty sequence is denoted by ϵ.

Given a binary relation $\to$ on a set S, let $\overset{*}{\to}$ denote the reflexive and transitive closure of $\to$. For any $k \in \mathbb{N}$, $\overset{k}{\to}$ denotes the k-fold composition of $\to$. An element $x \in S$ is a *normal form* (with respect to $\to$), if there exists no element $x' \in S$ such that $x \to x'$. Let $\mathsf{NF}(\to)$ denote the set of normal forms with respect to $\to$.

2.2. Evaluation and refinement

Let Σ be a *signature*. Each element $f \in \Sigma$ comes with an *arity* $n \in \mathbb{N}$; we write $f : n$. (First-order) terms are defined by the grammar $t ::= x \mid f(t_1, \ldots, t_n)$ where x is a variable and $f : n$. Let $\mathrm{T}_\Sigma\mathrm{V}$ be the set of all terms. A term is *closed* if it has no occurrence of variables. A term is *linear* if no variable occurs more than once. The notation $FV(t)$ denotes the set of all variables occurring in a term t.

A *position* of a term is given by a (possibly empty) sequence of positive numbers, in the usual manner. Concatenation of sequences p, q is denoted by pq or $p.q$. Let $\mathcal{P}os(t)$ be the set of all positions in a term t. We write $s[t]_p$ for the term that is obtained by replacing the sub-term of s at the position p with t. We write $s|_p$ for the sub-term of s at the position p.

A *substitution* θ is given by a sequence $\{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$ where $x_1, \ldots, x_k$ are distinct variables. We write $\mathsf{subst}\,\theta$ when θ is a substitution. Let $t\theta$ denote the term where all occurrences of $x_1, \ldots, x_k$ in t are replaced by $t_1, \ldots, t_k$, respectively. We call $t\theta$ an *instance* of t.

A *context* is a term that involves exactly one *hole* $\Box$. Let Ctx be the set of contexts. Let $C[t]$ denote the term where the hole $\Box$ of $C \in Ctx$ is replaced by t. A set $\mathcal{C}$ of contexts is *closed under substitutions* if $C \in \mathcal{C}$ implies $C\theta \in \mathcal{C}$ for any $\mathsf{subst}\,\theta$, and *closed under composition* if $C, C' \in \mathcal{C}$ implies $C[C'] \in \mathcal{C}$. The set $\mathcal{C}$ is *inductive* if any $C \in \mathcal{C}$ is $\Box$ or of the form $f(t_1, \ldots, t_{i-1}, C', t_{i+1}, \ldots, t_n)$ such that $C' \in \mathcal{C}$ and $f(t_1\theta, \ldots, t_{i-1}\theta, \Box, t_{i+1}\theta, \ldots, t_n\theta) \in \mathcal{C}$ for any $\mathsf{subst}\,\theta$. Note that an inductive set of contexts is closed under substitutions and composition.

Term evaluation systems (TES) can now be defined, as the standard term rewriting with the new restriction imposed by means of *evaluation contexts*.

Definition 2.1 (TES)**.** A *term evaluation system* is a tuple $(\Sigma, \mathcal{E}, Ectx, Val)$ consisting of

- a signature Σ,
- a set $\mathcal{E}$ of *evaluation rules*, where $(l \to r) \in \mathcal{E}$ with $l, r \in \mathrm{T}_\Sigma\mathrm{V}$ such that
  (i) l is not a variable, and
  (ii) every free variable occurring in r also occurs in l,
- a set $Ectx \subseteq Ctx$ of *evaluation contexts* that is closed under substitutions, closed under composition and inductive, and
- a set $Val \subseteq \mathsf{NF}(\to_{\mathcal{E}})$ of *values* that satisfies (i) $v \in Val$ implies $v\theta \in Val$ for any $\mathsf{subst}\,\theta$, and (ii) it comes with an equivalence relation $=_{Val} \subseteq Val \times Val$, where:
- the *evaluation relation* $\to_{\mathcal{E}} \subseteq \mathrm{T}_\Sigma\mathrm{V} \times \mathrm{T}_\Sigma\mathrm{V}$ is defined as follows:

$$\frac{\mathsf{subst}\,\theta \quad (l \to r) \in \mathcal{E} \quad E \in Ectx}{E[l\theta] \to_{\mathcal{E}} E[r\theta]}$$

Values specify which normal forms of $\to_{\mathcal{E}}$ are regarded as *successful* results. For example, in a TES for arithmetic, a term $x + y$ is a normal form but it is not deemed a successful result. The equivalence relation $=_{Val}$ specifies *observations* of these results in terms of equivalence classes. For example, when the syntactic equality $\equiv$ is used, each value $v \in Val$ becomes a distinct observation. On the other hand, when the total relation $\top$ is used, values are all identified; this means that successful termination is the only possible observation.

The evaluation relation $\to_{\mathcal{E}}$ is closed under evaluation contexts in $Ectx$ and under substitutions, thanks to $Ectx$ being inductive.

Each TES can be equipped with refinement, which resembles general, unrestricted, term rewriting.

Definition 2.2 (TERS)**.** A *term evaluation and refinement system* is a tuple $(\Sigma, \mathcal{E}, \mathcal{R}, Ectx, Val)$ consisting of

- a TES $(\Sigma, \mathcal{E}, Ectx, Val)$, and
- a set $\mathcal{R}$ of *refinement rules* where $(l \Rightarrow r) \in \mathcal{R}$ with $l, r \in \mathrm{T}_\Sigma\mathrm{V}$ such that
  (i) l is not a variable, and
  (ii) every free variable occurring in r also occurs in l.

The *refinement relation* $\Rightarrow_{\mathcal{R}} \subseteq \mathrm{T}_\Sigma\mathrm{V} \times \mathrm{T}_\Sigma\mathrm{V}$ is defined as follows:

$$\frac{\mathsf{subst}\,\theta \quad (l \Rightarrow r) \in \mathcal{R} \quad C \in Ctx}{C[l\theta] \Rightarrow_{\mathcal{R}} C[r\theta]}$$

We often simply write a TES as $(\mathcal{E}, Val)$ and a TERS as $(\mathcal{E}, \mathcal{R}, Val)$. The refinement relation $\Rightarrow_{\mathcal{R}}$ is closed under substitutions, and closed under arbitrary contexts in Ctx.

For an evaluation rule $l \to r$ or a refinement rule $l \Rightarrow r$, we refer to l as "lhs" and r as "rhs".

Example 2.1 (List append)**.** We define a TERS **Append** as follows.

Signature Σ	$[]: 0,\ (:): 2,\ (+\!\!+): 2$
Values Val	$V ::= [] \mid V : V$ with $=_{Val}$ defined by $\dfrac{}{[] =_{Val} []}$ $\quad \dfrac{V_1 =_{Val} V_1' \quad V_2 =_{Val} V_2'}{V_1 : V_2 =_{Val} V_1' : V_2'}$
Evaluation contexts $Ectx$	$E ::= \Box \mid E +\!\!+ t \mid E : t \mid V : E$
Evaluation rules $\mathcal{E}$	**Refinement rules** $\mathcal{R}$
$[] +\!\!+ ys \to ys$	$(xs +\!\!+ ys) +\!\!+ zs \Rightarrow xs +\!\!+ (ys +\!\!+ zs)$
$(x : xs) +\!\!+ ys \to x : (xs +\!\!+ ys)$	$xs +\!\!+ [] \Rightarrow xs$

This defines a well-known append function on strict lists with associativity as a refinement rule. Equations that naturally hold for lists can be considered as candidates for refinement rules.

2.3. Joinability and improvement

Evaluation is constrained by means of evaluation contexts, usually to have the evaluation relation $\to_{\mathcal{E}}$ deterministic. Bridging the gap between evaluation and refinement, we are interested in *joinability up to* $\mathcal{R}$ defined as follows. The joinability is quantitative with an extra constraint on the number of evaluation steps.

Definition 2.3 (Peaks, joinability)**.**

- An $\mathcal{E}$*-peak* is given by a triple (s_1, t, s_2) such that $t \to_{\mathcal{E}} s_1$ and $t \to_{\mathcal{E}} s_2$.
- An $(\mathcal{R}, \mathcal{E})$*-peak* is given by a triple (s_1, t, s_2) such that $t \Rightarrow_{\mathcal{R}} s_1$ and $t \to_{\mathcal{E}} s_2$.
- An $\mathcal{E}$-peak (s_1, t, s_2) is *trivial* if $s_1 \equiv s_2$ holds.
- An $(\mathcal{R}, \mathcal{E})$-peak (s_1, t, s_2) is *joinable up to* $\mathcal{R}$ if there exist $k, n \in \mathbb{N}$ and u_1, u_2 such that $s_1 \overset{m}{\to}_{\mathcal{E}} u_1$, $s_2 \overset{k}{\to}_{\mathcal{E}} u_2$, $1 + k \geq m$ and $u_2 \overset{*}{\Rightarrow}_{\mathcal{R}} u_1$.

Definition 2.4 (Rewriting properties)**.**

- A set $\mathcal{E}$ of evaluation rules is *deterministic* if every $\mathcal{E}$-peak is trivial.
- A TERS $(\mathcal{E}, \mathcal{R}, Val)$ is *locally coherent* if every $(\mathcal{R}, \mathcal{E})$-peak is joinable up to $\mathcal{R}$.

We also simply say an $(\mathcal{R}, \mathcal{E})$-peak is *joinable*, omitting "up to $\mathcal{R}$".

We formalise the key concept that relates evaluation with refinement in TERS. It is called ***(contextual) improvement*** [1]: that is, any refinement $t \Rightarrow_{\mathcal{R}} s$ cannot be distinguished by evaluation $\to_{\mathcal{E}}$ inside any contexts, and the refinement cannot increase the number of evaluation steps that are needed for termination. Observation is made according to the set Val of values and its associated equivalence relation $=_{Val}$.

Definition 2.5 (Value-invariance, improvement)**.**

- A TERS $(\mathcal{E}, \mathcal{R}, Val)$ is *value-invariant* if, for any $v \in Val$ and $s \in \mathrm{T}_{\Sigma}\mathrm{V}$,

  $$v \Rightarrow_{\mathcal{R}} s \quad \text{implies} \quad v =_{Val} s \in Val.$$

- For a TERS $(\mathcal{E}, \mathcal{R}, Val)$, $\mathcal{R}$ is an ***improvement*** w.r.t. $\mathcal{E}$ if, for any $k \in \mathbb{N}$, $v \in Val$, $t \Rightarrow_{\mathcal{R}} s$ and any $C \in Ctx$ such that $C[t], C[s]$ are closed terms,

  $$C[t] \overset{k}{\to}_{\mathcal{E}} v \quad \text{implies} \quad C[s] \overset{m}{\to}_{\mathcal{E}} v'$$

  for some $m \in \mathbb{N}$ and $v' \in Val$ such that $v =_{Val} v'$ and $k \geq m$.

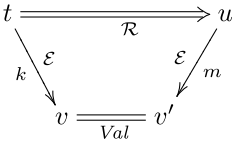
An example of non-local coherent TERS is given simply by $\mathcal{E} = \{f(x) \to 1\}$, $\mathcal{R} = \{f(x) \Rightarrow 0\}$ because an $(\mathcal{R},\mathcal{E})$-peak $(0, f(x), 1)$ cannot be joinable. Moreover, $\mathcal{R}$ is not a contextual improvement.

Directly proving improvement is challenging due to the universal quantification over all contexts. Our first main theorem addresses this challenge by providing a sufficient condition for improvement in TERSs, based on rewriting theory.

Theorem 2.1 (Sufficient condition for improvement: first-order ver)**.** *Let $(\mathcal{E}, \mathcal{R}, Val)$ be a TERS. If $\mathcal{E}$ is deterministic, and $(\mathcal{E}, \mathcal{R}, Val)$ is value-invariant and locally coherent, then the set $\mathcal{R}$ of refinement rules is an improvement w.r.t. the set $\mathcal{E}$ of evaluation rules.*

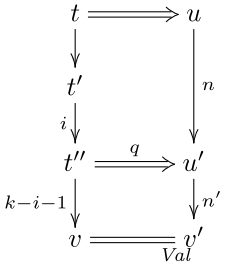
Proof. We define a predicate $P(k) \triangleq$ (if $t \Rightarrow_{\mathcal{R}} u$ and $t \overset{k}{\to}_{\mathcal{E}} v \in Val$ then there exists m, v' such that $u \overset{m}{\to}_{\mathcal{E}} v' \in Val, v =_{Val} v'$ and $k \geq m$).

We firstly prove that for any $t, u \in \mathrm{T}_{\Sigma}\mathrm{V}, k \in \mathbb{N}$, $P(k)$ holds by induction on k. The situation is depicted as follows:

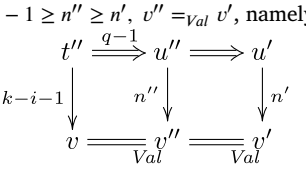


- *Base case.* When $k = 0$, we have $t = v$. Because the TERS $(\mathcal{E}, \mathcal{R})$ is value-invariant, we have $u \in Val$ and $v =_{Val} u$. We can take $m = 0$.
- *Inductive case.* When $k > 0$, there exists $t' \in \mathrm{T}_\Sigma\mathrm{V}$ such that $t \to_\mathcal{E} t' \overset{k-1}{\to}_\mathcal{E} v$. Because the TERS $(\mathcal{E}, \mathcal{R})$ is locally coherent, the $(\mathcal{R}, \mathcal{E})$-peak (u, t, t') is joinable up to $\mathcal{R}$; namely there exist $t'', u' \in \mathrm{T}_\Sigma\mathrm{V}$ and $i, q, n \in \mathbb{N}$ such that $t' \overset{i}{\to}_\mathcal{E} t''$, $u \overset{n}{\to}_\mathcal{E} u'$, $t'' \overset{q}{\Rightarrow}_\mathcal{R} u'$ and $1 + i \geq n$. Because the TERS $(\mathcal{E}, \mathcal{R})$ is deterministic, t'' must appear in the sequence $t' \overset{k-1}{\to}_\mathcal{E} v$, and hence $t \to_\mathcal{E} t' \overset{i}{\to}_\mathcal{E} t'' \overset{k-i-1}{\to}_\mathcal{E} v$. We define a predicate $Q(q) \triangleq$ (if $t'' \overset{q}{\Rightarrow}_\mathcal{R} u'$ and $t'' \overset{k-i-1}{\to}_\mathcal{E} v \in Val$ then there exist n', v' such that $u' \overset{n'}{\to}_\mathcal{E} v' \in Val, v =_{Val} v'$ and $k - i - 1 \geq n'$).

