Typed Term Evaluation Systems with Refinements for Contextual Improvement

Koko Muroya

National Institute of Informatics, Japan

Makoto Hamana

Kyusyu Institute of Technology, Japan

Abstract

For a programming language, there are two kinds of term rewriting: run-time rewriting ("evaluation") and compile-time rewriting ("refinement"). Whereas refinement resembles general term rewriting, evaluation is commonly constrained by Felleisen's evaluation contexts. While evaluation specifies a programming language, refinement models optimisation and should be validated with respect to evaluation. Such validation can be given by Sands' notion of contextual improvement. We formulate evaluation in a term-rewriting-theoretic manner for the first time, and introduce Term Evaluation and Refinement Systems (TERS). We then identify sufficient conditions for contextual improvement, and provide critical pair analysis for local coherence that is the key sufficient condition. We demonstrate our methodology of proving contextual improvement through examination of various calculi and functional programs, including a computational lambda-calculus, its extension with effect handlers, and the call-by-need lambda-calculus.

Keywords: term rewriting, evaluation contexts, critical pair analysis

1. Introduction

Term rewriting is a general model of computation. The ecosystem of a functional programming language utilizes two types of term rewriting: run-time rewriting, which we shall refer to as *evaluation*, and compile-time rewriting, referred to as *refinement*. Run-time evaluation specifies operational semantics of the language. It can only happen in a particular order, usually deterministically. On the other hand, compile-time refinement models optimisation. It can happen anywhere, nondeterministically. The difference between evaluation and refinement, as kinds of term rewriting, can be summarized in terms of *contexts*:

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

Evaluation $\to_{\mathcal{E}}$ uses a rewrite rule $l \to r$ inside a Felleisen's *evaluation context* [1, 2] $E \in \mathit{Ectx}$ only; this is a new kind of restriction from the rewriting theoretic point of view. In contrast, refinement $\Rightarrow_{\mathcal{R}}$ uses a rewrite rule $l \Rightarrow r$ inside an *arbitrary* context $C \in \mathit{Ctx}$; this resembles general term rewriting.

We analyse the roles of term rewriting in programming languages in this manner and divide them into *evaluation* and *refinement* for formalisation. This constitutes a novel theory that is more suitable as a semantics of programming

Email addresses: kmuroya@nii.ac.jp (Koko Muroya), hamana@csn.kyutech.ac.jp (Makoto Hamana)

languages. Evaluation *specifies* (the behavior of) a programming language as operational semantics. Evaluation is not merely a deterministic restriction of refinement. Refinement which models optimisation should be *validated* with respect to evaluation. Indeed, compiler optimisation is intended to preserve evaluation results and improve time efficiency of evaluation. Such preservation and improvement deserve formal validation.

Such validation can be provided as *observational equivalence* [3], and its quantitative variant, *contextual improvement* [4]. Observational equivalence $t \cong u$ asserts that two terms t and u cannot be distinguished by any context C; formally, if $C[t]$ terminates, $C[u]$ terminates with the same evaluation result, and vice versa. Contextual improvement additionally asserts that $C[u]$ terminates with no more evaluation steps than $C[t]$. This is a suitable notion to validate refinement which models optimisation.

Whereas the theory of refinement, which resembles general term rewriting, has been deeply developed, evaluation seems to be a new kind of restricted rewriting and it lacks a general theory from the perspective of term rewriting. This prevents useful ideas and techniques of term rewriting from transferring from refinement to evaluation. In recent work [5] on a proof methodology of observational equivalence, it is informally observed that a rewriting technique can be useful for proving observational equivalence and contextual improvement. This methodology informally employs critical pair analysis, a fundamental technique in rewriting theory. The idea is that $t \cong u$ holds if replacing t with u (which means applying a refinement rule $t \Rightarrow u$) in any program does not conflict with any evaluation rule $l \to r$.

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 $\to$:

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

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

A simplified computational lambda-calculus λ_{ml*} [6] is a TERS. Its terms are either values v, v' or computations p, p', and its evaluation (which has been studied [7]) is specified using evaluation contexts E and two evaluation rules $\to$:

$$v, v' ::= x \mid \lambda x.p, \qquad p, p' ::= \texttt{return}(v) \mid \texttt{let}\ x = p\ \texttt{in}\ p' \mid v\ v',$$
$$E ::= \Box \mid \texttt{let}\ x = E\ \texttt{in}\ p,\ (\lambda x.p)\ v \to p[v/x],\ \texttt{let}\ x = \texttt{return}(v)\ \texttt{in}\ p \to p[v/x].$$

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

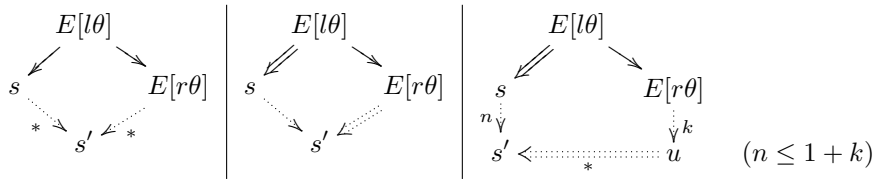


Figure 1: Joinability for confluence, commutation and local coherence

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

$$(\lambda x.p)\ v \Rightarrow p[v/x], \qquad \texttt{let}\ x = \texttt{return}(v)\ \texttt{in}\ p \Rightarrow p[v/x],$$
$$\lambda x.v\ x \Rightarrow v, \qquad \texttt{let}\ x = p\ \texttt{in}\ \texttt{return}(x) \Rightarrow p,$$
$$\texttt{let}\ x_1 = (\texttt{let}\ x_2 = p_2\ \texttt{in}\ p_1)\ \texttt{in}\ p_3 \Rightarrow \texttt{let}\ x_2 = p_2\ \texttt{in let}\ x_1 = p_1\ \texttt{in}\ p_3.$$

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*} which 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 evaluation of $C[t]$ terminates, then 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 analyse how each evaluation rule $l \to r$ can *conflict with* each refinement rule $l' \Rightarrow r'$. This is what exactly critical pair analysis is targeted at.

Critical pair analysis is usually for proving 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.

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 $1 + n$ 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 $E[t] \Rightarrow_{\mathcal{R}} s$. This modification should be "mild", and more precisely, refinement should not turn an evaluation context into a non-evaluation context (see Def. 2.15 ((ii))).

Note that local coherence can be seen as a generalisation of *commutation* [8]; 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

This paper aims at formalising this connection between observational equivalence proofs and critical pair analysis.

The key concepts of our development are evaluation contexts, ***values*** and local coherence. Evaluation contexts are treated axiomatically. Values specify *successful* results of evaluation; not all normal forms of evaluation are deemed successful. Such distinction of values has been studied in second-order rewriting [9]. Finally, local coherence is a notion from the rewriting literature; it is namely a sufficient condition for confluence in equational rewriting [10, 11]. We exploit the notion for TERS instead of equational rewriting. Note that TERS is *not* equational rewriting. Refinement is compile-time rewriting, and we do not evaluate modulo refinement.

This paper is the fully reworked and extended version of the conference paper [12]. The paper additionally includes new examples related to functional programming, detailed proofs, and comparison with existing rewriting strategies (Sec. 5).

More precisely, the contributions of this paper are summarized as follows.

1. We introduce a novel frameworks of *term evaluation systems (TES)*, and its combination with refinement, dubbed *term evaluation and refinement systems (TERS)*, in both first-order and second-order settings.

2. We identify sufficient conditions for contextual improvement that include a notion of *local coherence*.

3. We establish critical pair analysis for local coherence.

4. We demonstrate TERS with examples including a computational lambda-calculus and its extension with effect handlers.

1.3. Organisation

We first develop the theory of the first-order TERS in §2. This serves as the most direct system to illustrate our ideas and proof techniques for TERSs. As it is a simple system that can be regarded as a variant of Term Rewriting System (TRS) without higher-order functions, it should be accessible to many readers, particularly those familiar with first-order TRS.