  We prove that for any $q \in \mathbb{N}$, $Q(q)$ holds by induction on q. The situation is depicted as follows:



  - *Base case.* When $q = 0$, $t'' = u'$. We can take $n' = k - i - 1$ and $v' = v$. Since $1 + i \geq n$, we have $k \geq n + n'$.
  - *Inductive case.* When $q > 0$, by the I.H. $Q(q-1)$, we have $u'' \overset{n''}{\to}_\mathcal{E} v''$ such that $v'' =_{Val} v$ and $k - i - 1 \geq n''$. Therefore, $k \geq n'' + i + 1 \geq n''$. We have $t'' \overset{q-1}{\Rightarrow}_\mathcal{R} u'' \Rightarrow_\mathcal{R} u'$ for some $u'' \in \mathrm{T}_\Sigma\mathrm{V}$. Therefore, we can apply the I.H. $P(n'')$ of the outer induction. There exist n', v' such that $u' \overset{n'}{\to}_\mathcal{E} v' \in Val$, $k - i - 1 \geq n'' \geq n'$, $v'' =_{Val} v'$, namely



    By $1 + i \geq n$ and $k \geq n'' + i + 1$, we finally have $k \geq n'' + n \geq n' + n$.

  As a result, we have $u \overset{n+n'}{\to}_\mathcal{E} v'$ such that $v =_{Val} v'$ and $k \geq n + n'$. We can take $m = n + n'$.

  Since $\Rightarrow_\mathcal{R}$ is closed under any contexts, $t \Rightarrow_\mathcal{R} u$ implies $C[t] \Rightarrow_\mathcal{R} C[u]$ for any $C \in Ctx$. Therefore, $t \Rightarrow_\mathcal{R} u$ and $C[t] \overset{k}{\to}_\mathcal{E} v$ imply $C[u] \overset{m}{\to}_\mathcal{E} v'$ such that $k \geq m$ and $v =_{Val} v'$, for any $v \in Val$. □

This theorem requires proving determinism, value-invariance and local coherence. To establish determinism, a well-known syntactic condition called *orthogonality* [7, Section 4] is particularly useful. A set of rules is orthogonal if there is no overlap between any two rules and the left-hand side of every rule is linear. Every orthogonal TERS is confluent. When $\mathcal{E}$ is orthogonal, proving determinism reduces to showing that each term can be uniquely decomposed into an evaluation context and a redex. Checking value-invariance is usually straightforward, because in many computational calculi, values are designed so that refinement rewriting cannot be applied to them. Finally, we will show that local coherence can be shown by the critical pair analysis in Section 2.4.

Example 2.2 (Append, continued)**.** The TERS **Append**'s set $\mathcal{E}$ of evaluation rules in Example 2.1 is orthogonal. The evaluation contexts are defined to uniquely decompose a context and a redex. Therefore, it is deterministic. It is obviously value-invariant because any cons-list (i.e. term generated by (:) and []) cannot be rewritten by $\mathcal{R}$.

The following examples show another interesting aspect of refinement rules.

Example 2.3 (Ones)**.** Let **Ones** be the TERS defined as follows.

Signature Σ	$(:) : 2,\ \mathtt{nil}, 1, \mathtt{ones} : 0,\ \mathtt{ns} : 1$
Values Val	$V ::= 1 \mid V : t$ with $=_{Val}$ defined by $\dfrac{}{1 =_{Val} 1}$ $\quad \dfrac{V =_{Val} V' \quad t, t' \in \mathrm{T}_\Sigma\mathrm{V}}{V : t =_{Val} V' : t'}$
Evaluation contexts $Ectx$	$E ::= \Box \mid \mathtt{ns}(E) \mid E : t$
Evaluation rules $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$\mathtt{ones} \to 1 : \mathtt{ones}$	$\mathtt{ones} \Rightarrow \mathtt{ns}(1)$
$\mathtt{ns}(n) \to n : \mathtt{ns}(n)$	

This defines lazy lists rather than the strict lists in Example 2.1. The refinement $\mathtt{ones} \Rightarrow \mathtt{ns}(1)$ appears to represent a denotational semantic equivalence on infinite lists, i.e., the denotation of both sides is the infinite list of 1s [8]. We can take such rules as refinements and show that they indeed constitute an improvement. This refinement rule and its intent do not represent an equivalence of infinite lists, but rather a behavioural equivalence of ones and ns(1).

In this sense, the notion of improvement is broader than that of inductive theorems [9], which is an equality on finite terms. It is closely related to equality on streams as discussed in [10]. Actually, examples in [10] can be formulated as TERSs and many of the examples can be shown to be contextual improvement by our method.

Example 2.4 (Divergence). Let **Div** be the TERS defined as follows.

Signature Σ	$\Omega : 0,\ 0 : 0$
Values Val	$V ::= 0$ with $=_{Val}$ defined by $\dfrac{}{0 =_{Val} 0}$
Evaluation contexts $Ectx$	$E ::= \Box$
Evaluation rule $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$\Omega \to \Omega$	$\Omega \Rightarrow 0$

The refinement rule induces contextual improvement. This means that divergence (Ω) inside any context can be turned into a value.

2.4. Critical pair analysis for local coherence

2.4.1. Critical pairs

The definition of critical pairs is standard; it resembles the definition of critical pairs for commutation [5]. Note that critical pairs are generated by two kinds of overlaps, due to asymmetry of $(\mathcal{R}, \mathcal{E})$-peaks.

Definition 2.6 (Unifiers).

- A *unifier* between t and u is a substitution θ such that $t\theta = u\theta$.
- A *most general unifier* between t and u is given by a unifier θ between t and u such that, for any unifier σ between t and u, there exists a substitution σ' such that $\sigma = \theta\sigma'$.

We say that a rule $l_1 \to_1 r_1$ is a *variant* of a rule $l_2 \to_2 r_2$ if there exists an injective renaming ρ of variables such that $(l_1, r_1) = (l_2\rho, r_2\rho)$.

Definition 2.7 (Overlaps). Let $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{R}, \mathcal{E}\}$. Given rules $(l_1 \to_1 r_1) \in \mathcal{X}_1$, $(l_2 \to_2 r_2) \in \mathcal{X}_2$, a position p of l_1, and a substitution θ, a quadruple $(l_1 \to_1 r_1, l_2 \to_2 r_2, p, \theta)$ is an $(\mathcal{X}_1, \mathcal{X}_2)$-*overlap* if it satisfies the following.

- The rules $l_1 \to_1 r_1$ and $l_2 \to_2 r_2$ do not have common variables.
- If $p = \varepsilon$, the rules $l_1 \to_1 r_1$ and $l_2 \to_2 r_2$ are not variants of each other.
- The sub-term $l_1|_p$ is not a variable.
- The substitution θ is a most general unifier between $l_1|_p$ and l_2.

Definition 2.8 (Critical pairs).

- The *critical pair* generated by an $(\mathcal{R}, \mathcal{E})$-overlap $(l_1 \Rightarrow r_1, l_2 \to r_2, p, \theta)$ is an $(\mathcal{R}, \mathcal{E})$-peak $(r_1\theta, l_1\theta, (l_1\theta)[r_2\theta]_p)$.
- The *critical pair* generated by an $(\mathcal{E}, \mathcal{R})$-overlap $(l_1 \to r_1, l_2 \Rightarrow r_2, p, \theta)$ is an $(\mathcal{R}, \mathcal{E})$-peak $((l_1\theta)[r_2\theta]_p, l_1\theta, r_1\theta)$.

Lemma 2.1. *If a critical pair (t_1, s, t_2) is joinable, then for any substitution θ, $(t_1\theta, s\theta, t_2\theta)$ is a joinable $(\mathcal{R}, \mathcal{E})$-peak.*

Proof. We have a joinable $(\mathcal{R}, \mathcal{E})$-peak (t_1, s, t_2). Since refinement and evaluation are closed under substitutions, $(t_1\theta, s\theta, t_2\theta)$ is also a joinable $(\mathcal{R}, \mathcal{E})$-peak. □

2.4.2. Critical pair theorem

To obtain the so-called critical pair theorem, we need to impose extra conditions on TERS that are summarized below.

Definition 2.9 (Well-behaved TERS). A TERS $(\Sigma, \mathcal{E}, \mathcal{R}, Ectx, Val)$ is *well-behaved* if it satisfies the following.

(i) For any $C_1, C_2 \in Ctx$, if $C_1[C_2] \in Ectx$ then $C_1, C_2 \in Ectx$.
(ii) For any $E \in Ectx$ and $C' \in Ctx$, if $E \Rightarrow_{\mathcal{R}} C'$ then $C' \in Ectx$.
(iii) For any $(l \to_{\mathcal{E}} r) \in \mathcal{E}$, l is linear.
(iv) For any $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$,
  (a) both l and r are linear, and
  (b) for any $x \in FV(l)$, if $l\{x \mapsto \Box\} \in Ectx$ then $r\{x \mapsto \Box\} \in Ectx$.

The condition (i) is typically satisfied by inductively-defined evaluation contexts. The condition (ii) was already discussed in Section 1.1. The other conditions are technical (see Remark 2.1 for some details), but these are straightforward to verify.

Theorem 2.2 (Critical pair theorem). *A well-behaved TERS is locally coherent if and only if all its critical pairs are joinable.*

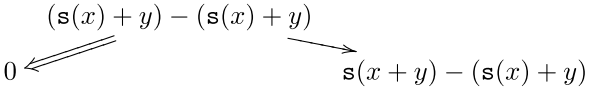
Proof. The proof is deferred to the proof of the more general second-order version (Theorem 3.2). □

Remark 2.1 (On linearity conditions)**.** For a TERS to be well-behaved, its evaluation rules must be left-linear, and its refinement rules must be linear (see Definition 2.9). This is because non-linear variables can lead to non-joinable $(\mathcal{R},\mathcal{E})$-peaks that are caused by rule applications at parallel positions, and are hence not instances of a critical pair. Joining such peaks would require applications of evaluation rules at parallel positions, but some applications may fail, due to the restriction imposed by evaluation contexts.

These linearity conditions could potentially be relaxed for some sets of evaluation contexts, but here we observe that a reasonable set of evaluation contexts and values already requires the linearity conditions. Without the linearity conditions, the set indeed leads to non-joinable $(\mathcal{R},\mathcal{E})$-peaks that are not instances of a critical pair. Let a TERS $\mathcal{E}_{\mathsf{S}}$ be defined as follows.

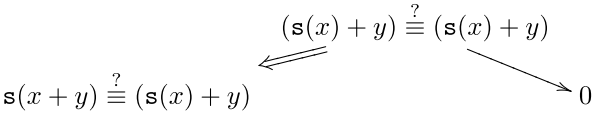
Signature Σ	$+\colon 2,\ -\colon 2,\ \overset{?}{\equiv}\colon 2,\ \mathtt{s}\colon 1,\ 0\colon 0$
Values *Val*	$V ::= 0 \mid \mathtt{s}(V)$ with $=_{Val}$ defined by $\dfrac{}{0 =_{Val} 0}$ $\dfrac{V =_{Val} V'}{\mathtt{s}(V) =_{Val} \mathtt{s}(V')}$
Evaluation contexts *Ectx*	$E ::= \Box \mid \mathtt{s}(E) \mid E + t \mid E - t \mid v - E$
Evaluation rules $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$0 + x \to x$	$x - x \Rightarrow 0$
$\mathtt{s}(x) + y \to \mathtt{s}(x + y)$	
$0 - x \to 0$	
$\mathtt{s}(x) - \mathtt{s}(y) \to x - y$	
$x \overset{?}{\equiv} x \to 0$	

The non-left-linear refinement rule $x - x \Rightarrow 0$ induces the following non-joinable $(\mathcal{R},\mathcal{E})$-peak.



In the term $\mathtt{s}(x + y) - (\mathtt{s}(x) + y)$, the sub-term $\mathtt{s}(x) + y$ cannot be evaluated, because $\mathtt{s}(x + y)$ is not a value.

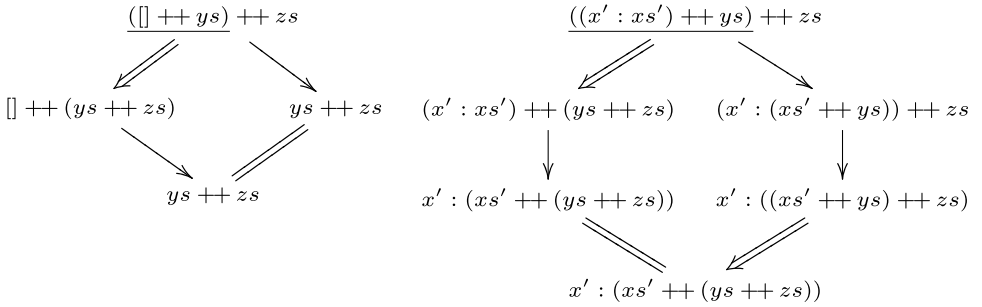
The non-left-linear evaluation rule $x \overset{?}{\equiv} x \to 0$ induces the following non-joinable $(\mathcal{R},\mathcal{E})$-peak.



In the term $\mathtt{s}(x + y) \overset{?}{\equiv} (\mathtt{s}(x) + y)$, the sub-term $\mathtt{s}(x) + y$ cannot be evaluated, because $\mathtt{s}(x + y)$ is not a value.

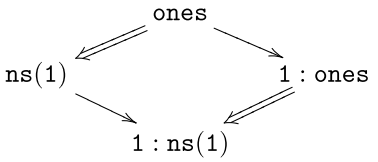
2.4.3. Examples

Example 2.5 (Append, continued)**.** The TERS **Append** in Example 2.1 has the following two joinable critical pairs.



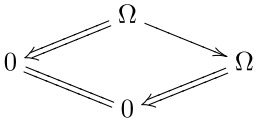
Because the TERS is well-behaved, it is locally coherent. By Theorem 2.1, we conclude that the refinement rule expressing the associativity of append is an *improvement* w.r.t. its evaluation.

Example 2.6 (Ones, continued)**.** The TERS **Ones** in Example 2.3 has the following joinable critical pair.



The TERS is well-behaved, and hence it is locally coherent. Because it is deterministic and value-invariant, by Theorem 2.1, the refinement rule $\mathtt{ones} \Rightarrow \mathtt{ns}(1)$ is an improvement.

Example 2.7 (Divergence, continued)**.** The TERS **Div** in Example 2.4 has the following joinable critical pair.



The TERS is well-behaved, and hence it is locally coherent. Because it is deterministic and value-invariant, by Theorem 2.1, the refinement rule $\Omega \Rightarrow 0$ is an improvement.

3. Simply-typed second-order term evaluation and refinement systems

Next we extend our framework to the second-order case. By second-order we mean to use second-order abstract syntax [11,12], i.e. syntax with variable binding and metavariables. It allows us to formally deal with higher-order term languages as in second-order algebraic theories [13] and second-order computation systems [14,15].

3.1. Types

We assume that $\mathcal{A}$ is a set of *atomic types* (e.g. Bool, Nat, etc.). We also assume a set of *type constructors* together with arities $n \in \mathbb{N}, n \geq 1$. A *type signature* is a set of atomic types and type constructors. The sets of *molecular types* (mol types, for short) $\mathcal{T}_0$ and *types* $\mathcal{T}$ are generated by the following rules:

$$\frac{b \in \mathcal{A}}{b \in \mathcal{T}_0} \qquad \frac{b_1, \ldots, b_n \in \mathcal{T}_0 \quad T \text{ } n\text{-ary type constructor}}{T(b_1, \ldots, b_n) \in \mathcal{T}_0} \qquad \frac{a_1, \ldots, a_n, b \in \mathcal{T}_0}{a_1, \ldots, a_n \to b \in \mathcal{T}}$$

Example 3.1. Assume atomic types Bool and Nat, and a unary type constructor List. Then the set $\mathcal{T}_0$ of all molecular types is the least set satisfying

$$\mathcal{T}_0 = \{\mathsf{Bool}, \mathsf{Nat}\} \cup \{\mathsf{List}(a) \mid a \in \mathcal{T}_0\}$$

Namely, given a molecular type a, we have a molecular type $\mathsf{List}(a)$.

3.2. Meta-terms

A *signature* Σ is a set of function symbols of the form

$$f : (\overline{a_1} \to b_1), \ldots, (\overline{a_m} \to b_m) \to c \tag{1}$$

where all a_i, b_i, c are molecular types (thus any function symbol is of up to second-order type).

A ***metavariable*** is a variable declared as $M : \overline{a} \to b$ (written as capital letters $M, N, K, \ldots$). A ***variable*** of a molecular type is merely called variable (written usually $x, y, \ldots$, or sometimes written x^b when it is of type b). The raw syntax is given as follows.

- ***Terms*** have the form $t ::= x \mid x.t \mid f(t_1, \ldots, t_n)$.
- ***Meta-terms*** extend terms to $t ::= x \mid x.t \mid f(t_1, \ldots, t_n) \mid M[t_1, \ldots, t_n]$.

The form $x.t$ is abstraction where $x.$ is a variable binding. We may write $x_1, \ldots, x_n.t$ for $x_1.\cdots.x_n.t$, and we assume ordinary α-equivalence for bound variables. The last form is called a *meta-application*, meaning that when we instantiate $M : \overline{a} \to b$ with a term s, free variables of s (which are of types $\overline{a}$) are replaced with (meta-)terms $t_1, \ldots, t_n$.

Example 3.2. The simply-typed λ-terms on the set Ty of simple types generated by a set BTy of base types are modelled in our setting as follows. We define the set $\mathcal{A}$ of atomic types as BTy. We suppose a type constructor Arr to encode arrow types. The set of Ty of all simple types for the λ-calculus is the least set satisfying

$$\mathrm{Ty} = \mathrm{BTy} \cup \{\mathsf{Arr}(a, b) \mid a, b \in \mathrm{Ty}\}.$$

Thus, the set $\mathcal{T}_0$ of all molecular types is Ty. The λ-terms are given by a signature

$$\Sigma_{\mathrm{stl}} = \left\{ \begin{array}{ll} \lambda_{a,b} & : (a \to b) \to \mathsf{Arr}(a, b) \\ @_{a,b} & : \mathsf{Arr}(a, b), a \to b \end{array} \;\middle|\; a, b \in \mathrm{Ty} \right\}$$

An example λ-term is $@_{a,b}(\lambda_{a,b}(x^a.\, M[x]), N)$ that involves two metavariables $M : a \to b, N : a$.

We use the following notational convention throughout the paper. We will present a signature by omitting mol type subscripts a, b (see also more detailed account [15]). For example, simply writing function symbols λ and @, we mean $\lambda_{a,b}$ and $@_{a,b}$ in Σ_{stl} having appropriate mol type subscripts a, b. We will also use @ as an infix operator, thus the example λ-term $@_{a,b}(\lambda_{a,b}(x^a.\, M[x]), N)$ is denoted by $\lambda(x.\, M[x])@N$.

$$\frac{y : b \in \Gamma}{\Theta \rhd \Gamma \vdash \ y \ : b} \qquad \frac{(M : a_1, \ldots, a_m \to b) \in \Theta \quad \Theta \rhd \Gamma \vdash \ t_i \ : a_i \quad (1 \le i \le m)}{\Theta \rhd \Gamma \vdash \ M[t_1, \ldots, t_m] \ : b}$$

$$\frac{f : (\overline{a_1} \to b_1), \cdots, (\overline{a_m} \to b_m) \to c \in \Sigma \quad \Theta \rhd \Gamma, \overline{x_i : a_i} \vdash \ t_i \ : b_i \quad (1 \le i \le m)}{\Theta \rhd \Gamma \vdash \ f(\ \overline{x_1^{a_1}}.t_1, \ldots, \overline{x_m^{a_m}}.t_m\ ) \ : c}$$

Fig. 2. Typing rules of meta-terms.

Remark 3.1. In our framework of second-order TERSs, types are stratified into atomic types, molecular types, and types. Molecular types are obtained by closing atomic types under the type constructors. Molecular types play the role of "base types" in ordinary type theories (see the types in (1)). However, in our setting we require base types to be generated from even more primitive entities. For this reason, we introduce atomic types as the most primitive ones and then form molecular types from them.

Another reason is that we wish to use distinct terminology for object-level and meta-level types—that is, to distinguish the "base types" of the object-level target calculus from the "base types" (which we call atomic types) of the meta-level second-order TERS.

Molecular types exactly correspond to the base types in [16,17].

Remark 3.2. The syntactic structure of meta-terms and substitution for abstract syntax with variable binding was first introduced by Aczel [18]. This formal language allowed him to consider a general framework of rewrite rules for calculi with variable binding. This influenced Klop's rewrite system of Combinatory Reduction System (CRS) [19], where meta-terms are also used as the syntax.

Second-order abstract syntax provides a primitive mechanism of variable binding, represented by expressions of the form $x.t$. Importantly, this binder is not tied to any particular syntactic construct such as λ-abstraction. The role of $x.t$ is solely to introduce a scope for the variable x within the term t; all syntactic constructs that require binding make use of this single primitive. This design deliberately separates the notions of primitive binding, and language-level constructs that employ binding, which assume particular evaluation rules (such as the β-reduction). As a consequence, it offers a uniform treatment of a wide range of syntactic forms without multiplying special-purpose binding operators. λ-abstraction is merely one instance as exemplified in Example 3.2. Because binding is primitive and uniform, many common constructs in programming languages are expressed directly:

- let expressions $\mathtt{let}(s, x.t)$ (Example 3.4)
- do/monadic sequencing $\mathtt{do}(u, x.t)$ (Example 3.5)
- effect handlers binding continuation arguments $\mathtt{handler}(c, x.k)$ (Example 3.5)

In all these cases, $x.t$ is the same primitive binding mechanism. The outer constructor (`let`, `do`, `handler`, …) determines special meaning; the binding itself is provided uniformly.

For a meta-term t, the set of its free variables is denoted by $FV(t)$, the set of its bound variables is denoted by $BV(t)$, and the set of its free metavariables is denoted by $FM(t)$.

A metavariable context Θ is a sequence of (metavariable:type)-pairs, and a variable context Γ is a sequence of (variable:mol type)-pairs. A typing judgement is of the form

$$\Theta \rhd \Gamma \vdash \ t \ : b.$$

A meta-term t is *well-typed* by the typing rules Fig. 2. A sequence of types may be empty in the above definition. The empty sequence is denoted by $()$, which may be omitted, e.g., $b_1, \ldots, b_m \to c$, or $() \to c$. The latter case is simply denoted by c.

3.3. Contextual sets of meta-terms

In the proofs of this paper, we will use the structure of type and context-indexed sets. A *contextual set* A is a family $\{A_b(\Gamma) \mid b \in \mathcal{T}_0,\ \text{variable context } \Gamma\}$ of sets indexed by types and variable contexts. Set operations such as $\cup, \uplus, \cap$ are extended to contextual sets by index-wise constructions, such as $A \cup B$ by $\{A_b(\Gamma) \cup B_b(\Gamma) \mid b \in \mathcal{T}_0,\ \text{variable context } \Gamma\}$. Throughout this paper, for a contextual set A, we simply write $a \in A$ if there exist b, Γ such that $a \in A_b(\Gamma)$. The indices are usually easily inferred from context. A map $f : A \to B$ between contextual sets is given by indexed functions $\{f_b(\Gamma) : A_b(\Gamma) \to B_b(\Gamma) \mid b \in \mathcal{T}_0,\ \text{variable context } \Gamma\}$. Examples of contextual sets are the contextual set of meta-terms $\mathrm{M}_\Sigma\Theta$ and of terms $\mathrm{T}_\Sigma\mathrm{V}$ defined by

$$(\mathrm{M}_\Sigma\Theta)_b(\Gamma) \triangleq \{t \mid \Theta \rhd \Gamma \vdash t : b\}, \qquad (\mathrm{T}_\Sigma\mathrm{V})_b(\Gamma) \triangleq \{t \mid \ \rhd \Gamma \vdash t : b\}.$$

for a given metavariable context Θ and a given signature Σ.

A term is *closed* if it has no occurrence of variables.

A *position* of a meta-term is given by a (possibly empty) sequence of positive numbers. The set $\mathcal{P}os(t)$ of all positions in a meta-term t is inductively defined as follows.

$$\mathcal{P}os(x) = \{\varepsilon\}$$

$$\mathcal{P}os(x.t) = \{\varepsilon\} \cup \{1.p \mid p \in \mathcal{P}os(t)\}$$
$$\mathcal{P}os(f(t_1, \dots, t_l)) = \{\varepsilon\} \cup \{i.p \mid i \in [l], p \in \mathcal{P}os(t_i)\}$$
$$\mathcal{P}os(M[t_1, \dots, t_m]) = \{\varepsilon\} \cup \{i.p \mid i \in [m], p \in \mathcal{P}os(t_i)\}$$

We write $s[t]_p$ for the meta-term that is obtained by replacing the sub-term of s at the position p with t. We write $s|_p$ for the sub-term of s at the position p.

We say that a position p in a meta-term t is a *metavariable position* if $t|_p$ is a meta-application, i.e., $t|_p = M[t_1, \dots, t_n]$. This description includes the case $t|_p = M$ where $n = 0$, for which we identify $M[\,]$ with just a metavariable M.

A *substitution* θ is given by a sequence $[M_1 \mapsto \overline{x_1}.s_1, \dots, M_k \mapsto \overline{x_k}.s_k]$ such that: (i) $M_1, \dots, M_k$ are distinct metavariables, and (ii) for some Θ, Γ and for each $i \in [k]$, $(M_i : \overline{a_i} \to b_i) \in \Theta$ and $\Theta \rhd \Gamma, \overline{x_i : a_i} \vdash s_i : b_i$ hold. We call $M_1, \dots, M_k$ the *domain* of θ and write $Dom(\theta)$. Given a meta-term $\Theta, M_1 : \overline{a_1} \to b_1, \dots, M_k : \overline{a_k} \to b_k \rhd \Gamma \vdash t : b$, a meta-term $t\theta$ is defined by

$$x\theta = x$$
$$(f(\overline{x_1}.t_1, \dots, \overline{x_l}.t_l))\theta = f(\overline{x_1}.t_1\theta, \dots, \overline{x_l}.t_l\theta)$$
$$(M[t_1, \dots, t_m])\theta = \begin{cases} s_i\{(x_i)_1 \mapsto t_1\theta, \dots, (x_i)_m \mapsto t_m\theta\} & (\exists i \in [k].\ M \equiv M_i) \\ M[t_1\theta, \dots, t_m\theta] & (\text{otherwise}) \end{cases}$$

The meta-term $s_i\{(x_i)_1 \mapsto t_1\theta, \dots, (x_i)_m \mapsto t_m\theta\}$ is the result of standard (capture-avoiding) substitution for variables. We call $t\theta$ an *instance* of t.

Given two substitutions $\theta_1 = [\overline{M \mapsto \overline{x}.s}, \overline{K \mapsto \overline{z}.t}]$ and $\theta_2 = [\overline{N \mapsto \overline{y}.u}, \overline{K \mapsto \overline{z}.t}]$ such that the concatenation $\overline{M}, \overline{N}, \overline{K}$ is a sequence of distinct metavariables, their *composition* $\theta_1\theta_2$ is defined by $\theta_1\theta_2 = [\overline{M \mapsto \overline{x}.s\theta_2}, \overline{K \mapsto \overline{z}.t\theta_2}, \overline{N \mapsto \overline{y}.u}]$. It satisfies $(t\theta_1)\theta_2 = t(\theta_1\theta_2)$.

A *renaming* is given by a sequence $\rho = [\overline{M \mapsto N}]$ such that ρ is injective, and $\overline{M} \cap \overline{N} = \varnothing$.

We will use Miller's notion of higher-order pattern [20], which makes higher-order unification decidable. A meta-term is a *higher-order pattern* (*pattern* for short) if each occurrence of meta-application must be of the form $M[x_1, \dots, x_m]$, where $x_1, \dots, x_m$ are distinct bound variables.

Definition 3.1 (Contexts)**.** Suppose that for each type σ, there exists a constant $\Box^\sigma$ of type σ called a *hole*. A *(typed) context* C is a well-typed meta-term that involves exactly one hole. We denote by $C : \sigma \to \tau$ when the meta-term C of type τ involves the hole $\Box^\sigma$ of type σ.

Let $C[t]$ denote the term where the hole $\Box^\sigma$ of a context C is replaced by a meta-term t of type σ. Note that, unlike substitution, the operation of filling in the context hole may cause the capture of bound variables.

Hereafter, we omit the superscript σ in a hole when it is clear from context. A context is *flat* if any prefix of the position of the hole is not a metavariable position; e.g. $f(x.\Box)$ is a flat context, but $M[\Box]$ and $M[f(x.\Box)]$ are not flat contexts. Let Ctx be the set of all contexts. We use the same properties for a set C of contexts, as in the first-order case: namely, closure under substitutions, closure under composition, and inductivity (see Section 2.2).

3.4. Syntax classes

We introduce a notion of syntactic classification for terms, typically used for distinguishing values and non-values, following [15]. In *loc. cit.*, the call-by-value lambda-calculus (dubbed λ_{value}-calculus) involves the following two syntax classes of values and non-values.

Values $V ::= x \mid \lambda(x.M)$ **Non-values** $P ::= M@N$

This also specifies two special names V, P of metavariables that are used for values and non-values.

Definition 3.2 (Syntax class)**.** We define a set *Sclass* of names for *syntax classes*. Each syntax class is associated with an inductively defined set of well-typed meta-terms specified by a BNF grammar or inference rules (cf. Example 4.1). Every metavariable is either associated with a syntax class and called "⟨syntax class name⟩ metavariable", or not associated and called *general metavariable*. We assume the following two default syntax classes:

- **values** associated with well-typed values, and
- **general** associated with the set of all well-typed meta-terms.