Next, we develop the theory of the second-order TERS in §3. This extends the first-order TERS in two directions. The first is an extension to second-order syntax, incorporating variable binding and substitution into the syntax. The second is an extension to simple types, which was not done in the conference version [12]. As a result, the framework of TERS can faithfully represent both fundamental calculi and the operational semantics of typed higher-order functional programming languages.

The following two sections extend the conference version [12]. In §4 we provide a new example: the call-by-need lambda-calculus. In §5 we compare TE(R)S with known frameworks for specifying rewriting strategies. We namely discuss the *innermost* strategy, and Lucas' *context-sensitive* rewriting [13].

We conclude this paper with discussion on related work (§6) and future work (§7).

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, $[0] = \varnothing$, $[1] = \{1\}$, $[2] = \{1, 2\}$.

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 $\xrightarrow{*}$ denote the reflexive and transitive closure of $\to$. For any $k \in \mathbb{N}$, $\xrightarrow{k}$ 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 $f\colon n$. (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 T_Σ be the set of terms. A term is *closed* if it has no occurrence of variables. A term is *linear* if no variable occurs more than once.

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 $C \in \mathit{Ctx}$ 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 $\mathsf{subst}\,\theta$.

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 $(\Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Val})$ consisting of

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

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

Values specify which normal forms of $\to_{\mathcal{E}}$ are regarded as *successful* results. For example, in a TES for arithmetics, a term $x+y$ is a normal form but it is not deemed a successful result. The equivalence relation $=_{\mathit{Val}}$ specifies *observations*

of these results in terms of equivalence classes. For example, when the syntactic equality $\equiv$ is used, each value $v \in \mathit{Val}$ 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. It is also closed 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 $(\Sigma, \mathcal{E}, \mathcal{R}, \mathit{Ectx}, \mathit{Val})$ consisting of

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

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

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

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

Example 2.3 (List append) We define a TERS **Append** as follows.

Signature Σ	$[\,]\colon 0,\ (:)\colon 2,\ (+\!\!+)\colon 2$
Values Val	$V ::= [\,] \mid V : V$
	with $=_{\mathit{Val}}$ defined by
	$\dfrac{}{[\,] =_{\mathit{Val}} [\,]} \quad \dfrac{V_1 =_{\mathit{Val}} V_1' \quad V_2 =_{\mathit{Val}} V_2'}{V_1 : V_2 =_{\mathit{Val}} V_1' : V_2'}$
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 2.4 (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 $s_1 \overset{k}{\to}_{\mathcal{E}} u_1$, $s_2 \overset{n}{\to}_{\mathcal{E}} u_2$, $1 + k \geq n$ and $u_2 \overset{*}{\Rightarrow}_{\mathcal{R}} u_1$.

Definition 2.5 (Rewriting properties)

- A set $\mathcal{E}$ of evaluation rules is *deterministic* if every $\mathcal{E}$-peak is trivial.
- A TERS $(\mathcal{E}, \mathcal{R}, \mathit{Val})$ 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 formalize the key concept that relates evaluation with refinement in TERS. It is called ***(contextual) improvement*** [4]: 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 $=_{\mathit{Val}}$.

Definition 2.6 (Value-invariance, improvement)

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

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

- For a TERS $(\mathcal{E}, \mathcal{R}, \mathit{Val})$, $\mathcal{R}$ is ***improvement*** w.r.t. $\mathcal{E}$ if, for any $k \in \mathbb{N}$, $v \in \mathit{Val}$, $t \Rightarrow_{\mathcal{R}} s$ and any $C \in \mathit{Ctx}$ 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 $v' \in \mathit{Val}$ such that $v =_{\mathit{Val}} v'$ and $k \geq m$.

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.

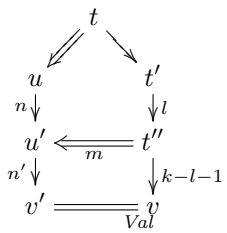
Theorem 2.7 (Sufficient condition for improvement: first-order version) Let $(\mathcal{E}, \mathcal{R}, \mathit{Val})$ be a TERS. If $\mathcal{E}$ is deterministic, and $(\mathcal{E}, \mathcal{R}, \mathit{Val})$ is value-invariant and locally coherent, then the set $\mathcal{R}$ of refinement rules is improvement w.r.t. the set $\mathcal{E}$ of evaluation rules.

Proof. Take arbitrary $k \in \mathbb{N}$ and $t, u \in \mathrm{T}_\Sigma$ such that $t \Rightarrow_{\mathcal{R}} u$ and $t \overset{k}{\to}_{\mathcal{E}} v \in \mathit{Val}$. We first prove that $t \Rightarrow_{\mathcal{R}} u$ and $t \overset{k}{\to}_{\mathcal{E}} v$ imply $u \overset{m}{\to}_{\mathcal{E}} v'$, $v =_{\mathit{Val}} v'$ and $k \geq m$, for any $k \in \mathbb{N}$, by induction on k.

Base case.. When $k = 0$, we have $t = v$. Because the TERS $(\mathcal{E}, \mathcal{R})$ is value-invariant, we have $u \in \mathit{Val}$ and $v =_{\mathit{Val}} u$. We can take $m = 0$.

Inductive case.. When $k > 0$, there exists $t' \in \mathrm{T}_\Sigma$ 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 $t'', u' \in \mathrm{T}_\Sigma$ and $l, m, n \in \mathbb{N}$ such that $t' \overset{l}{\to}_{\mathcal{E}} t''$, $u \overset{n}{\to}_{\mathcal{E}} u'$, $t'' \overset{m}{\Rightarrow}_{\mathcal{R}} u'$ and $1 + l \geq n$. 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 $t \to_{\mathcal{E}} t' \overset{l}{\to}_{\mathcal{E}} t'' \overset{k-l-1}{\to}_{\mathcal{E}} v$. We

prove that we have the following situation:



namely that there exist $n' \in \mathbb{N}$ and $v' \in \mathit{Val}$ such that $u' \xrightarrow{n'}_{\mathcal{E}} v'$ and $v =_{\mathit{Val}} v'$, by induction on $m \in \mathbb{N}$.

- *Base case.* When $m = 0$, $t'' = u'$. We can take $n' = k - l - 1$ and $v' = v$. Because $1 + l \geq n$, we have $k \geq n + n'$.

- *Inductive case.* When $m > 0$, we have $t'' \overset{m-1}{\Rightarrow}_{\mathcal{R}} u'' \Rightarrow_{\mathcal{R}} u'$ for some $u'' \in \mathrm{T}_\Sigma$. By I.H. on $m - 1$, we have $u'' \xrightarrow{n''}_{\mathcal{E}} v''$ such that $v'' =_{\mathit{Val}} v$ and $k - l - 1 \geq n''$. Furthermore, by I.H. of the outer induction on n'', we have $u' \xrightarrow{n'}_{\mathcal{E}} v'$ such that $v'' =_{\mathit{Val}} v'$ and $n'' \geq n'$. We finally have $k \geq n + n'$.

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

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

This theorem requires proving determinism, value-invariance and local coherence. To establish determinism, a well-known syntactic condition called *orthogonality* [14, Sec.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. In typical TERS, the equivalence relation $=_{\mathit{Val}}$ on values can be decided by simply comparing head symbols, making it straightforward to verify value-invariance. Finally, we will show that local coherence can be shown by critical pair analysis in Sec. 2.4.

Example 2.8 (Append, continued) The TERS **Append**'s set $\mathcal{E}$ of evaluation rules in Example 2.3 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 2.9 (Ones) Let **Ones** be the TERS defined as follows.

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

This defines lazy lists rather than the strict lists in Example 2.3. The refinement $\mathtt{ones} \Rightarrow \mathtt{ns}(\mathtt{1})$ appears to represent a denotational semantic equivalence on infinite lists, i.e., the denotation of both sides is the infinite list of 1s [15]. We can take such rules as refinements and show that they indeed constitute an improvement. This refinement rule and its intent are not an equivalence of infinite lists, but rather a behavioral equivalence of `ones` and `ns(1)`.