Therefore, any well-typed meta-term belongs to at least the general syntax class.

For example, in the case of λ_{value}-calculus, we define

$$Sclass = \{\ \textbf{values}\ V ::= x \mid \lambda(x.M), \qquad \textbf{non-values}\ P ::= M@N\ \}.$$

The metavariable V is a value metavariable, P is a non-value metavariable, and M, N are general metavariables. Substitutions must also be consistent with syntax classes.

Definition 3.3. A substitution θ is ***valid*** if for each assignment $(M \mapsto \overline{x}.t) \in \theta$, the following holds:

$$\text{(RuleSub)}\frac{\Theta \triangleright \Gamma, \overline{x_i : a_i} \vdash \ s_i \ : b_i \quad (1 \le i \le k) \qquad \mathsf{valid}\ [\overline{M \mapsto \overline{x}.s}] \\ (M_1 : (\overline{a_1} \to b_1), \ldots, M_k : (\overline{a_k} \to b_k) \triangleright \ \vdash \ \ell \to r \ : c) \in \mathcal{E}}{\Theta \triangleright \Gamma \vdash \ \ell\,[\overline{M \mapsto \overline{x}.s}] \to_{\mathcal{E}} r\,[\overline{M \mapsto \overline{x}.s}] \ : c}$$

$$\text{(Ectx)}\frac{E : b \to c \in Ectx \qquad \Theta \triangleright \Gamma \vdash \ t \to_{\mathcal{E}} t' \ : b}{\Theta \triangleright \Gamma \vdash \ E[t] \to_{\mathcal{E}} E[t'] \ : c}$$

Fig. 3. Typed second-order evaluation $\to_{\mathcal{E}}$ on meta-terms.

$$\text{(RuleSub)}\frac{\Theta \triangleright \Gamma, \overline{x_i : a_i} \vdash \ s_i \ : b_i \quad (1 \le i \le k) \qquad \mathsf{valid}\ [\overline{M \mapsto \overline{x}.s}] \\ (M_1 : (\overline{a_1} \to b_1), \ldots, M_k : (\overline{a_k} \to b_k) \triangleright \ \vdash \ \ell \Rightarrow r \ : c) \in \mathcal{R}}{\Theta \triangleright \Gamma \vdash \ \ell\,[\overline{M \mapsto \overline{x}.s}] \Rightarrow_{\mathcal{R}} r\,[\overline{M \mapsto \overline{x}.s}] \ : c}$$

$$\text{(Ctx)}\frac{C : b \to c \in Ctx \qquad \Theta \triangleright \Gamma \vdash \ t \Rightarrow_{\mathcal{R}} t' \ : b}{\Theta \triangleright \Gamma \vdash \ C[t] \Rightarrow_{\mathcal{R}} C[t'] \ : c}$$

Fig. 4. Typed second-order refinement $\Rightarrow_{\mathcal{R}}$ on meta-terms.

- let S be the associated meta-term set of M's syntax class, and
- T the associated meta-term set of t's syntax class, and
- $S \supseteq T$ holds.

We write $\mathsf{valid}\,\theta$ when θ is a valid substitution.

For example, for given a general metavariable M and a value metavariable V, a substitution $\theta : M \mapsto V$ is valid, whereas $\theta : V \mapsto M$ is not valid because the set of all values are included in the set of all meta-terms.

Composition of valid substitutions is again valid, under the assumption that each syntax class is closed under substitutions: that is, for each syntax class, if a meta-term t is included then $t\theta$ is also included, where θ is a substitution.

3.5. Evaluation and refinement

Definition 3.4. For meta-terms $\Theta \triangleright \ \vdash \ell \ : b$ and $\Theta \triangleright \ \vdash r \ : b$, an *evaluation* (resp. a *refinement*) *rule* is of the form

$$\Theta \triangleright \ \vdash \ell \to r \ : b$$

satisfying:

1. ℓ is not a variable nor a metavariable.
2. ℓ is a higher-order pattern.
3. All metavariables in r appear in ℓ.

We usually omit the context and type and simply write $\ell \to r$ for an evaluation (resp. $\ell \Rightarrow r$ for a refinement) rule.

We assume that the lhs of every rule is a higher-order pattern to make unification decidable, which is important for computing rewrite steps and critical pairs. Second-order TESs and TERSs can now be defined, in an analogous way to the first-order setting.

Definition 3.5 (Second-order TES)**.** A *second-order term evaluation system* is a tuple $(Type, \Sigma, \mathcal{E}, Ectx, Sclass)$ consisting of

(i) a type signature $Type$,
(ii) a signature Σ,
(iii) a set $\mathcal{E}$ of *evaluation rules*,
(iv) a set $Ectx \subseteq Ctx$ of flat contexts, called *evaluation contexts*, that is closed under substitutions, closed under composition and inductive, and
(v) a set $Sclass$ of syntax classes that satisfies
  (a) it includes a *value* class associated with the set Val
  (b) $Val \subseteq \mathsf{NF}(\to_{\mathcal{E}})$
  (c) $v \in Val$ implies $v\theta \in Val$ for any $\mathsf{valid}\,\theta$
  (d) it comes with an equivalence relation $=_{Val} \subseteq Val \times Val$.

The ***evaluation relation*** $\to_{\mathcal{E}} \subseteq \mathrm{M}_{\Sigma}\Theta \times \mathrm{M}_{\Sigma}\Theta$ is defined in Fig. 3.

The evaluation relation $\to_{\mathcal{E}}$ is closed under evaluation contexts in $Ectx$. It is closed under substitutions, thanks to $Ectx$ being an inductive set of flat contexts.

Definition 3.6 (Second-order TERS)**.** A *second-order term evaluation and refinement system* is a tuple $(\mathit{Type}, \Sigma, \mathcal{E}, \mathcal{R}, \mathit{Ectx}, \mathit{Sclass})$ consisting of

(i) a second-order TES $(\mathit{Type}, \Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Sclass})$, and
(ii) a set $\mathcal{R}$ of *refinement rules* that satisfies
  (a) for any $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$ and $\mathsf{valid}\,\theta$, if $l\theta$ belongs to a syntax class S then $r\theta$ belongs to S.
  (b) for any $(l \to_{\mathcal{E}} r) \in \mathcal{R}$ and $\mathsf{valid}\,\theta$, if $l\theta$ belongs to a syntax class S then $r\theta$ belongs to S.

The ***refinement relation*** $\Rightarrow_{\mathcal{R}}$ on well-typed meta-terms is defined in Fig. 4.
The refinement relation $\Rightarrow_{\mathcal{R}}$ is closed under arbitrary contexts in Ctx and valid substitutions.

3.6. Joinability and improvement

The definitions of peaks, joinability, rewriting properties, and improvement are inherited from the first-order case (see Section 2.3). Finally, the first main theorem (Theorem 2.1) also holds in the second-order setting:

Theorem 3.1 (Sufficient condition for improvement: second-order ver)**.** *If a second-order TERS* $(\Sigma, \mathcal{E}, \mathcal{R}, \mathit{Ectx}, \mathit{Sclass})$ *is deterministic, value-invariant and locally coherent, then* $\mathcal{R}$ *is an improvement w.r.t.* $\mathcal{E}$.

3.7. Examples

In the remainder of this section, we present a few examples of TERS. We may omit type scripts for simplicity.

Example 3.3 (The untyped call-by-value lambda-calculus [21])**.** With types, we can represent an untyped calculus as a TERS. A TERS **CBV**λ of Plotkin's untyped (left-to-right) call-by-value lambda-calculus is defined as follows.

Type signature $\quad \iota$

Signature $\Sigma \quad \lambda : (\iota \to \iota) \to \iota,\ @ : \iota, \iota \to \iota$

Syntax class Sclass

values $\quad V ::= x \mid \lambda(x.M) \quad$ with $=_{\mathit{Val}}$ defined by $\quad \dfrac{v \in \mathit{Val}}{x =_{\mathit{Val}} v} \quad \dfrac{}{\lambda(x.t) =_{\mathit{Val}} \lambda(y.t')}$

Evaluation contexts $\mathit{Ectx} \quad E ::= \Box \mid E@M \mid V@E$

Evaluation rule $\mathcal{E}$

$\lambda(x.M[x])@V \to M[V]$

Refinement rules $\mathcal{R}$

$\lambda(x.M[x])@V \Rightarrow M[V]$

$\lambda(x.V@x) \Rightarrow V$

Note that the notation $\lambda(x.V@x)$ using the second-order abstract syntax automatically ensures that V cannot involve the variable x, because any substitution for the metavariable V must be capture-avoiding. Note that $=_{\mathit{Val}}$ equates all values, making successful termination the only possible observation. Note also that the evaluation proceeds *from left to right* in the TERS **CBV**λ. For example, we have:

$$\begin{aligned}(\underline{\lambda(x.x)@\lambda(y.y)})@(\lambda(z.z)@\lambda(w.w)) &\to_{\mathcal{E}} \lambda(y.y)@(\underline{\lambda(z.z)@\lambda(w.w)})\\ &\to_{\mathcal{E}} \underline{\lambda(y.y)@\lambda(w.w)}\\ &\to_{\mathcal{E}} \lambda(w.w),\\ (\lambda(x.x)@\lambda(y.y))@(\underline{\lambda(z.z)@\lambda(w.w)}) &\not\to_{\mathcal{E}} (\lambda(x.x)@\lambda(y.y))@\lambda(w.w).\end{aligned}$$

The last evaluation is impossible, because $(\lambda(x.x)@\lambda(y.y))@\Box$ is not a valid evaluation context. We also emphasise that the TERS **CBV**λ is *different* from the lambda-calculus with the (leftmost) innermost strategy. Namely, no evaluation can happen inside abstraction; $\lambda(x.\Box)$ is not a valid evaluation context, and therefore $\lambda(x.\underline{\lambda(y.y)@\lambda(z.z)}) \not\to_{\mathcal{E}} \lambda(x.\lambda(z.z))$.

Example 3.4 (A simplified computational lambda-calculus λ_{ml*} [3,4])**.** A notion of evaluation for Sabry and Wadler's computational lambda-calculus λ_{ml*} [3] has been studied [4]. A TERS **Comp**λ_{ml*} of the computational lambda-calculus is defined as follows:

Type signature $\quad \iota,\ \mathsf{Arr}(-,-),\ \mathsf{T}(-)$

Signature Σ

$\lambda_{a,b} : (a \to \mathsf{T}(b)) \to \mathsf{Arr}(a,b),\ (@_{a,b}) : \mathsf{Arr}(a,b), a \to \mathsf{T}(b),\ \mathtt{let}_{a,b} : \mathsf{T}(a), (a \to \mathsf{T}(b)) \to \mathsf{T}(b),\ \mathtt{return}_a : a \to \mathsf{T}(a)$

Syntax classes Sclass

values $\quad V, V' ::= x \mid \lambda(x.P) \quad$ with $=_{\mathit{Val}}$ defined by $\quad \dfrac{v \in \mathit{Val}}{x =_{\mathit{Val}} v} \quad \dfrac{}{\lambda(x.t) =_{\mathit{Val}} \lambda(y.t')}$

computations $P, P' ::= \mathtt{return}(V) \mid \mathtt{let}(P, x.P') \mid V\,@\,V'$

Evaluation contexts $Ectx$ $E ::= \Box \mid \mathtt{let}(E, x.P)$

Evaluation rules $\mathcal{E}$

$$\lambda(x.P[x])@V \to P[V] \tag{1}$$

$$\mathtt{let}(\mathtt{return}(V), x.P[x]) \to P[V] \tag{2}$$

Refinement rules $\mathcal{R}$

$$\lambda(x.P[x])@V \Rightarrow P[V] \tag{r1}$$

$$\mathtt{let}(\mathtt{return}(V), x.P[x]) \Rightarrow P[V] \tag{r2}$$

$$\lambda(x.V\,@\,x) \Rightarrow V \tag{r3}$$

$$\mathtt{let}(P, x.\mathtt{return}(x)) \Rightarrow P \tag{r4}$$

$$\mathtt{let}(\mathtt{let}(P_1, x_1.P_2[x_1]), x_2.P_3[x_2]) \Rightarrow \mathtt{let}(P_1, x_1.\mathtt{let}(P_2[x_1], x_2.P_3[x_2])) \tag{r5}$$

Example 3.5 (Effect handlers [22])**.** A TERS **Hndl** is defined in Fig. 5, where V, V_1, V_2 are value metavariables, H is a handler metavariable, and $P, P_1, P_2, \ldots$ are computation metavariables.

We only consider two operations $\mathtt{op}_1, \mathtt{op}_2$ and two handlers: $\mathtt{handler}_1$ for catching the first operation $\mathtt{op}_1$ and $\mathtt{handler}_0$ for catching no operation, for simplicity. We change the evaluation rule (7) to be the so-called *shallow handling*; the original, *deep handling*, rule [22] can be accommodated to a TERS, but this TERS would not be well-behaved.[1] We also select the refinement rules that do not correspond to an evaluation rule and those whose lhs is a higher-order pattern.[2] The refinement rules are numbered according to the original presentation [22, Fig. 7].

3.8. Second-order critical pair analysis for local coherence

3.8.1. Critical pairs

The following definitions are analogous to those of first-order TERS, except for lifters (Definition 3.7 below) that are additional, and unifiers (Definition 3.8 below) that are defined for terms with the same free variables only. We will discuss these in Remark 3.3. The definition of critical pairs is again standard (cf. [23]), akin to commutation.

Definition 3.7 (Lifter)**.** Given a sequence $\overline{x}$ of variables and a sequence $\overline{K}$ of metavariables, an $\overline{x^a}$*-lifter of a pattern* t *away from* $\overline{K}$ is given by a substitution $\sigma = [\overline{M \mapsto \overline{y}.(M\rho)[\overline{y}, \overline{x}]}]$ where $\overline{M} = FM(t)$ and $\rho = [\overline{M \mapsto N}]$ is a renaming such that (i) $\overline{N} \cap \overline{K} = \emptyset$, and (ii) if $M_i : \overline{a_i} \to b_i$ then $N_i : \overline{a_i}, \overline{a} \to b_i$.

Definition 3.8 (Unifiers)**.**

- Given two meta-terms t, u such that $FV(t) = FV(u)$, a *unifier* between t and u is a valid substitution θ such that $t\theta = u\theta$.
- A *most general unifier* between t and u is given by a unifier θ between t and u such that, for any unifier σ between t and u, there exists a valid substitution σ' such that $\sigma = \theta\sigma'$.

Definition 3.9 (Overlaps)**.** Let $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{R}, \mathcal{E}\}$. Given rules $(l_1 \twoheadrightarrow_1 r_1) \in \mathcal{X}_1$, $(l_2 \twoheadrightarrow_2 r_2) \in \mathcal{X}_2$ and a substitution θ, a tuple $(l_1 \twoheadrightarrow_1 r_1, l_2 \twoheadrightarrow_2 r_2, p, \sigma, \theta)$ is an $(\mathcal{X}_1, \mathcal{X}_2)$*-overlap* if it satisfies the following.

- The rules $l_1 \twoheadrightarrow_1 r_1$ and $l_2 \twoheadrightarrow_2 r_2$ do not have common variables or metavariables.
- If $p = \varepsilon$, the rules $l_1 \twoheadrightarrow_1 r_1$ and $l_2 \twoheadrightarrow_2 r_2$ are not variants of each other.
- The sub-term $l_1|_p$ is not a meta-application, where p is a position of l_1.
- For $\overline{x} = FV(l_1|_p) \cap BV(l_1)$, σ is an $\overline{x}$-lifter of l_2 away from $FM(l_1)$.
- The substitution θ is a most general unifier between $l_1|_p$ and $l_2\sigma$.

Definition 3.10 (Critical pairs)**.**

- The *critical pair* generated by an $(\mathcal{R}, \mathcal{E})$-overlap $(l_1 \Rightarrow r_1, l_2 \to r_2, p, \sigma, \theta)$ is an $(\mathcal{R}, \mathcal{E})$-peak $(r_1\theta, l_1\theta, (l_1\theta)[(r_2\sigma)\theta]_p)$.
- The *critical pair* generated by an $(\mathcal{E}, \mathcal{R})$-overlap $(l_1 \to r_1, l_2 \Rightarrow r_2, p, \sigma, \theta)$ is an $(\mathcal{R}, \mathcal{E})$-peak $((l_1\theta)[(r_2\sigma)\theta]_p, l_1\theta, r_1\theta)$.

Remark 3.3. Compared to the first-order setting, the definition of critical pairs requires the use of *lifters* defined in Definition 3.7 in the higher-order setting. The following second-order TERS illustrates the necessity of lifters, which has not been clearly explained in prior work, such as [23]. The TERS is a variant of the call-by-name lambda-calculus, with its refinement rule being an "incorrect" eta-rule.

Type signature ι

[1] More specifically, the metavariable P_1 would appear twice in the rhs of the original rule of the evaluation rule (7).

[2] The refinement rule (7) in [22, Fig. 7] is the only refinement rule whose lhs is not a higher-order pattern.

Type signature $\mathsf{Bool},\ \mathsf{Arr}(-,-),\ \mathsf{Hndl}(-,-),\ \mathsf{T}(-)$

Signature Σ

$\mathtt{true}, \mathtt{false}\colon \mathsf{Bool},\ \mathtt{fun}_{a,b}\colon (a \to \mathsf{T}(b)) \to \mathsf{Arr}(a,b),\ (@_{a,b})\colon \mathsf{Arr}(a,b), a \to \mathsf{T}(b),$
$\mathtt{return}_a\colon a \to \mathsf{T}(a),\ \mathtt{op}_{1_{a,b,c}}\colon a, (b \to \mathsf{T}(c)) \to \mathsf{T}(c),\ \mathtt{op}_{0_{a,b,c}}\colon a, (b \to \mathsf{T}(c)) \to \mathsf{T}(c),$
$\mathtt{handler}_{1_{a,b,c}}\colon (a \to \mathsf{T}(b), (a', \mathsf{Arr}(b', \mathsf{T}(c')) \to \mathsf{T}(c')) \to \mathsf{Hndl}(\mathsf{T}(a), \mathsf{T}(b)),$
$\mathtt{handler}_{0_{a,b}}\colon (a \to \mathsf{T}(b)) \to \mathsf{Hndl}(\mathsf{T}(a), \mathsf{T}(b)),\ \mathtt{do}_{a,b}\colon \mathsf{T}(a), (a \to \mathsf{T}(b)) \to \mathsf{T}(b),$
$\mathtt{if}_a\colon \mathsf{Bool}, \mathsf{T}(a), \mathsf{T}(a) \to \mathsf{T}(a),\ \mathtt{with\_handle}_{a,b}\colon \mathsf{Hndl}(\mathsf{T}(a), \mathsf{T}(b)), \mathsf{T}(a) \to \mathsf{T}(b)$

Syntax classes *Sclass*

functions $F ::= x \mid \mathtt{fun}(x.P)$

values $V ::= \mathtt{true} \mid \mathtt{false} \mid F \mid H$

with $=_{Val}$ defined by $\dfrac{b \in \{\mathtt{true}, \mathtt{false}\}}{b =_{Val} b}$ $\dfrac{v \in Val}{x =_{Val} v}$ $\dfrac{}{\mathtt{fun}(x.t) =_{Val} \mathtt{fun}(y.t')}$

$$\frac{}{\mathtt{handler}_1(x.p, x.k.p_1) =_{Val} \mathtt{handler}_1(x.p', x.k.p_1')}$$

$$\frac{}{\mathtt{handler}_0(x.p) =_{Val} \mathtt{handler}_0(x.p')}$$

handlers $H ::= \mathtt{handler}_1(x.P, x.k.P_1) \mid \mathtt{handler}_0(x.P)$

computations $P, P_1, P_2 ::= \mathtt{return}(V) \mid \mathtt{op}_1(V, y.P) \mid \mathtt{op}_0(V, y.P) \mid \mathtt{do}(P_1, x.P_2)$
$\mid \mathtt{if}(V, P_1, P_2) \mid F\,@\,V \mid \mathtt{with\_handle}(H, P)$

Evaluation contexts *Ectx*

$E ::= \square \mid \mathtt{do}(E, x.P) \mid \mathtt{with\_handle}(H, E) \mid \mathsf{if}(E, t, t) \mid E@M \mid V@E$

Evaluation rules $\mathcal{E}$ where $i = 0, 1$

$$\mathtt{do}(\mathtt{return}(V), x.P[x]) \to P[V] \tag{1}$$
$$\mathtt{do}(\mathtt{op}_i(V, y.P_1[y]), x.P_2[x]) \to \mathtt{op}_i(V, y.\mathtt{do}(P_1[y], x.P_2[x])) \tag{2}$$
$$\mathtt{if}(\mathtt{true}, P_1, P_2) \to P_1 \tag{3}$$
$$\mathtt{if}(\mathtt{false}, P_1, P_2) \to P_2 \tag{4}$$
$$\mathtt{fun}(x.P[x])@V \to P[V] \tag{5}$$

In the following three rules, $h_1 \equiv \mathtt{handler}_1(x.P[x], x.k.P_1[x,k])$.

$$\mathtt{with\_handle}(h_1, \mathtt{return}(V)) \to P[V] \tag{6}$$
$$\mathtt{with\_handle}(h_1, \mathtt{op}_1(V, y.P'[y])) \to P_1[V, \mathtt{fun}(y.P'[y])] \tag{7}$$
$$\mathtt{with\_handle}(h_1, \mathtt{op}_2(V, y.P'[y])) \to \mathtt{op}_2(V, y.\mathtt{with\_handle}(h_1, P'[y])) \tag{8}$$

In the following two rules, $h_0 \equiv \mathtt{handler}_0(x.P[x])$.

$$\mathtt{with\_handle}(h_0, \mathtt{return}(V)) \to P[V] \tag{9}$$
$$\mathtt{with\_handle}(h_0, \mathtt{op}_i(V, y.P'[y])) \to \mathtt{op}_i(V, y.\mathtt{with\_handle}(h_0, P'[y])) \tag{10}$$

Refinement rules $\mathcal{R}$

$$\mathtt{do}(P, x.\mathtt{return}(x)) \Rightarrow P \tag{r3}$$
$$\mathtt{do}(\mathtt{do}(P_1, x_1.P_2[x_1]), x_2.P_3[x_2]) \Rightarrow \mathtt{do}(P_1, x_1.\mathtt{do}(P_2[x_1], x_2.P_3[x_2])) \tag{r4}$$
$$\mathtt{fun}(x.F@x) \Rightarrow F \tag{r9}$$
$$\mathtt{with\_handle}(\mathtt{handler}_0(x.P[x]), P') \Rightarrow \mathtt{do}(P', x.P[x]) \tag{r13}$$

Fig. 5. The TERS **Hndl**.

Signature Σ $\quad \lambda : (\iota \to \iota) \to \iota,\ @ : \iota, \iota \to \iota,\ 0 : \iota$

Syntax class *Sclass*

values $\quad V ::= x \quad$ with $=_{Val}$ defined by $\dfrac{}{x =_{Val} y}$

Evaluation contexts *Ectx* $\quad E ::= \Box \mid E@M \mid \lambda(x.E)$

Evaluation rule $\mathcal{E}$

$\lambda(y.M[y])@N \to M[N]$

Refinement rule $\mathcal{R}$

$\lambda(x.M'[x]@x) \Rightarrow M'[0]$

Analogous to the first-order setting, one would consider an overlap between the evaluation rule $\to$ and the refinement rule $\Rightarrow$. This would amount to taking a most general unifier between the lhs $\lambda(y.M[y])@N$ of the evaluation rule and a sub-term $M'[x]@x$ of the lhs of the refinement rule. One natural candidate of the most general unifier is:

$$\theta = [M \mapsto y.K[y], N \mapsto x, M' \mapsto x.\lambda(y.K[y])].$$

However, this is not most general; it cannot have both of the two unifiers below as instances:

$$\theta_1 = [M \mapsto y.y, N \mapsto x, M' \mapsto x'.\lambda(y.y)],$$
$$\theta_2 = [M \mapsto y.x, N \mapsto x, M' \mapsto x'.\lambda(y.x')].$$

To deal with this apparent lack of most general unifiers, we restrict the definition of unifiers so that we take unifiers of two terms t and u only when $FV(t) = FV(u)$. In the above example, this amounts to taking a unifier between terms $\lambda(y.M[y,x])@N[x]$ and $M'[x]@x$ instead of terms $\lambda(y.M[y])@N$ and $M'[x]@x$. Namely, we derive the term $\lambda(y.M[y,x])@N[x]$ from the term $\lambda(y.M[y])@N$ so that it has x as a free variable. *Lifters* are introduced for this derivation purpose; the term $\lambda(y.M[y,x])@N[x]$ is precisely an x-lifter of the term $\lambda(y.M[y])@N$.

Lemma 3.1. *Let l_1, l_2 be two patterns with no common metavariables, p be a position of l_1 such that the sub-term $l_1|_p$ is not a meta-application, $\overline{x} = FV(l_1|_p) \cap BV(l_1)$, and σ be an $\overline{x}$-lifter of l_2 away from $FM(l_1)$. The following are equivalent.*

1. $\exists\theta.\ (l_1|_p)\theta = (l_2\sigma)\theta$
2. $\exists\theta_1, \theta_2.\ (l_1|_p)\theta_1 = (l_2)\theta_2$

Moreover, $\theta|_{FM(l_1)} = \theta_1$ and $(\sigma\theta)|_{FM(l_2)} = \theta_2$ hold.

Proof. *(1) $\Longrightarrow$ (2).* We can take θ_1 to be the restriction of θ to $FM(l_1)$, i.e. $\theta_1 = \theta|_{FM(l_1)}$. We can take θ_2 to be the composition of σ and θ, namely $\theta_2 = \sigma\theta$.

(2) $\Longrightarrow$ (1). Without loss of generality, we assume that $Dom(\theta_1) \subseteq FM(l_1)$. Let ρ be the renaming associated with the lifter σ, and θ_2' be a substitution $[\overline{(M\rho) \mapsto \overline{y}, \overline{x}.\ \theta_2(M[\overline{x}])}]$. We take $\theta = \theta_1 \cup \theta_2'$. $\square$

Lemma 3.2. *If a critical pair (t_1, s, t_2) is joinable, then for any valid substitution θ, $(t_1\theta, s\theta, t_2\theta)$ is a joinable $(\mathcal{R}, \mathcal{E})$-peak.*

Proof. We have a joinable $(\mathcal{R}, \mathcal{E})$-peak (t_1, s, t_2). Because evaluation is closed under valid substitutions, and refinement satisfies $t \Rightarrow_{\mathcal{R}} u \implies t\theta \overset{*}{\Rightarrow}_{\mathcal{R}} u\theta$, $(t_1\theta, s\theta, t_2\theta)$ is also a joinable $(\mathcal{R}, \mathcal{E})$-peak. $\square$

To obtain the so-called critical pair theorem, TERSs are required to be well-behaved again. The following conditions are similar to the first-order case (see Definition 2.9), except for the last two conditions which ensure that evaluation and refinement are consistent with syntax classes.

We say that a meta-term t is *linear w.r.t. non-value metavariables* if every metavariable that is not a value metavariable occurs at most once in t. Note that the linear restriction does not apply to value metavariables, allowing a value metavariable to occur more than twice in t.

Definition 3.11 (Well-behaved TERS)**.** A TERS $(\Sigma, \mathcal{E}, \mathcal{R}, Ectx, Sclass)$ is *well-behaved* if it satisfies the following.

(i) For any $C_1, C_2 \in Ctx$, if $C_1[C_2] \in Ectx$ then $C_1, C_2 \in Ectx$.
(ii) For any $E \in Ectx$ and $C' \in Ctx$, if $E \Rightarrow_{\mathcal{R}} C'$ then $C' \in Ectx$.
(iii) For any $(l \to_{\mathcal{E}} r) \in \mathcal{E}$, l is linear w.r.t. non-value metavariables.
(iv) For any $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$,
  (a) both l and r are linear w.r.t. non-value metavariables, and
  (b) for every non-value metavariable $M : \overline{a} \to b \in FM(l)$, if $l[M \mapsto \overline{x}.\Box] \in Ectx$ then $r[M \mapsto \overline{x}.\Box] \in Ectx$.

Thanks to the linearity condition of *non-value* metavariables, we can have the distributive law of the map function on lists over list append as a well-behaved refinement rule: i.e. $\mathtt{map}(V, M)$++$\mathtt{map}(V, N) \Rightarrow \mathtt{map}(V, M$++$N)$ where the first function argument of `map` is restricted to values.

Theorem 3.2 (Critical pair theorem)**.** *A well-behaved TERS is locally coherent if and only if all its critical pairs are joinable.*

Proof. The "only if" part is straightforward. In the following, we prove the "if" part.

Take an arbitrary $(\mathcal{R},\mathcal{E})$-peak (t_1, s, t_2). Our goal is to prove that this $(\mathcal{R},\mathcal{E})$-peak is joinable. Since $s \Rightarrow_{\mathcal{R}} t_1$, there exist $p \in \mathcal{P}os(s)$, $(l \Rightarrow r) \in \mathcal{R}$ and $\mathsf{valid}\,\theta$ such that $s|_p = l\theta$, $t_1 = s[r\theta]_p$ and $s[\Box]_p \in Ctx$. We prove that the $(\mathcal{R},\mathcal{E})$-peak (t_1, s, t_2) is joinable, by induction on the length of the position p.