In this sense, the notion of improvement is broader than that of inductive theorems [16], which is an equality on finite terms.

Example 2.10 (Divergence) Let **Div** be the TERS defined as follows.

Signature Σ	$\Omega\colon 0,\ 0\colon 0$
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 [8]. Note that critical pairs are generated by two kinds of overlaps, due to asymmetry of $(\mathcal{R}, \mathcal{E})$-peaks.

Definition 2.11 (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 $\sigma = \theta\sigma'$.

Definition 2.12 (Overlaps) Let $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{R}, \mathcal{E}\}$. Given rules $(l_1 \to_{\triangleright 1} r_1) \in \mathcal{X}_1$, $(l_2 \to_{\triangleright 2} r_2) \in \mathcal{X}_2$ and a substitution θ, a quadruple $(l_1 \to_{\triangleright 1} r_1, l_2 \to_{\triangleright 2} r_2, p, \theta)$ is an $(\mathcal{X}_1, \mathcal{X}_2)$-*overlap* if it satisfies the following.

- The rules $l_1 \to_{\triangleright 1} r_1$ and $l_2 \to_{\triangleright 2} r_2$ do not have common variables.
- If $p = \varepsilon$, the rules $l_1 \to_{\triangleright 1} r_1$ and $l_2 \to_{\triangleright 2} r_2$ are not variants of each other.

- The sub-term $l_1|_p$ is not a variable, where p is a position of l_1.
- The substitution θ is a most general unifier between $l_1|_p$ and l_2.

Definition 2.13 (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 2.14 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 substitution, $(t_1\theta, s\theta, t_2\theta)$ 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 2.15 (Well-behaved TERS) A TERS $(\Sigma, \mathcal{E}, \mathcal{R}, \mathit{Ectx}, \mathit{Val})$ is *well-behaved* if it satisfies the following.

(i) For any $C_1, C_2 \in \mathit{Ctx}$, if $C_1[C_2] \in \mathit{Ectx}$ then $C_1, C_2 \in \mathit{Ectx}$.
(ii) For any $E \in \mathit{Ectx}$ and $C' \in \mathit{Ctx}$, if $E \Rightarrow_{\mathcal{R}} C'$ then $C' \in \mathit{Ectx}$.
(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 $l\{x \mapsto \Box\} \in \mathit{Ectx}$ then $r\{x \mapsto \Box\} \in \mathit{Ectx}$.

The condition (i) is typically satisfied by inductively-defined evaluation contexts. The condition (ii) was already discussed in Sec. 1.1. The other conditions are technical (see Remark 2.17 for some details), but these are straightforward to verify.

Theorem 2.16 (Critical pair theorem) A well-behaved TERS is locally coherent if and only if every critical pair is 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 $\mathsf{subst}\,\theta$ such that $s|_p = l\theta$, $t_1 = s[r\theta]_p$ and $s[\Box]_p \in \mathit{Ctx}$. 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 $\mathsf{subst}\,\theta'$ such that $(l\theta)|_{p'} = l'\theta'$, $t_2 = (l\theta)[r'\theta']_{p'}$ and $(l\theta)[\Box]_{p'} \in \mathit{Ectx}$. 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, there are two possibilities.

– If p' is a non-variable position of l, we have $(l|_{p'})\theta = (l\theta)|_{p'} = l'\theta'$. Since $FV(l) \cap FV(l') = \varnothing$, we can take $\theta \cup \theta'$ as a unifier between $l|_{p'}$ and l'. 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 and a variable y such that: $q_1 \in \mathcal{P}os(l)$, $l|_{q_1} = y$, $q_2 \in \mathcal{P}os(y\theta)$, and $p' = q_1 q_2$. Because of the condition ((i)) of Def. 2.15, $(l\theta)[\Box]_{p'} \in Ectx$ implies $l[\Box]_{q_1}, y\theta[\Box]_{q_2} \in Ectx$. The variable y must appear at most once in both l and r, due to the condition ((iii)a) of Def. 2.15. If y 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 y appears once in r, the rule $l' \to r'$ can be applied to t_1 thanks to the condition ((iii)b) of Def. 2.15, and the rule $l \Rightarrow r$ can be applied to t_2. 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 $s = f(\overline{x_1}.u_1, \ldots, \overline{x_i}.u_i, \ldots, \overline{x_k}.u_k)$, $l\theta = u_i|_{p_t}$. We have an $(\mathcal{R}, \mathcal{E})$-peak

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

By $s \to_{\mathcal{E}} t_2$, there exist $p' \in \mathcal{P}os(s)$, $(l' \to r') \in \mathcal{E}$ and $\mathsf{subst}\,\theta'$ 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-variable position of l', we have $(l'|_p)\theta' = (l'\theta')|_p = l\theta$. Since $FV(l) \cap FV(l') = \varnothing$, we can take $\theta \cup \theta'$ as a unifier between $l'|_p$ and l. 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 variable y such that: $q_1 \in \mathcal{P}os(l')$, $l'|_{q_1} = y$, $q_2 \in \mathcal{P}os(y\theta')$, and $p = q_1 q_2$. The variable y appears at most once in l', due to the condition ((iv)) of Def. 2.15. We can apply the rule $l' \to r'$ to t_1. We can also apply the rule $l \Rightarrow r$ to t_2, as many times as y 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 $f(\ldots, \overline{x_i}.u_i[\Box]_{p_t}, \ldots) \in Ectx$, we have $f(\ldots, \overline{x_i}.\Box, \ldots) \in Ectx$ too, thanks to the condition ((i)) of Def. 2.15. 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

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

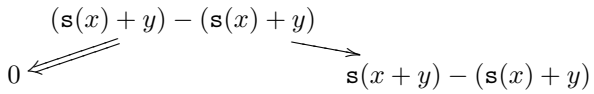
  is joinable (to $f(\ldots, \overline{x_{i'}}.u_{i'}[r'\theta']_{p'_t}, \ldots, \overline{x_i}.u_i[r\theta]_{p_t}, \ldots)$), thanks to the condition ((ii)) of Def. 2.15.

□

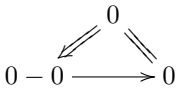
Remark 2.17 (On linearity conditions) For a TERS to be well-behaved, its evaluation rules must be left-linear, and its refinement rules must be linear (see Def. 2.15). Here we observe that relaxing these linearity conditions, with a reasonable set of evaluation contexts and values, leads to non-joinable $(\mathcal{R}, \mathcal{E})$-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
	$\dfrac{}{0 =_{Val} 0} \quad \dfrac{v =_{Val} v'}{\mathtt{s}(v) =_{Val} \mathtt{s}(v')}$
Evaluation contexts *Ectx*	$E ::= \Box \mid \mathtt{s}(E) \mid E + t \mid E - t \mid v - E$
Evaluation rules $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$0 + x \to x$	$x - x \Rightarrow 0$
$\mathtt{s}(x) + y \to \mathtt{s}(x + y)$	$0 \Rightarrow x - x$
$0 - x \to 0$	
$\mathtt{s}(x) - \mathtt{s}(y) \to x - y$	
$x \overset{?}{\equiv} x \to 0$	

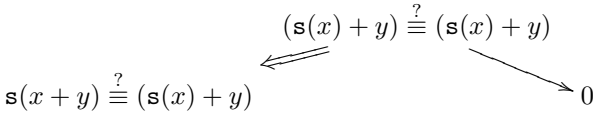
Firstly, the non-left-linear refinement rule $x - x \Rightarrow 0$ induces the following non-joinable $(\mathcal{R}, \mathcal{E})$-peak.