Base case. When $|p| = 0$, i.e. $p = \varepsilon$, we have $s = l\theta$ and $t_1 = r\theta$. Because $l\theta \to_{\mathcal{E}} t_2$, there exist $p' \in \mathcal{P}os(l\theta)$, $(l' \to r') \in \mathcal{E}$ and $\mathsf{valid}\,\theta'$ such that $(l\theta)|_{p'} = l'\theta'$, $t_2 = (l\theta)[r'\theta']_{p'}$ and $(l\theta)[\Box]_{p'} \in Ectx$. We have an $(\mathcal{R},\mathcal{E})$-peak $P = (r\theta, l\theta, (l\theta)[r'\theta']_{p'})$.

- If $p' = \varepsilon$, and $l \Rightarrow r$ and $l \to r$ are variants of each other, we have $r\theta = r'\theta'$ and the $(\mathcal{R},\mathcal{E})$-peak P is joinable.
- Otherwise, because $(l\theta)[\Box]_{p'} \in Ectx$ is a flat context, every prefix of p' but p' itself is not a metavariable position in $l\theta$.
  - If p' is a non-metavariable position of l, since l and l' are patterns, $l|_{p'}$ must not be a meta-application nor an argument of a meta-application (i.e. a variable). Therefore we have $(l|_{p'})\theta = (l\theta)|_{p'} = l'\theta'$. By Lemma 3.1, there is a unifier between $l|_{p'}$ and $l'\sigma$ for an appropriate lifter σ. The $(\mathcal{R},\mathcal{E})$-peak P is an instance of the critical pair generated by an $(\mathcal{R},\mathcal{E})$-overlap.
  - Otherwise, there exist sequences q_1, q_2, a metavariable N and a sequence $\overline{y}$ such that: $q_1 \in \mathcal{P}os(l)$, $l|_{q_1} = N[\overline{y}]$, $q_2 \in \mathcal{P}os((N[\overline{y}])\theta)$, and $p' = q_1 q_2$. Because of the condition (i) of Definition 3.11, $(l\theta)[\Box]_{p'} \in Ectx$ implies $l[\Box]_{q_1}, (N[\overline{y}])\theta[\Box]_{q_2} \in Ectx$. In particular, the latter means that $(N[\overline{y}])\theta \notin \mathsf{NF}(\to_{\mathcal{E}})$, and hence N is not a value metavariable. The metavariable N must appear at most once in both l and r, due to the condition (iv)a of Definition 3.11. If N does not appear in r, the $(\mathcal{R},\mathcal{E})$-peak P is joinable by applying the rule $l \Rightarrow r$ to t_2. Otherwise, i.e. if N appears once in r, the rule $l' \to r'$ can be applied to t_1 thanks to the condition (iv)b of Definition 3.11, and the rule $l \Rightarrow r$ can be applied to t_2, thanks to the condition (iv) of Definition 3.11. These two applications yield the same result. Therefore, we can conclude that the $(\mathcal{R},\mathcal{E})$-peak P is joinable.

Inductive case. When $|p| > 0$, we have $p = ip_t$ for some positive number i and some sequence p_t. We have either $s = f(\overline{x_1}.u_1, \dots, \overline{x_i}.u_i, \dots, \overline{x_k}.u_k)$ or $s = M[u_1, \dots, u_i, \dots, u_k]$, such that $l\theta = u_i|_{p_t}$.

First, assume that we have an $(\mathcal{R},\mathcal{E})$-peak

$$P' = (f(\overline{x_1}.u_1, \dots, \overline{x_i}.u_i[r\theta]_{p_t}, \dots, \overline{x_k}.u_k), f(\overline{x_1}.u_1, \dots, \overline{x_i}.u_i, \dots, \overline{x_l}.u_l), t_2).$$

By $s \to_{\mathcal{E}} t_2$, there exist $p' \in \mathcal{P}os(s)$, $(l' \to r') \in \mathcal{E}$ and $\mathsf{valid}\,\theta'$ such that $s|_{p'} = l'\theta'$, $t_2 = s[r'\theta']_{p'}$ and $s[\Box]_{p'} \in Ectx$. We proceed by case analysis on $p' \in \mathcal{P}os(s)$.

- When $p' = \varepsilon$, we have $s = l'\theta'$ and $t_2 = r'\theta'$.
  - If p is a non-metavariable position of l', since l and l' are patterns, $l'|_p$ must not be a meta-application nor an argument of a meta-application (i.e. a variable). Therefore we have $(l'|_p)\theta' = (l'\theta')|_p = l\theta$. By Lemma 3.1, there is a unifier between $l'|_p$ and $l\sigma$ for an appropriate lifter σ. The $(\mathcal{R},\mathcal{E})$-peak P' is an instance of the critical pair generated by an $(\mathcal{E},\mathcal{R})$-overlap.
  - Otherwise, there exist sequences q_1, q_2 and a metavariable M such that: $q_1 \in \mathcal{P}os(l')$, $l'|_{q_1} = M[\overline{y}]$, $q_2 \in \mathcal{P}os(M[\overline{y}]\theta')$, and $p = q_1 q_2$. The metavariable M appears at most once in l', due to the condition (iii) of Definition 3.11. We can apply the rule $l' \to r'$ to t_1. The substitution θ' is valid, thanks to the condition (iii) of Definition 3.11. We can also apply the rule $l \Rightarrow r$ to t_2 as many times as M appears in r'. These applications of $l' \to r'$ and $l \Rightarrow r$ yield the same result. The $(\mathcal{R},\mathcal{E})$-peak P' is therefore joinable.
- When $p' \neq \varepsilon$, i.e. $p' = i'p'_t$ for some positive number i' and some sequence p'_t, there are two possibilities.
  - When $i' = i$, by I.H., we have a joinable $(\mathcal{R},\mathcal{E})$-peak $Q = (u_i[r\theta]_{p_t}, u_i, u_i[r'\theta']_{p'_t})$. Because $f(\dots, \overline{x_i}.u_i[\Box]_{p_t}, \dots) \in Ectx$, we have $f(\dots, \overline{x_i}.\Box, \dots) \in Ectx$ too, thanks to the condition (i) of Definition 3.11. Therefore, joinability of the $(\mathcal{R},\mathcal{E})$-peak Q implies joinability of the $(\mathcal{R},\mathcal{E})$-peak P'.
  - When $i' \neq i$, we can assume that $i' < i$ without loss of generality. The $(\mathcal{R},\mathcal{E})$-peak

$$\begin{aligned} P' = (&f(\dots, \overline{x_{i'}}.u_{i'}, \dots, \overline{x_i}.u_i[r\theta]_{p_t}, \dots), f(\dots, \overline{x_{i'}}.u_{i'}, \dots, \overline{x_i}.u_i, \dots), \\ &f(\dots, \overline{x_{i'}}.u_{i'}[r'\theta']_{p'_t}, \dots, \overline{x_i}.u_i, \dots)) \end{aligned}$$

    is joinable (to $f(\dots, \overline{x_{i'}}.u_{i'}[r'\theta']_{p'_t}, \dots, \overline{x_i}.u_i[r\theta]_{p_t}, \dots)$), thanks to the condition (ii) of Definition 3.11.

Second, assume that we have an $(\mathcal{R},\mathcal{E})$-peak

$$P' = (M[u_1, \dots, u_i[r\theta]_{p_t}, \dots, u_k], M[u_1, \dots, u_i, \dots, u_l], t_2).$$

By $s \to_{\mathcal{E}} t_2$, there exist $p' \in \mathcal{P}os(s)$, $(l' \to r') \in \mathcal{E}$ and $\mathsf{valid}\,\theta'$ such that $s|_{p'} = l'\theta'$, $t_2 = s[r'\theta']_{p'}$ and $s[\Box]_{p'} \in Ectx$. We proceed by case analysis on $p' \in \mathcal{P}os(s)$.

- When $p' = \varepsilon$, $M[u_1, \dots, u_k] = l'\theta'$. Because l' is a higher-order pattern, this is impossible.
- When $p' \neq \varepsilon$, the proof is the same as the case for the $(\mathcal{R},\mathcal{E})$-peak

$$P' = (f(\overline{x_1}.u_1, \dots, \overline{x_i}.u_i[r\theta]_{p_t}, \dots, \overline{x_k}.u_k), f(\overline{x_1}.u_1, \dots, \overline{x_i}.u_i, \dots, \overline{x_l}.u_l), t_2).$$

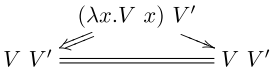
□

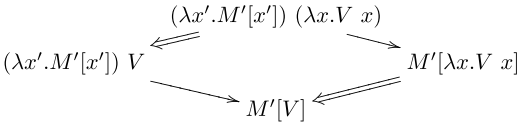
3.8.2. The three examples

The TERSs **CBV**λ, **Comp**λ_{ml*} and **Hndl** are deterministic and value-invariant. They have the refinement rule that corresponds to the so-called eta-equivalence, e.g. $\lambda(x.V\,@\,x) \Rightarrow V$. It turns an abstraction, which is a value, into a value that is identified by $=_{Val}$ to the original abstraction. Therefore these TERSs are value-invariant.

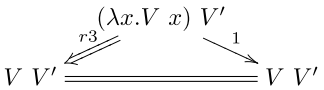
The three TERSs are well-behaved. Most of the conditions are trivially satisfied, or easy to check. The condition (ii) can be checked by straightforward induction on $E \in \mathit{Ectx}$.

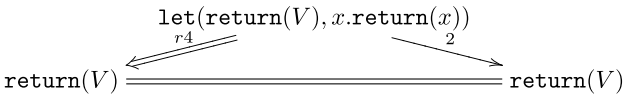
To establish local coherence, the next step is to check every critical pair is joinable. The TERS **CBV**λ has the following two joinable critical pairs, which are for the second refinement rule (the so-called eta-rule) and the sole evaluation rule.

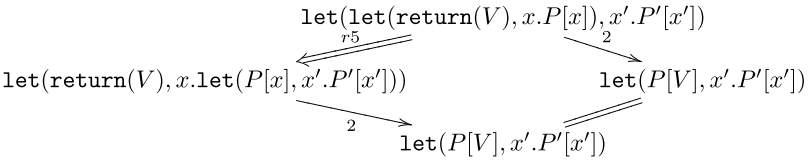




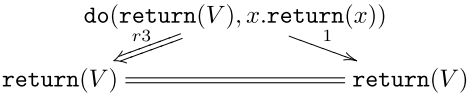
The TERS **Comp**λ_{ml*} has the following three critical pairs. In the following, arrows $\to$, $\Rightarrow$ are labelled by a number that indicates which evaluation/refinement rule is applied.

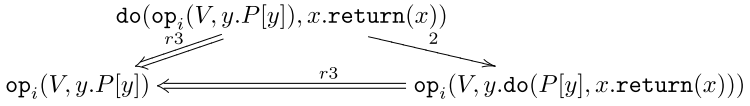


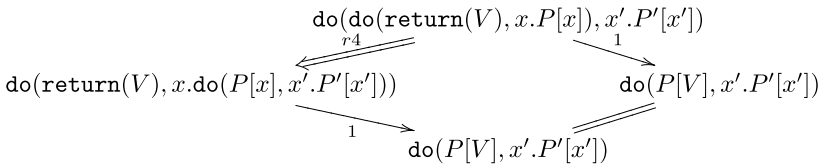


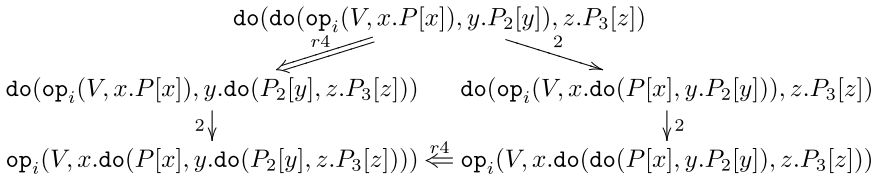


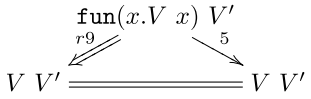
The TERS **Hndl** has the following ten joinable critical pairs; we merge some of them below using $\mathtt{op}_i$ ($i \in [2]$). In the following, arrows $\to$, $\Rightarrow$ are labelled by a number that indicates which evaluation/refinement rule is applied, and we set $h_0 \equiv \mathtt{handler}_0(x.P[x])$.

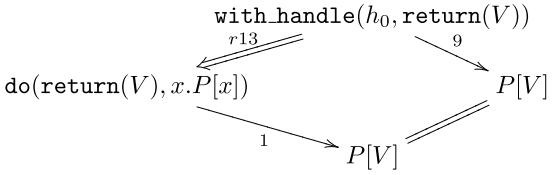


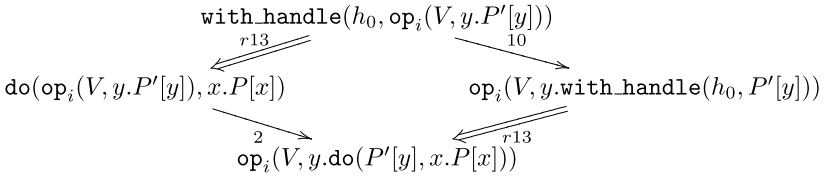










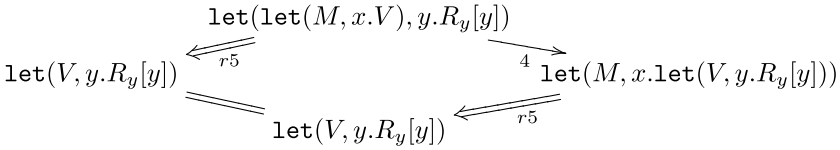


Finally, by Theorems 3.1 and 3.2, the refinement rules $\mathcal{R}$ of each of the three TERSs are an *improvement* with respect to the evaluation rules $\mathcal{E}$.

4. Further example: the call-by-need lambda-calculus

Example 4.1 (Typed call-by-need lambda-calculus [24]). A TERS **CBNeed**λ of the call-by-need lambda-calculus is defined in Fig. 6. The *residual* syntax class is the only syntax class in this paper that is not defined by BNF. A residual r for a variable x contains x only once, such that $r[\Box]_p \in \mathit{Ectx}$ for the position p of x in r. Like the last refinement rule in Example 3.3, the refinement rule (r5) uses the notation $\mathtt{let}(M, x.N)$ to ensure that N never involves the variable x.

This TERS **CBNeed**λ is deterministic, value-invariant, and also well-behaved. It has the following joinable critical pair.



Therefore, by Theorems 3.1 and 3.2, the refinement rules $\mathcal{R}$ of the TERS **CBNeed**λ are an *improvement* with respect to the evaluation rules $\mathcal{E}$.

5. Evaluation formalisms and rewriting strategies

The purpose of this section is to clarify the position of Term Evaluation Systems (TESs), the evaluation formalism introduced in this paper, with respect to existing rewriting-based approaches to modelling evaluation. In particular, we consider rewriting strategies that have been extensively studied in the term rewriting literature, such as innermost rewriting and context-sensitive rewriting.

Type signature $\iota,\ \mathsf{Arr}(-,-)$

Signature Σ

$\lambda_{a,b}\colon (a \to b) \to \mathsf{Arr}(a,b),\ (@_{a,b})\colon \mathsf{Arr}(a,b), a \to b,\ \mathtt{let}_{a,b}\colon a, (a \to b) \to b$

Syntax classes *Sclass*

answers $A ::= x \mid \lambda(x.M)$

values $V ::= \lambda(x.M) \mid \mathtt{let}(M, x.V)$

with $=_{Val}$ defined by $\dfrac{}{\lambda(x.t) =_{Val} \lambda(y.t')} \quad \dfrac{v =_{Val} v'}{\mathtt{let}(t, x.v) =_{Val} \mathtt{let}(t', x.v')}$

residuals R_x **for each variable** x

$$\frac{}{x \in R_x} \qquad \frac{r \in R_x \quad t \in \mathrm{T}_\Sigma\mathrm{V} \quad x \notin FV(t) \quad r@t \text{ is well-typed}}{r@t \in R_x}$$

$$\frac{t \in \mathrm{T}_\Sigma\mathrm{V} \quad r \in R_x \quad x \neq y \quad x \notin FV(t) \quad \mathtt{let}(t, y.r) \text{ is well-typed}}{\mathtt{let}(t, y.r) \in R_x}$$

$$\frac{r \in R_x \quad r' \in R_y \quad x \neq y \quad x \notin FV(r') \quad \mathtt{let}(r, y.r') \text{ is well-typed}}{\mathtt{let}(r, y.r') \in R_x}$$

Evaluation contexts *Ectx* $E ::= \Box \mid E@M \mid \mathtt{let}(M, x.E) \mid \mathtt{let}(E, x.E[x])$

Evaluation rules $\mathcal{E}$

$$\lambda(x.M[x])@N \to \mathtt{let}(N, x.M[x]) \tag{1}$$
$$\mathtt{let}(A, x.R_x[x]) \to R_x[A] \tag{2}$$
$$\mathtt{let}(M, x.V[x])@N \to \mathtt{let}(M, x.V[x]@N) \tag{3}$$
$$\mathtt{let}(\mathtt{let}(M, x.V[x]), y.R_y[y]) \to \mathtt{let}(M, x.\mathtt{let}(V[x], y.R_y[y])) \tag{4}$$

Refinement rules $\mathcal{R}$

$$\lambda(x.M[x])@N \Rightarrow \mathtt{let}(N, x.M[x]) \tag{r1}$$
$$\mathtt{let}(A, x.R_x[x]) \Rightarrow R_x[A] \tag{r2}$$
$$\mathtt{let}(M, x.V[x])@N \Rightarrow \mathtt{let}(M, x.V[x]@N) \tag{r3}$$
$$\mathtt{let}(\mathtt{let}(M, x.V[x]), y.R_y[y]) \Rightarrow \mathtt{let}(M, x.\mathtt{let}(V[x], y.R_y[y])) \tag{r4}$$
$$\mathtt{let}(M, x.N) \Rightarrow N \tag{r5}$$

Fig. 6. Call-by-need lambda-calculus.

The discussion in this section serves as a sanity check for the TES framework. We show that several strategy-based notions of evaluation can be encoded within TES by an appropriate choice of evaluation contexts. This demonstrates that TES is expressive enough to capture standard evaluation mechanisms studied in rewriting theory, while providing the explicit notion of evaluation context required for reasoning about contextual improvement.

No refinement rules are considered in this section, and no claims are made about improvement or optimisation properties of the rewriting strategies discussed here. Accordingly, this section is intentionally orthogonal to the main technical development of the paper, whose primary contribution is a proof methodology for establishing contextual improvement via local coherence and critical pair analysis.

We begin by recalling the basic notion of rewriting strategies and discuss innermost rewriting as a canonical example. We then consider context-sensitive rewriting and show how context-sensitive restrictions on rewrite position can be represented within TES. The encodings presented here are straightforward and do not introduce new theoretical results; rather, they serve to situate TES within the landscape of existing rewriting-based approaches to evaluation.

5.1. Innermost strategy

In term rewriting systems, a one-step rewriting strategy is a way to choose one redex to contract in each rewriting step. In this section, we discuss several rewriting strategies including context-sensitive rewriting and how TESs formalism realises strategies. First, we recall the ordinary definitions.

Definition 5.1.

(i) *A one-step rewriting strategy* F is a function F on terms such that $t = F(t)$ if t is a normal form of a rewrite system $\mathcal{R}$, and $t \to_{\mathcal{R}} F(t)$ otherwise [7, Definition 4.9.1].
(ii) *A proper sub-redex* of a redex t is a redex contained in t and distinct from t itself.
(iii) *An innermost redex* is a redex that has no proper sub-redex.
(iv) *An innermost strategy* reduces an innermost redex in each rewrite step.
(v) *The leftmost-innermost strategy* reduces the leftmost innermost redex.

5.2. The leftmost-innermost strategy

Let $(\mathit{Type}, \Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Sclass})$ be a second-order TES. Suppose that it satisfies the following conditions:

(i) $\mathit{Ectx} = \mathit{Ctx}$.
(ii) Every critical pair of $\mathcal{E}$ is trivial.
(iii) Every metavariable in rules in $\mathcal{E}$ is a value-metavariable.
(iv) $\mathit{Val} = \mathsf{NF}(\to_{\mathcal{E}})$.

Then the evaluation $\to_{\mathcal{E}}$ always takes an innermost rewriting strategy. In this setting, a valid substitution assigns a normal form to each variable in a rule.

The leftmost-innermost strategy can be realised by defining evaluation contexts like

$$E ::= \Box \mid f(E, t_2, \ldots, t_n) \mid f(V, E, t_2, \ldots, t_n) \mid f(V, \ldots, V, E) \mid \cdots$$

Remark 5.1. The TES $\mathcal{E}$ consisting of the following rules shows necessity of the triviality conditions on critical pairs of $\mathcal{E}$.

$$f(c) \to 0 \tag{1}$$
$$f(g(x)) \to 1 \tag{2}$$
$$g(a) \to c \tag{3}$$

Consider the innermost rewrite sequence

$$f(\underline{g(a)}) \to_{\mathcal{E}} \underline{f(c)} \to_{\mathcal{E}} 0$$

which rewrites the underlined innermost redexes. As a TES, we obtain the first rewrite step by an inference that instantiates the rule (3) and puts it to the context $f(\Box)$. However, we can also obtain an outermost rewrite as an instance of the rule (2):

$$(\text{RuleSub})\frac{f(g(x)) \to 1 \in \mathcal{E} \quad \mathsf{valid}\ [x \mapsto a]}{f(g(a)) \to_{\mathcal{E}} 1}$$

Restrictions on context and substitution are not sufficient to prohibit this, because there is a non-trivial critical pair between (2) and (3). We therefore need the condition to exclude non-trivial critical pairs.

5.3. Innermost strategy in rewrite systems vs call-by-value in TES

In term rewriting literature, it is often explained that the innermost rewriting realises the call-by-value evaluation. Strictly speaking, however, they are different. The innermost rewriting is a strategy specified by innermost redexes merely, and the call-by-value evaluation is a calling mechanism in programming languages, where the notion of values is also important.

To concretely show the difference, we consider a TRS $(\Sigma, \mathcal{R})$ defined by $\Sigma = \{0, s, +, g, h\}$, where g, h are 0-ary function symbols, and

$$\mathcal{R} = \{0 + y \to y,\ s(x) + y \to s(x + y),\ g \to h\}.$$

Then $0 + g \to_{\mathcal{R}} 0 + h \to_{\mathcal{R}} h$ (normal form) is an innermost rewrite sequence.

We now consider a corresponding second-order TES $\mathcal{E}_{\mathrm{cbv}}$ that adopts the call-by-value evaluation. We take the same signature Σ with

$$\textbf{Values } V ::= 0 \mid s(V) \qquad \textbf{Evaluation contexts } E ::= \Box \mid E + t \mid V + E$$

and a set $\mathcal{E}_{\mathrm{cbv}} \triangleq \mathcal{R}$ of evaluation rules, but now we take the variables x, y as value metavariables to satisfy the condition (iii) in Section 5.2. Then we have $0 + g \to_{\mathcal{E}} 0 + h$, which is a normal form. The first $+$-rule is not applicable to $0 + h$ because the y in the $+$-rule is a value variable and h is not a value. Innermost rewriting strategies cannot simulate this behaviour. Therefore, the innermost rewriting and call-by-value are different.

Remark 5.2. Note that call-by-value evaluation is *not* a rewriting strategy in the formal sense of Definition 5.1 (i) because the strategy function F must return the next rewritten term whenever the given term is not a normal form of non-strategic rewriting. In the above case, $0 + h$ is not a normal form of $\mathcal{R}$, but the call-by-value evaluation does not provide the next rewritten term of it. Therefore, the call-by-value evaluation should be considered as a *restriction*, rather than a strategy.

5.4. Context-sensitive rewriting

In the first-order setting, another established way to specify a rewriting strategy (or more precisely, a restriction, cf. Remark 5.2) is *context-sensitive rewriting* [25,26]. We first recall its definition.

Definition 5.2 (Context-sensitive rewriting)**.** A *context-sensitive term rewriting system* (*CS-TRS* for short) $(\Sigma, \mathcal{R}, \mu)$ is given by:

- a signature Σ
- a set $\mathcal{R}$ of *rewrite rules* $(l \Rightarrow r) \in \mathcal{R}$ such that l is not a variable and $FV(l) \supseteq FV(r)$
- a *replacement map* μ that assigns each $(f : n) \in \Sigma$ a subset $\mu(f) \subseteq [n]$.

A replacement map μ, which is the core of context-sensitive rewriting, specifies where rewriting can happen. Specifically, it induces a set $\mathcal{P}os^{\mu}(t)$ of positions of a term t as follows.

$$\mathcal{P}os^{\mu}(x) = \{\epsilon\}, \quad \mathcal{P}os^{\mu}(f(t_1, \dots, t_n)) = \{\epsilon\} \cup \bigcup_{i \in \mu(f)} i.\mathcal{P}os^{\mu}(t_i).$$

A CS-TRS $(\Sigma, \mathcal{R}, \mu)$ induces the following rewrite relation $\hookrightarrow_{\mathcal{R},\mu}$.

$$\frac{\mathsf{subst}\,\theta \quad (l \Rightarrow r) \in \mathcal{R} \quad p \in \mathcal{P}os^{\mu}(t) \quad t|_p = l\theta \quad u = t[r\theta]_p}{t \hookrightarrow_{\mathcal{R},\mu} u}$$

A rewrite $l \Rightarrow r$ can only happen when the position p is an *active* position, i.e. p satisfies $p \in \mathcal{P}os^{\mu}(t)$.

Although context-sensitive rewriting can restrict where rewriting can happen, it does not inherently specify in which order rewriting can happen. Indeed, any CS-TRS can be translated into a potentially nondeterministic TES.

Definition 5.3 (Nondeterministic construction)**.** Given a CS-TRS $(\Sigma, \mathcal{R}, \mu)$, we define a first-order TES $(\Sigma, \mathcal{E}, Ectx, Val)$ as follows.

- $\mathcal{E} = \{l \to r \mid (l \Rightarrow r) \in \mathcal{R}\}$
- Let $\Sigma_{active} = \{f \in \Sigma \mid \exists (l \to r) \in \mathcal{E}.\, \exists t_1, \dots, t_n \in \mathrm{T}_{\Sigma}\mathrm{V}.\, l = f(t_1, \dots, t_n)\}$, and $\Sigma_{passive} = \Sigma \backslash \Sigma_{active}$. The sets $Ectx$ and Val are inductively defined as below.

$$\frac{}{\Box \in Ectx} \qquad \frac{(f : n) \in \Sigma \quad n > 0 \quad i \in \mu(f) \quad E \in Ectx \quad \{t_j \in \mathrm{T}_{\Sigma}\mathrm{V}\}_{j \in [n] \backslash \{i\}}}{f(t_1, \dots, t_{i-1}, E, t_{i+1}, \dots, t_n) \in Ectx}$$

$$\frac{(f : n) \in \Sigma_{passive} \quad \{t_j \in Val\}_{j \in \mu(f)} \quad \{t_j \in \mathrm{T}_{\Sigma}\mathrm{V}\}_{j \in [n] \backslash \mu(f)}}{f(t_1, \dots, t_n) \in Val}$$

We call this construction *nondeterministic* to emphasise that the resulting evaluation relation $\to_{\mathcal{E}}$ is potentially nondeterministic.

Proposition 5.1. *Let $(\Sigma, \mathcal{R}, \mu)$ be a CS-TRS and $(\Sigma, \mathcal{E}, Ectx, Val)$ be the TES obtained from the CS-TRS by the nondeterministic construction. We have:*

$$t \hookrightarrow_{\mathcal{R},\mu} u \iff t \to_{\mathcal{E}} u$$

Proof. [$\Leftarrow$]: There exist $(l \to r) \in \mathcal{E}$, $E \in Ectx$ and $\mathsf{subst}\,\theta$ such that $t = E[l\theta]$ and $u = E[r\theta]$. Let p be the position of $\Box$ in E. By construction of $Ectx$, we have $p \in \mathcal{P}os^{\mu}(E[l\theta])$.

[$\Rightarrow$]: There exist $(l \to r) \in \mathcal{E}$, $\mathsf{subst}\,\theta$ and $p \in \mathcal{P}os^{\mu}(t)$ such that $t|_p = l\theta$ and $u = t[r\theta]_p$. We can prove by straightforward induction on p that we have $t[\Box]_p \in Ectx$. $\square$

In summary, the operational behaviour induced by a context-sensitive term rewriting system can be represented within Term Evaluation Systems by allowing nondeterminism. Conversely, there exist deterministic TESs whose behaviour cannot be captured by context-sensitive restrictions.

For example, a replacement map $\mu(\mathtt{if}) = \{1\}$ specifies that only the first argument (i.e. the guard t of $\mathtt{if}(t, s_1, s_2)$) can be rewritten. The nondeterministic construction encodes this into evaluation contexts as $E ::= \Box \mid \mathtt{if}(E, s_1, s_2)$.