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



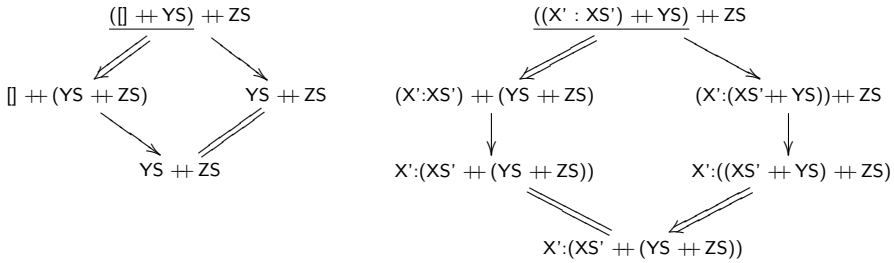
This $(\mathcal{R}, \mathcal{E})$-peak is not joinable with respect to our definition of joinability (see Def. 2.4). The bottom term $0 - 0$ must not take more evaluation steps than the top term 0. Finally, the non-left-linear evaluation rule $x \overset{?}{\equiv} x \to 0$ induces the following non-joinable $(\mathcal{R}, \mathcal{E})$-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.

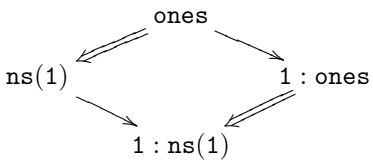
2.4.3. Examples

Example 2.18 (Append, continued) The TERS **Append** in Example 2.3 has the following two joinable critical pairs.



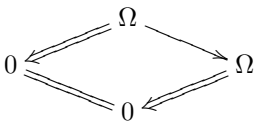
Because the TERS is well-behaved, it is locally coherent. By Thm. 2.7, we conclude that the refinement rule expressing the associativity of append is *improvement* w.r.t. its evaluation.

Example 2.19 (Ones, continued) The TERS **Ones** in Example 2.9 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 Thm. 2.7, the refinement rule $\texttt{ones} \Rightarrow \texttt{ns}(\texttt{1})$ is improvement.

Example 2.20 (Divergence, continued) The TERS **Div** in Example 2.10 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 Thm. 2.7, the refinement rule $\Omega \Rightarrow 0$ is 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 [17, 18], 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 [19] and second-order computation systems [20, 9]. The framework also involves simple-types with type constructors.

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:

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

Remark 3.1 Molecular types work as "base types" in ordinary type theories. But in our usage, we need "base types" which are constructed from "more basic" types. Hence we first assume atomic types as the most atomic ones, and then generate molecular types from them. Molecular types exactly correspond to base types in [21, 22].

Example 3.2 Assume atomic types Bool and Nat, and a unary type constructor List. Then the set $\mathcal{T}_0$ of all mol 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 mol type a, we have a mol type $\mathsf{List}(a)$.

3.2. Meta-terms

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

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

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

Example 3.3 The simply-typed λ-terms on the set Ty of simple types generated by a set BTy of base types are modeled in our setting as follows. Let $\mathcal{A} = \mathrm{BTy}$. We suppose a type constructor Arr to encode arrow types. The set of Typ of all simple types for the λ-calculus is the least set satisfying

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

The λ-terms are given by a signature

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

The β-reduction law is presented as

$$(beta) \quad M : a \to b,\ N : a \ \triangleright \ \vdash \mathsf{app}_{a,b}(\mathsf{lam}_{a,b}(x^a.\, M[x]),\ N) \ \Rightarrow \ M[N] : b$$

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 [9]). For example, simply writing function symbols lam and app, we mean $\mathsf{lam}_{a,b}$ and $\mathsf{app}_{a,b}$ in Σ_{stl} having appropriate mol type subscripts a, b.

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 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

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

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

Figure 2: Typing rules of meta-terms

with (meta-)terms $t_1, \ldots, t_n$. 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.

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 context Γ is a sequence of (variable:mol type)-pairs. A judgement is of the form

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

A meta-term t is *well-typed* by the typing rules Fig. 2. Note that a raw meta-term of the form $x.t$ does not have a type. But we will use raw terms $x.t$ in various places, such as in substitution below.

Let X be the countably infinite set of variables. 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 $\{A_b(\Gamma) \mid b \in \mathbb{T},\ \text{context } \Gamma\}$ 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 $\{A_b(\Gamma) \cup B_b(\Gamma) \mid b \in \mathbb{T},\ \text{context } \Gamma\}$. 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 $\{f_b(\Gamma) : A_b(\Gamma) \to B_b(\Gamma) \mid b \in \mathbb{T},\ \text{context } \Gamma\}$. Examples of contextual sets are the contextual set of meta-terms M_Σ and of terms T_Σ defined by

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

for 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.

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

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., $t|_p = M[t_1, \ldots, t_n]$. 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 $[M_1 \mapsto \overline{x_1}.s_1, \ldots, M_k \mapsto \overline{x_k}.s_k]$ such that: (i) $M_1, \ldots, M_k$ are distinct metavariables, and (ii) for some Θ, Γ and for each $i \in [k]$, $(M_i \colon |\overline{x_i}|) \in Z$ and $\Theta \rhd \Gamma, \overline{x_i} \vdash s_i$ hold. We call $M_1, \ldots, M_k$ the *domain* of θ and write $Dom(\theta)$. We call $\Theta \rhd \Gamma$ a *support* of θ, and write $\mathsf{subst}_{\Theta \rhd \Gamma}\, \theta$ when θ is a substitution with a support $\Theta \rhd \Gamma$. We sometimes simply write $\mathsf{subst}\, \theta$, omitting the support. Given a meta-term $\Theta, M_1 \colon |\overline{x_1}|, \ldots, M_k \colon |\overline{x_k}| \rhd \Gamma \vdash t$, a meta-term $t\theta$ is defined by

$$x\theta = x \qquad (f(\overline{x_1}.t_1, \ldots, \overline{x_l}.t_l))\theta = f(\overline{x_1}.t_1\theta, \ldots, \overline{x_l}.t_l\theta)$$

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

The meta-term $s_i\{(x_i)_1 \mapsto t_1\theta, \ldots, (x_i)_m \mapsto t_m\theta\}$ 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 $\overline{M} \cap \overline{N} = \varnothing$.

A meta-term is a *higher-order pattern* (*pattern* in short) if each occurrence of meta-application must be of the form $M[x_1, \ldots, x_m]$, where $x_1, \ldots, x_m$ are distinct bound variables.

Definition 3.4 (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.

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 $\mathsf{subst}\, \theta$.

3.4. Syntax classes

We introduce a notion of syntactic classification for terms, typically used for distinguishing values and non-values, following [9]. In *loc. cit.*, the call-by-value lambda-calculus (dubbed λ_{value}-calculus) involves the following two syntax classes of values and non-values.

$$\textbf{Values} \quad V ::= x \mid \lambda x.M \qquad\qquad \textbf{Non-values} \quad P ::= M\ N$$

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

Definition 3.5 (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

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

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

- 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 substitution: 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 3.7 For meta-terms $\Theta \triangleright\ \vdash\ \ell\ : \tau$ and $\Theta \triangleright\ \vdash\ r\ : \tau$, an *evaluation* (resp. a *refinement*) *rule* is of the form

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

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 Miller's higher-order pattern [23] to make unification decidable, which is important for computing rewrite steps and critical pairs. Second-order TES and TERS can now be defined, in an analogous way to the first-order setting.

$$\text{(RuleSub)}\ \frac{\Theta \triangleright \Gamma, \overline{x_i : \tau_i} \vdash\ s_i\ : \sigma_i \quad (1 \le i \le k) \qquad \mathsf{valid}\ [\overline{M \mapsto \overline{x}.s}] \qquad (M_1 : (\overline{\tau_1} \to \sigma_1), \ldots, M_k : (\overline{\tau_k} \to \sigma_k) \triangleright \vdash\ \ell \to r\ : \tau) \in \mathcal{E}}{\Theta \triangleright \Gamma \vdash\ \ell\,[\overline{M \mapsto \overline{x}.s}] \to_{\mathcal{E}} r\,[\overline{M \mapsto \overline{x}.s}]\ : \tau}$$