Another example is $\mu(+) = \{1, 2\}$ that specifies both of the two arguments of summation $+$ can be rewritten. This is encoded as

$$E ::= \Box \mid E + t \mid t + E.$$

This specification of evaluation contexts induces a *nondeterministic* evaluation relation $\to_{\mathcal{E}}$; both $(1 + 2) + (3 + 4) \to_{\mathcal{E}} 3 + (3 + 4)$ and $(1 + 2) + (3 + 4) \to_{\mathcal{E}} (1 + 2) + 7$ are possible. Note the difference with a left-to-right *deterministic* specification of evaluation contexts

$$E ::= \Box \mid E + t \mid v + E$$

where v represents a value. With this specification, only $(1 + 2) + (3 + 4) \to_{\mathcal{E}} 3 + (3 + 4)$ is possible, and $(1 + 2) + (3 + 4) \to_{\mathcal{E}} (1 + 2) + 7$ is impossible because $1 + 2$ is not a value and $(1 + 2) + \Box$ is not a valid evaluation context.

This example shows a difference between context-sensitive rewriting and TES. Context-sensitive rewriting does not inherently specify an evaluation order on function arguments, such as left-to-right evaluation, whereas TES can explicitly define an evaluation order.

Another example showing a difference is in the comparison with the call-by-value evaluation in Section 5.3. Context-sensitive rewriting does not inherently simulate the behaviour of TES $\mathcal{E}_{\mathrm{cbv}}$.

Another point we highlight is that existing formulations of context-sensitive rewriting are typically presented in untyped, first-order settings. In contrast, the second-order TERS framework developed in this paper incorporates *types* and *higher-order* functions. This design choice allows us to model the operational semantics of calculi for both lazy and strict higher-order functional languages, as illustrated by Examples 3.2, 3.3–3.5, and 4.1.

6. Related work

Having clarified the role of Term Evaluation Systems with respect to existing rewriting strategies, we now discuss related work that is directly concerned with contextual improvement and its use in the validation of program transformations.

6.1. Evaluation from the term-rewriting perspective

Unlike general term rewriting (i.e., refinement), evaluation that uses Felleisen's evaluation contexts has received little attention in the rewriting literature. As an exception, Faggian et al. [4,27] studied evaluation for specific simplified computational lambda-calculi including λ_{ml*}. They proved that refinement implies observational equivalence, crucially using the fact that refinement is confluent in these calculi. In contrast, we study evaluation for general TERSs. We identify sufficient conditions (e.g. local coherence) for contextual improvement, not relying on confluence of refinement.

6.2. Proof methodologies for improvement

There is rich literature on methodologies for proving observational equivalence [28–31]. Some methodologies have been applied to effect handlers [32,33]. We provide a novel term-rewriting-theoretic methodology centred around local coherence and critical pair analysis.

Our methodology is partly automatable, thanks to the fact that critical pair analysis for second-order computation systems can be automated [14,15]. Our prototype analyser based on this technology can automatically check the joinability of the critical pairs in the examples. There are few works on automating observational equivalence proofs for functional programs. Known examples, including the tool SyTeCi [34], are based on or inspired by algorithmic game semantics [35].

This work is focused on contextual improvement, a quantitative variant of observational equivalence. There is relatively limited literature on proof methodologies for contextual improvement. A coinductive approach based on applicative bisimulation has been used for space improvement [36] and time improvement [37]. This line of work, however, does not come with any form of automation.

6.3. Improvement theory for call-by-need

A well-established line of work studies program transformations for call-by-need using *improvement* preorders, most notably the operational theory of Moran and Sands [2]. Their framework refines contextual approximation for a specific lazy core language by observing not only termination but also the *machine-step cost* of evaluation, using Sestoft's abstract machine as the operational basis. Improvement ($M \sqsubseteq N$) means that, in every program context, $C[M]$ terminates with no greater step-cost than $C[N]$. Their results include a context lemma and an inequational "tick algebra" that subsumes the call-by-need calculi of Ariola et al. [38,39], and powerful induction principles for reasoning about recursive programs (e.g. fixpoint fusion) [1,40].

Our work is complementary but different in scope. Whereas Moran-Sands-style improvement is tied to a particular call-by-need language and to a concrete machine-step cost model, we work with a *general* typed second-order term rewriting framework that can express the operational semantics of a wide range of calculi, not restricted to lazy evaluation.

Instead of building a cost model externally, we count the *reduction steps* internal to a TERS, and characterise contextual improvement through the syntactic properties of a pair (E, R) of evaluation and refinement rules. Our results give generic, syntax-directed criteria — based on critical pairs and local coherence — under which a refinement system R is a contextual improvement of E, regardless of the evaluation strategy embodied by the underlying TERS. In this sense, our approach is strictly more general than call-by-need specific improvement theories, and may be combined with a concrete cost model in future work.

7. Conclusion and future work

We developed the rewriting-theoretic methodology for proving contextual improvement, both in the first-order setting and in the simply-typed second-order setting. We identified local coherence as a sufficient condition for contextual improvement, and showed that local coherence can be established by critical pair analysis. We demonstrated the proof methodology with calculi with effects and sharing.

The formal development relies on the frameworks of TESs and TERSs that we introduced. TERSs explicitly distinguish evaluation and refinement, and enable rewriting-theoretic reasoning about their interaction. We are interested in extending such rewriting-theoretic reasoning; for example to check if a TERS is deterministic, and if refinement implies observational equivalence instead of contextual improvement.

CRediT authorship contribution statement

Koko Muroya: Writing – review & editing, Writing – original draft, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization; **Makoto Hamana:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Koko Muroya reports financial support was provided by JSPS. Makoto Hamana reports financial support was provided by JSPS. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by JSPS, KAKENHI Project No. 20H04164, and No. 22K17850, Japan.

References

[1] D. Sands, Total correctness by local improvement in the transformation of functional programs, ACM Trans. Program. Lang. Syst. 18 (2) (1996) 175–234. https://doi.org/10.1145/227699.227716
[2] A. Moran, D. Sands, Improvement in a lazy context: an operational theory for call-by-need, in: POPL 1999, ACM, 1999, pp. 43–56. https://doi.org/10.1145/292540.292547
[3] A. Sabry, P. Wadler, A reflection on call-by-value, in: R. Harper, R.L. Wexelblat (Eds.), Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24–26, 1996, ACM, 1996, pp. 13–24. https://doi.org/10.1145/232627.232631
[4] C. Faggian, G. Guerrieri, R. Treglia, Evaluation in the computational calculus is non-confluent, in: 10th International Workshop of Confluence, IWC 2021, 2021, pp. 31–36. http://www.lix.polytechnique.fr/iwc2021/papers/IWC_2021_paper_6.pdf.
[5] Y. Toyama, Commutativity of Term Rewriting Systems, North-Holland, 1988, pp. 393–407.
[6] K. Muroya, M. Hamana, Term evaluation systems with refinements: first-order, second-order, and contextual improvement, in: J. Gibbons, D. Miller (Eds.), Functional and Logic Programming - 17th International Symposium, FLOPS 2024, Kumamoto, Japan, May 15–17, 2024, Proceedings, 14659 of Lecture Notes in Computer Science, Springer, 2024, pp. 31–61. https://doi.org/10.1007/978-981-97-2300-3_3
[7] Y. Terese, Term Rewriting Systems, 55 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2003.
[8] M. Hamana, Equivalence of the quotient term model and the least complete herbrand model for a functional-logic language, J. Funct. Log. Program. 1997 (1) (1997). http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/1997/A97-01/A97-01.html
[9] G.P. Huet, J. Hullot, Proofs by induction in equational theories with constructors, J. Comput. Syst. Sci. 25 (2) (1982) 239–266. https://doi.org/10.1016/0022-0000(82)90006-X
[10] H. Zantema, J. Endrullis, Proving equality of streams automatically, in: M. Schmidt-Schauß (Ed.), Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, Novi Sad, Serbia, May 30, - June 1, 2011, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, pp. 393–408. https://doi.org/10.4230/LIPICS.RTA.2011.393
[11] M. Hamana, Free Σ-monoids: a higher-order syntax with metavariables, in: W. Chin (Ed.), Programming Languages and Systems: Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4–6, 2004. Proceedings, 3302 of Lecture Notes in Computer Science, Springer, 2004, pp. 348–363. https://doi.org/10.1007/978-3-540-30477-7_23
[12] M.P. Fiore, Second-order and dependently-sorted abstract syntax, in: Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24–27 June 2008, Pittsburgh, PA, USA, IEEE Computer Society, 2008, pp. 57–68. https://doi.org/10.1109/LICS.2008.38
[13] M.P. Fiore, O. Mahmoud, Second-order algebraic theories - (Extended abstract), in: P. Hlinený, A. Kucera (Eds.), Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23–27, 2010. Proceedings, Lecture Notes in Computer Science, Springer, 2010, pp. 368–380. https://doi.org/10.1007/978-3-642-15155-2_33
[14] M. Hamana, How to prove decidability of equational theories with second-order computation analyser SOL, J. Funct. Program. 29 (2019) e20. https://doi.org/10.1017/S0956796819000157
[15] M. Hamana, T. Abe, K. Kikuchi, Polymorphic computation systems: theory and practice of confluence with call-by-value, Sci. Comput. Program. 187 (2020) 102322. https://doi.org/10.1016/J.SCICO.2019.102322
[16] F. Blanqui, Termination and confluence of higher-order rewrite systems, in: L. Bachmair (Ed.), Rewriting Techniques and Applications, 11th International Conference, RTA 2000, Norwich, UK, July 10–12, 2000, Proceedings, Lecture Notes in Computer Science, Springer, 2000, pp. 47–61. https://doi.org/10.1007/10721975_4
[17] S. Staton, Instances of computational effects: an algebraic perspective, in: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25–28, 2013, IEEE Computer Society, 2013, p. 519. https://doi.org/10.1109/LICS.2013.58
[18] P. Aczel, A General Church-Rosser Theorem, Technical Report, University of Manchester, 1978.
[19] J.W. Klop, Combinatory Reduction Systems, Ph.D. thesis, CWI, Amsterdam, 1980. volume 127 of Mathematical Centre Tracts.
[20] D. Miller, A logic programming language with lambda-abstraction, function variables, and simple unification, J. Log. Comput. 1 (4) (1991) 497–536. https://doi.org/10.1093/logcom/1.4.497
[21] G.D. Plotkin, Call-by-name, call-by-value and the lambda-calculus, Theor. Comp. Sci. 1 (2) (1975) 125–259. https://doi.org/10.1016/0304-3975(75)90017-1
[22] M. Pretnar, An introduction to algebraic effects and handlers. Invited tutorial paper, in: D.R. Ghica (Ed.), The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, the Netherlands, June 22–25, 2015, 319 of Electronic Notes in Theoretical Computer Science, Elsevier, 2015, pp. 19–35. https://doi.org/10.1016/J.ENTCS.2015.12.003
[23] R. Mayr, T. Nipkow, Higher-order rewrite systems and their confluence, Theor. Comput. Sci. 192 (1) (1998) 3–29. https://doi.org/10.1016/S0304-3975(97)00143-6
[24] J. Maraist, M. Odersky, D.N. Turner, P. Wadler, Call-by-name, call-by-value, call-by-need and the linear lambda calculus, Theor. Comp. Sci. 228 (1-2) (1999) 175–210. https://doi.org/10.1016/S0304-3975(98)00358-2
[25] S. Lucas, Context-sensitive computations in functional and functional logic programs, J. Funct. Log. Program. 1998 (1) (1998). http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/1998/A98-01/A98-01.html.
[26] S. Lucas, Context-sensitive rewriting, ACM Comput. Surv. 53 (4) (2021) 78:1–78:36. https://doi.org/10.1145/3397677
[27] C. Faggian, G. Guerrieri, U. de'Liguoro, R. Treglia, On reduction and normalization in the computational core, Math. Struct. Comput. Sci. 32 (7) (2022) 934–981. https://doi.org/10.1017/S0960129522000433
[28] S. Abramsky, The Lazy Lambda-Calculus, Addison Wesley, 1990, pp. 65–117.
[29] V. Koutavas, P. Levy, E. Sumii, From applicative to environmental bisimulation, Electr. Not. Theor. Comput. Sci. 276 (2011) 215–235. https://doi.org/10.1016/j.entcs.2011.09.023
[30] G.D. Plotkin, Lambda-definability and logical relations, 1973. Memorandum SAI-RM-4.
[31] R. Statman, Logical relations and the typed lambda-calculus, Inf. Control 65 (2/3) (1985) 85–97. https://doi.org/10.1016/S0019-9958(85)80001-2
[32] D. Biernacki, M. Piróg, P. Polesiuk, F. Sieczkowski, Handle with care: relational interpretation of algebraic effects and handlers, Proc. ACM Program. Lang. 2 (POPL) (2018) 8:1–8:30. https://doi.org/10.1145/3158096

[33] D. Biernacki, S. Lenglet, P. Polesiuk, A complete normal-form bisimilarity for algebraic effects and handlers, in: Z.M. Ariola (Ed.), 5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference), 167 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 7:1–7:22. https://doi.org/10.4230/LIPICS.FSCD.2020.7
[34] G. Jaber, SyTeCi: automating contextual equivalence for higher-order programs with references, Proc. ACM Program. Lang. 4 (POPL) (2020) 59:1–59:28. https://doi.org/10.1145/3371127
[35] S. Abramsky, Algorithmic game semantics: a tutorial introduction, in: NATO Advanced Study Institute 2001, 2001, pp. 21–47.
[36] E. Sumii, A bisimulation-like proof method for contextual properties in untyped lambda-calculus with references and deallocation, Theor. Comput. Sci. 411 (51-52) (2010) 4358–4378. https://doi.org/10.1016/J.TCS.2010.09.009
[37] U.D. Lago, F. Gavazzo, Effectful normal form bisimulation, in: L. Caires (Ed.), Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, 11423 of Lecture Notes in Computer Science, Springer, 2019, pp. 263–292. https://doi.org/10.1007/978-3-030-17184-1_10
[38] Z.M. Ariola, M. Felleisen, J. Maraist, M. Odersky, P. Wadler, The call-by-need lambda calculus, in: R.K. Cytron, P. Lee (Eds.), Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23–25, 1995, ACM Press, 1995, pp. 233–246. https://doi.org/10.1145/199448.199507
[39] J. Maraist, M. Odersky, P. Wadler, The call-by-need lambda calculus, J. Funct. Program. 8 (3) (1998) 275–317. https://doi.org/10.1017/S0956796898003037
[40] D. Sands, Operational theories of improvement in functional languages (Extended abstract), in: R. Heldal, C.K. Holst, P. Wadler (Eds.), Functional Programming, Glasgow 1991, Proceedings of the 1991 Glasgow Workshop on Functional Programming, Portree, Isle of Skye, UK, 12–14 August 1991, Workshops in Computing, Springer, 1991, pp. 298–311. https://doi.org/10.1007/978-1-4471-3196-0_24