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

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

$$\text{(RuleSub)}\ \frac{\Theta \triangleright \Gamma, \overline{x_i : \tau_i} \vdash\ s_i\ : \sigma_i \quad (1 \le i \le k) \qquad \mathsf{valid}\ [\overline{M \mapsto \overline{x}.s}] \qquad (M_1 : (\overline{\tau_1} \to \sigma_1), \ldots, M_k : (\overline{\tau_k} \to \sigma_k) \triangleright \vdash\ \ell \Rightarrow r\ : \tau) \in \mathcal{R}}{\Theta \triangleright \Gamma \vdash\ \ell\,[\overline{M \mapsto \overline{x}.s}] \Rightarrow_{\mathcal{R}} r\,[\overline{M \mapsto \overline{x}.s}]\ : \tau}$$

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

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

Definition 3.8 (Second-order TES) A *second-order term evaluation system* is a tuple $(\mathit{Type}, \Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Sclass})$ consisting of

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

The ***evaluation relation*** $\to_{\mathcal{E}} \subseteq \mathrm{M}_\Sigma \times \mathrm{M}_\Sigma$ 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 3.9 (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

- a second-order TES $(\mathit{Type}, \Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Sclass})$, and
- a set $\mathcal{R}$ of *refinement rules* that satisfies the following.
- 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.
- 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 (see Def. 2.4), rewriting properties (Def. 2.5), and improvement (Def. 2.6) are inherited from the first-order case. Finally, the

first main theorem (Thm. 2.7) also holds in the second-order setting:

Theorem 3.10 (Sufficient condition for improvement: second-order version) 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 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 3.11 (The untyped call-by-value lambda-calculus [24]) 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 ι

Signature Σ

$\lambda\colon (\iota \to \iota) \to \iota,\ @\colon \iota, \iota \to \iota$

Syntax class *Sclass*

values $V ::= x \mid \lambda(x.M)$

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

Evaluation contexts *Ectx* $\quad E ::= \Box \mid E@M \mid V@E$

Evaluation rule $\mathcal{E}$

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

Refinement rules $\mathcal{R}$

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

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

We note that evaluation proceeds *from left to right* in the TERS **CBV**λ. For example, we have:

$$\begin{aligned}(\underline{\lambda(x.x)@\lambda(y.y)})@(\lambda(z.z)@\lambda(w.w)) &\to_{\mathcal{E}} \lambda(y.y)@(\underline{\lambda(z.z)@\lambda(w.w)}) \\ &\to_{\mathcal{E}} \underline{\lambda(y.y)@\lambda(w.w)} \\ &\to_{\mathcal{E}} \lambda(w.w),\end{aligned}$$

$$(\lambda(x.x)@\lambda(y.y))@(\underline{\lambda(z.z)@\lambda(w.w)}) \not\to_{\mathcal{E}} (\lambda(x.x)@\lambda(y.y))@\lambda(w.w).$$

The last evaluation is impossible, because $(\lambda(x.x)@\lambda(y.y))@\Box$ is not a valid evaluation context. We also emphasize 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 3.12 (A simplified computational lambda-calculus λ_{ml*} [6, 7]) A notion of evaluation for Sabry and Wadler's computational lambda-calculus λ_{ml*} [6] has been studied [7]. A TERS **Comp**λ_{ml*} of the computational lambda-

calculus is defined as follows:

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

Signature Σ

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

$\mathtt{let}_{a,b}\colon \mathsf{T}(a), (a \to \mathsf{T}(b)) \to \mathsf{T}(b),\ \mathtt{return}_a\colon a \to \mathsf{T}(a)$

Syntax classes *Sclass*

values $V, V' ::= x \mid \lambda(x.P)$

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

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 3.13 (Effect handlers [25]) A TERS **Hndl** is defined in Fig. 3.7, 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 [25] 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 Miller's higher-order pattern[2]. The refinement rules are numbered according to the original presentation [25, 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. The definition of critical pairs is again standard (cf. [26]), akin to commutation.

Definition 3.14 (Lifter) Given a sequence $\overline{x}$ of variables and a sequence $\overline{K}$ of meta-variables, an $\overline{x^a}$*-lifter of a pattern* t *away* $\overline{K}$ 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) $\overline{N} \cap \overline{K} = \varnothing$, and (ii) if $M_i\colon \overline{a_i} \to b_i$ then $N_i\colon \overline{a_i}, \overline{a} \to b_i$.

[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 [25, Fig. 7] is the only refinement rule whose lhs is not a Miller's higher-order pattern.

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

Signature Σ

$\mathtt{true}, \mathtt{false}\colon \mathsf{Bool},\ \mathtt{fun}_{a,b}\colon (a \to \mathsf{T}(b)) \to \mathsf{Arr}(a,b),\ (@_{a,b})\colon \mathsf{Arr}(a,b), a \to \mathsf{T}(b),$

$\mathtt{return}_a\colon a \to \mathsf{T}(a),\ \mathtt{op}_{1_{a,b,c}}\colon a, (b \to \mathsf{T}(c)) \to \mathsf{T}(c),\ \mathtt{op}_{0_{a,b,c}}\colon a, (b \to \mathsf{T}(c)) \to \mathsf{T}(c),$

$\mathtt{handler}_{1_{a,b,c}}\colon (a \to \mathsf{T}(b)), (a', (b' \to \mathsf{T}(c')) \to \mathsf{T}(c')) \to \mathsf{Hndl}(\mathsf{T}(a), \mathsf{T}(b)),$

$\mathtt{handler}_{0_{a,b}}\colon (a \to \mathsf{T}(b)) \to \mathsf{Hndl}(\mathsf{T}(a), \mathsf{T}(b)),\ \mathtt{do}_{a,b}\colon \mathsf{T}(a), (a \to \mathsf{T}(b)) \to \mathsf{T}(b),$

$\mathtt{if}_a\colon \mathsf{Bool}, \mathsf{T}(a), \mathsf{T}(a) \to \mathsf{T}(a),\ \mathtt{with\_handle}_{a,b}\colon \mathsf{Hndl}(\mathsf{T}(a), \mathsf{T}(b)), \mathsf{T}(a) \to \mathsf{T}(b)$

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

Evaluation contexts *Ectx* $E ::= \square \mid \mathtt{do}(E, x.P) \mid \mathtt{with\_handle}(H, E)$

Evaluation rules $\mathcal{E}$ where $i \in [2]$

$$\mathtt{do}(\mathtt{return}(V), x.P[x]) \to P[V] \tag{1}$$

$$\mathtt{do}(\mathtt{op}_i(V, y.P_1[y]), x.P_2[x]) \to \mathtt{op}_i(V, y.\mathtt{do}(P_1[y], x.P_2[x])) \tag{2}$$

$$\mathtt{if}(\mathtt{true}, P_1, P_2) \to P_1 \tag{3}$$

$$\mathtt{if}(\mathtt{false}, P_1, P_2) \to P_2 \tag{4}$$

$$\mathtt{fun}(x.P[x])@V \to P[V] \tag{5}$$

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 $\mathcal{R}$

$$\mathtt{do}(P, x.\mathtt{return}(x)) \Rightarrow P \tag{r3}$$

$$\mathtt{do}(\mathtt{do}(P_1, x_1.P_2[x_1]), x_2.P_3[x_2]) \Rightarrow \mathtt{do}(P_1, x_1.\mathtt{do}(P_2[x_1], x_2.P_3[x_2])) \tag{r4}$$

$$\mathtt{fun}(x.F@x) \Rightarrow F \tag{r9}$$

$$\mathtt{with\_handle}(\mathtt{handler}_0(x.P[x]), P') \Rightarrow \mathtt{do}(P', x.P[x]) \tag{r13}$$

Figure 5: The TERS **Hndl**

Definition 3.15 (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 3.16 (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 $FM(l_1)$.
- The substitution θ is a most general unifier between $l_1|_p$ and $l_2\sigma$.

Definition 3.17 (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 3.18 Compared to the first-order setting, the definition of critical pairs need to use *lifters* defined in Definition 3.14 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 [26]. The TERS is a variant of the call-by-name lambda-calculus, with its refinement rule being "incorrect" eta-rule.

Type signature ι

Signature Σ

$\lambda\colon (\iota \to \iota) \to \iota,\ @\colon \iota, \iota \to \iota,\ 0\colon \iota$

Syntax class *Sclass*

values $V ::= x$

with $=_{Val}$ defined by $\dfrac{}{x =_{Val} y}$

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

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 take a most general unifier between the lhs $\lambda(y.M[y])@N$ of the evaluation rule and a subterm $M'[x]@x$ 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 3.19 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 $FM(l_1)$. 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 $[(M\rho) \mapsto \overline{y}, \overline{x}.\ \theta_2(M[\overline{x}])]$. We take $\theta = \theta_1 \cup \theta_2'$. □

Lemma 3.20 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. □

To obtain the so-called critical pair theorem, TERS need to be well-behaved again. The following conditions are similar to the first-order case (see Def. 2.15), 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 3.21 (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 $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$,

(a) both l and r are linear w.r.t. non-value metavariables, and
(b) for every non-value metavariable $M : \overline{\sigma} \to \tau \in FM(l)$, if $l[M \mapsto \overline{x}.\Box] \in Ectx$ then $r[M \mapsto \overline{x}.\Box] \in Ectx$.

(iv) For any $(l \to_{\mathcal{E}} r) \in \mathcal{E}$, l is linear w.r.t. non-value metavariables.

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 3.22 (Critical pair theorem) A well-behaved TERS is locally coherent if and only if every critical pair is 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 $s[\Box]_p \in \mathit{Ctx}$. 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 $(l\theta)[\Box]_{p'} \in \mathit{Ectx}$. 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 $(l\theta)[\Box]_{p'} \in \mathit{Ectx}$ 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 Lem. 3.19, 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 Def. 3.21, $(l\theta)[\Box]_{p'} \in \mathit{Ectx}$ implies $l[\Box]_{q_1}, (N[\overline{y}])\theta[\Box]_{q_2} \in \mathit{Ectx}$. 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 ((iii)a) of Def. 3.21. 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 ((iii)b) of Def. 3.21, and the rule $l \Rightarrow r$ can be applied to t_2, thanks to the condition (3.9) of Def. 3.21. 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 $s = f(\overline{x_1}.u_1, \ldots, \overline{x_i}.u_i, \ldots, \overline{x_k}.u_k)$ or $s = M[u_1, \ldots, u_i, \ldots, u_k]$, such that $l\theta = u_i|_{p_t}$.

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

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

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 $s[\Box]_{p'} \in \mathit{Ectx}$. 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 Lem. 3.19, 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 ((iv)) of Def. 3.21. We can apply the rule $l' \to r'$ to t_1. The substitution θ' is valid, thanks to the condition (3.9) of Def. 3.21. 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 $f(\ldots, \overline{x_i}.u_i[\Box]_{p_t}, \ldots) \in \mathit{Ectx}$, we have $f(\ldots, \overline{x_i}.\Box, \ldots) \in \mathit{Ectx}$ too, thanks to the condition ((i)) of Def. 3.21. 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

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

    is joinable (to $f(\ldots, \overline{x_{i'}}.u_{i'}[r'\theta']_{p'_t}, \ldots, \overline{x_i}.u_i[r\theta]_{p_t}, \ldots)$), thanks to the condition ((ii)) of Def. 3.21.

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

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

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 $s[\Box]_{p'} \in \mathit{Ectx}$. We proceed by case analysis on $p' \in \mathcal{P}os(s)$.

- When $p' = \varepsilon$, $M[u_1, \ldots, u_k] = l'\theta'$. 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

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

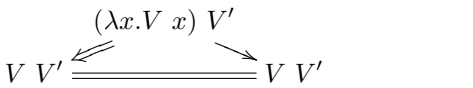
□

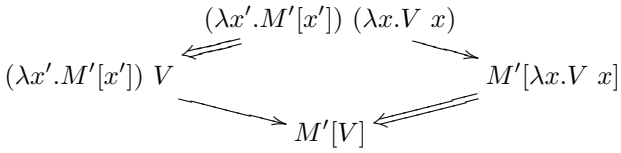
3.8.2. The three examples

Firstly, 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 $=_{\mathit{Val}}$ to the original abstraction. Therefore these TERSs are value-invariance.

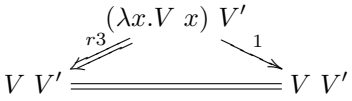
Secondly, the three TERSs are well-behaved. Most of the conditions are trivially satisfied, or easy to check. The condition (ii) can be checked by straightforward induction on $E \in \mathit{Ectx}$.

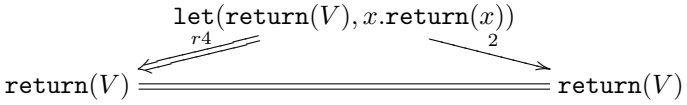
To establish local coherence, the next step is to check every critical pair is joinable. The TERS **CBV**λ has the following two joinable critical pairs, which are for the second refinement rule (the so-called eta-rule) and the sole evaluation rule.

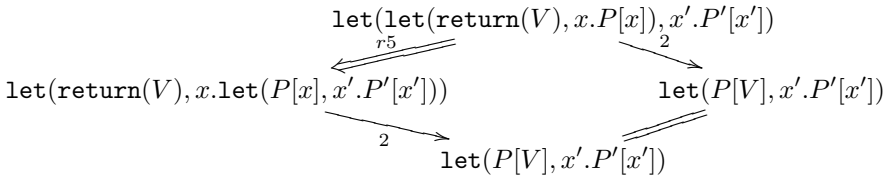




The TERS **Comp**λ_{ml*} has the following three critical pairs. In the following, arrows $\to$, $\Rightarrow$ are labelled by a number that indicates which evaluation/refinement rule is applied.

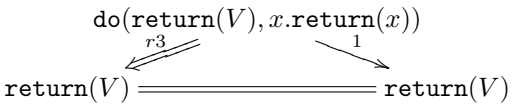


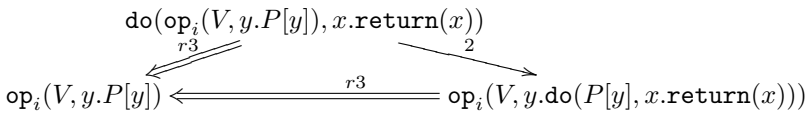


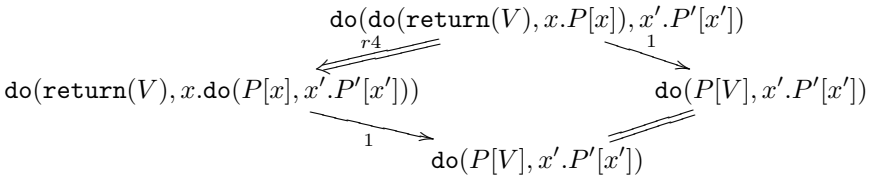


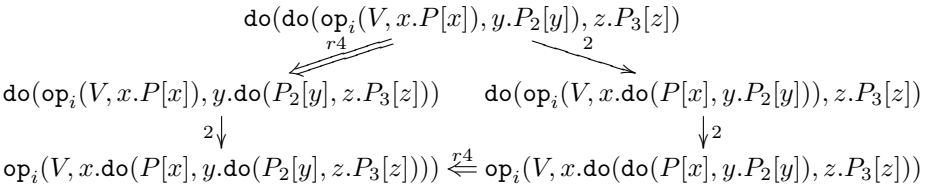
The TERS **Hndl** has the following ten joinable critical pairs; we merge some of them below using $\mathtt{op}_i$ ($i \in [2]$). In the following, arrows $\to$, $\Rightarrow$ are labelled by a number that indicates which evaluation/refinement rule is applied, and we set

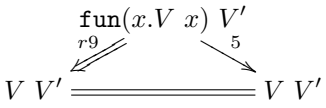
$h_0 \equiv \mathtt{handler}_0(x.P[x])$.

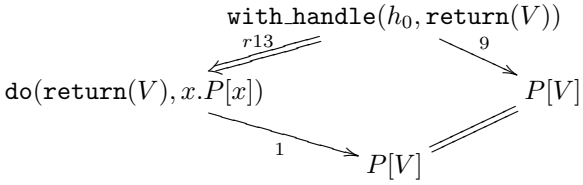


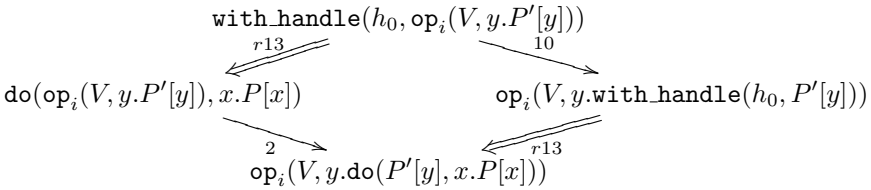












Finally, by Thm. 3.10 and Thm. 3.22, the refinement rules $\mathcal{R}$ of each of the three TERSs are *improvement* with respect to the evaluation rules $\mathcal{E}$.

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

Example 4.1 (Typed call-by-need lambda-calculus [27]) A TERS **CBNeedλ** 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.

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

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

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

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

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

Evaluation rules $\mathcal{E}$

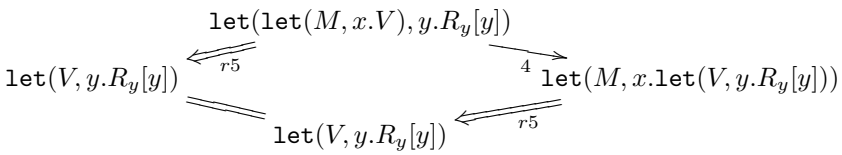
$$\lambda(x.M[x])@N \to \mathtt{let}(N, x.M[x]) \tag{1}$$
$$\mathtt{let}(A, x.R_x[x]) \to R_x[A] \tag{2}$$
$$\mathtt{let}(M, x.V[x])@N \to \mathtt{let}(M, x.V[x]@N) \tag{3}$$
$$\mathtt{let}(\mathtt{let}(M, x.V[x]), y.R_y[y]) \to \mathtt{let}(M, x.\mathtt{let}(V[x], y.R_y[y])) \tag{4}$$

Refinement rules $\mathcal{R}$

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

Figure 6: Call-by-need lambda-calculus

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



Therefore, by Thm. 3.10 and Thm. 3.22, the refinement rules $\mathcal{R}$ of the TERS **CBNeed**λ are *improvement* with respect to the evaluation rules $\mathcal{E}$.

5. Rewriting strategies and term evaluation

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 realizes 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 [14, Definition 4.9.1].
(ii) An innermost redex is a redex that has no proper subredex.
(iii) The innermost strategy reduces an innermost redex in each rewrite step.
(iv) The leftmost-innermost strategy reduces the leftmost-innermost redex.

5.1. The 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 the 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 realized by defining evaluation contexts like

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

Remark 5.2 The TES $\mathcal{E}$ consisting of the following rules shows why the condition (ii) is necessary.

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

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):

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

We want to prohibit this, but we cannot achieve it by imposing restrictions on context and substitution. This is because there is a non-trivial critical pair between (2) and (3), so the condition (ii) is necessary.

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

In term rewriting literature, it is often explained that the innermost rewriting realizes 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 $\Sigma = \{0, +, g, h\}$, 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 §5.1. 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. The innermost rewriting strategy cannot simulate this behavior. Therefore, the innermost strategy and call-by-value are different.

Remark 5.3 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, of rewriting as discussed in the case of context-sensitive rewriting [28, Remark 1].

5.3. Context-sensitive rewriting

In the first-order setting, another established way to specify a rewriting strategy (or more precisely, a restriction, cf. Remark 5.3) is Lucas' *context-sensitive rewriting* [13]. We first recall its definition.

Definition 5.4 (Context-sensitive rewriting [13]) A *context-sensitive term rewriting system* (*CS-TRS* in 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.

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

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

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

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

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

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

Proposition 5.6 Let $(\Sigma, \mathcal{R}, \mu)$ be a CS-TRS and $(\Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Val})$ 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 $(l \to r) \in \mathcal{E}$, $E \in \mathit{Ectx}$ 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 $t[\Box]_p \in \mathit{Ectx}$.

In summary, every CS-TRS can be simulated by a nondeterministic TES. However, a deterministic TES cannot, in general, be simulated by a CS-TRS.

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 §5.2. Context-sensitive rewriting does not inherently simulate the behavior of TES $\mathcal{E}_{\text{cbv}}$.

The following is taken from the literature on context-sensitive rewriting.

Example 5.7 (Nats [13, Ex. 8.19]) Let **Nats** be the TERS defined as follows.

Signature Σ	$\mathtt{nats}\colon 0,\ \mathtt{inc}\colon 1,\ \mathtt{hd}\colon 1,\ \mathtt{tl}\colon 1,\ (:)\colon 2,$ $\mathtt{s}\colon 1,\ 0\colon 0$
Values *Val*	$V ::= 0 \mid \mathtt{s}(V) \mid V : t$ with $=_{Val}$ defined by $\dfrac{}{0 =_{Val} 0} \quad \dfrac{V =_{Val} V'}{\mathtt{s}(V) =_{Val} \mathtt{s}(V')}$ $\dfrac{V =_{Val} V' \quad t, t' \in \mathrm{T}_\Sigma}{V : t =_{Val} V' : t'}$
Evaluation contexts *Ectx*	$E ::= \Box \mid \mathtt{hd}(E) \mid \mathtt{tl}(E) \mid \mathtt{inc}(E) \mid E : t \mid \mathtt{s}(E)$
Evaluation rules $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$\mathtt{nats} \to 0 : \mathtt{inc}(\mathtt{nats})$	$\mathtt{tl}(\mathtt{inc}(\mathtt{nats})) \Rightarrow \mathtt{inc}(\mathtt{tl}(\mathtt{nats}))$
$\mathtt{inc}(x : y) \to \mathtt{s}(x) : \mathtt{inc}(y)$	
$\mathtt{hd}(x : y) \to x$	
$\mathtt{tl}(x : y) \to y$	

This TERS **Nats** is deterministic, value-invariant and locally coherent. By Thm. 2.7, its refinement $\mathcal{R}$ is *improvement* w.r.t. its evaluation $\mathcal{E}$. In the proof of local coherence, we observe that the TERS **Nats** has one critical pair; it is joinable as in Fig. 7. The direction of refinement, which we reversed compared to the original [13], is crucial. Refinement must not increase the number of evaluation steps.

Another point we highlight is that the context-sensitive rewriting systems are currently formulated as untyped, first-order systems only. In contrast, the second-order TERS framework incorporates types and higher-order functions, making it suitable for more faithfully modelling the operational semantics for real-world lazy and strict higher-order typed functional programming languages, such as Haskell and ML, as demonstrated in Example 3.3, 3.11, 3.12, 3.13, and 4.1.

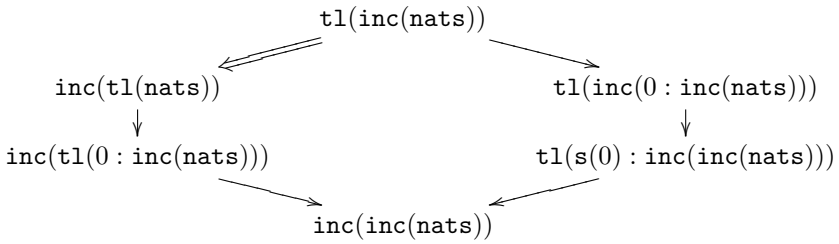


Figure 7: Joinability of the critical pair

6. Related work

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. [7, 29] 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 TERS. 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 [30, 31, 32, 33]. Some methodologies have been applied to effect handlers [34, 35]. 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 [20, 9]. Our prototype analyzer based on this technology could 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 [36], are based on or inspired by algorithmic game semantics [37].

This work is targeted at 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 [38] and time improvement [39]. This line of work, however, does not come with any form of automation.

7. Conclusion and future work

We formalised evaluation from the term-rewriting perspective, and introduced TERS in both first-order and second-order settings. To validate refinement (which models optimisation) with respect to evaluation, we employed the concept of contextual improvement, and identified sufficient conditions for it. The key condition is local coherence, for which we developed critical pair analysis. We demonstrated TERS with examples including λ_{ml*} and its extension with effect handlers.

This work contributes to bridging the gap between general term rewriting and evaluation, by introducing TERS. We are interested in bringing more term-rewriting techniques and insights to evaluation; for example to check if a TERS is deterministic, and if refinement implies observational equivalence instead of contextual improvement.

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

References

[1] M. Felleisen, The theory and practice of first-class prompts, in: J. Ferrante, P. Mager (Eds.), Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988, ACM Press, 1988, pp. 180–190. doi:10.1145/73560.73576.
URL https://doi.org/10.1145/73560.73576

[2] M. Felleisen, lambda-v-cs: An extended lambda-calculus for scheme, in: J. Chailloux (Ed.), Proceedings of the 1988 ACM Conference on LISP and Functional Programming, LFP 1988, Snowbird, Utah, USA, July 25-27, 1988, ACM, 1988, pp. 72–85. doi:10.1145/62678.62686.
URL https://doi.org/10.1145/62678.62686

[3] J. H. Morris Jr, Lambda-calculus models of programming languages., Ph.D. thesis, Massachusetts Institute of Technology (1969).

[4] D. Sands, Total correctness by local improvement in the transformation of functional programs, ACM Trans. Program. Lang. Syst. 18 (2) (1996) 175–234. doi:10.1145/227699.227716.

[5] K. Muroya, Hypernet semantics of programming languages, Ph.D. thesis, University of Birmingham (2020).
URL https://etheses.bham.ac.uk/id/eprint/10433/

[6] 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. doi:10.1145/232627.232631.
URL https://doi.org/10.1145/232627.232631

[7] 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.
URL http://www.lix.polytechnique.fr/iwc2021/papers/IWC_2021_paper_6.pdf

[8] Y. Toyama, Commutativity of term rewriting systems, North-Holland, 1988, pp. 393–407.

[9] M. Hamana, T. Abe, K. Kikuchi, Polymorphic computation systems: Theory and practice of confluenc
Sci. Comput. Program. 187 (2020) 102322. doi:10.1016/J.SCICO.2019.102322.
URL https://doi.org/10.1016/j.scico.2019.102322

[10] G. P. Huet, Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems, J. ACM 27 (4) (1980) 797–821. doi:10.1145/322217.322230.

[11] T. Aoto, Y. Toyama, A reduction-preserving completion for proving confluence of non-terminating term rewriting systems, Log. Methods Comput. Sci. 8 (1). doi:10.2168/LMCS-8(1:31)2012.

[12] K. Muroya, M. Hamana, Term evaluation systems with refinements: First-order, second-order, and co in: J. Gibbons, D. Miller (Eds.), Functional and Logic Programming - 17th International Symposium, FLOPS 2024, Kumamoto, Japan, May 15-17, 2024, Proceedings, Vol. 14659 of Lecture Notes in Computer Science, Springer, 2024, pp. 31–61. doi:10.1007/978-981-97-2300-3_3.
URL https://doi.org/10.1007/978-981-97-2300-3_3

[13] S. Lucas, Context-sensitive rewriting, ACM Comput. Surv. 53 (4) (2021) 78:1–78:36. doi:10.1145/3397677.
URL https://doi.org/10.1145/3397677

[14] Terese, Term Rewriting Systems, no. 55 in Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2003.

[15] M. Hamana, Equivalence of the quotient term model and the least complete herbrand model for a functional-logic language, Journal of Functional and Logic Programming 1997 (1).

[16] G. P. Huet, J. Hullot, Proofs by induction in equational theories with constructors, J. Comput. Syst. Sci. 25 (2) (1982) 239–266. doi:10.1016/0022-0000(82)90006-X.
URL https://doi.org/10.1016/0022-0000(82)90006-X

[17] 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, Vol. 3302 of Lecture Notes in Computer Science, Springer, 2004, pp. 348–363. doi:10.1007/978-3-540-30477-7_23.

[18] M. Fiore, Second-order and dependently-sorted abstract syntax, in: Proc. of LICS'08, 2008, pp. 57–68.

[19] M. Fiore, O. Mahmoud, Second-order algebraic theories, in: Proc. of MFCS'10, LNCS 6281, 2010, pp. 368–380.

[20] M. Hamana, How to prove decidability of equational theories with second-order computation analyser J. Funct. Program. 29 (2019) e20. doi:10.1017/S0956796819000157.
URL https://doi.org/10.1017/S0956796819000157

[21] F. Blanqui, Termination and confluence of higher-order rewrite systems, in: Rewriting Techniques and Application (RTA 2000), LNCS 1833, Springer, 2000, pp. 47–61.

[22] S. Staton, Instances of computational effects: An algebraic perspective, in: Proc. of LICS'13, 2013, p. 519.

[23] D. Miller, A logic programming language with lambda-abstraction, function variables, and simple uni J. Log. Comput. 1 (4) (1991) 497–536. doi:10.1093/logcom/1.4.497.
URL https://doi.org/10.1093/logcom/1.4.497

[24] G. D. Plotkin, Call-by-name, call-by-value and the lambda-calculus, Theor. Comp. Sci. 1 (2) (1975) 125–259. doi:10.1016/0304-3975(75)90017-1.

[25] 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, Vol. 319 of Electronic Notes in Theoretical Computer Science, Elsevier, 2015, pp. 19–35. doi:10.1016/J.ENTCS.2015.12.003.
URL https://doi.org/10.1016/j.entcs.2015.12.003

[26] R. Mayr, T. Nipkow, Higher-order rewrite systems and their confluence, Theor. Comput. Sci. 192 (1) (1998) 3–29. doi:10.1016/S0304-3975(97)00143-6.
URL https://doi.org/10.1016/S0304-3975(97)00143-6

[27] 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. doi:10.1016/S0304-3975(98)00358-2.

[28] S. Lucas, Context-sensitive rewriting strategies, Inf. Comput. 178 (1) (2002) 294–343. doi:10.1006/inco.2002.3176.
URL https://doi.org/10.1006/inco.2002.3176

[29] 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. doi:10.1017/S0960129522000433.
URL https://doi.org/10.1017/S0960129522000433

[30] S. Abramsky, The lazy lambda-calculus, Addison Wesley, 1990, pp. 65–117.

[31] V. Koutavas, P. Levy, E. Sumii, From applicative to environmental bisimulation, Elect. Notes in Theor. Comp. Sci. 276 (2011) 215–235. doi:10.1016/j.entcs.2011.09.023.

[32] G. D. Plotkin, Lambda-definability and logical relations, memorandum SAI-RM-4 (1973).

[33] R. Statman, Logical relations and the typed lambda-calculus, Information and Control 65 (2/3) (1985) 85–97. doi:10.1016/S0019-9958(85)80001-2.

[34] 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. doi:10.1145/3158096.
URL https://doi.org/10.1145/3158096

[35] 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), Vol. 167 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 7:1–7:22. doi:10.4230/LIPICS.FSCD.2020.7.
URL https://doi.org/10.4230/LIPIcs.FSCD.2020.7

[36] G. Jaber, Syteci: automating contextual equivalence for higher-order programs with references, Proc. ACM Program. Lang. 4 (POPL) (2020) 59:1–59:28. doi:10.1145/3371127.
URL https://doi.org/10.1145/3371127

[37] S. Abramsky, Algorithmic game semantics: a tutorial introduction, in: NATO Advanced Study Institute 2001, 2001, pp. 21–47.

[38] E. Sumii, A bisimulation-like proof method for contextual properties in untyped lambda-calculus with Theor. Comput. Sci. 411 (51-52) (2010) 4358–4378. doi:10.1016/J.TCS.2010.09.009.
URL https://doi.org/10.1016/j.tcs.2010.09.009

[39] 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, Vol. 11423 of Lecture Notes in Computer Science, Springer, 2019, pp. 263–292. doi:10.1007/978-3-030-17184-1_10.
URL https://doi.org/10.1007/978-3-030-17184-1_10