Term Evaluation Systems with Refinements: First-Order, Second-Order, and Contextual Improvement

Koko Muroya[1][0000−0003−0454−6900] and Makoto Hamana[2][0000−0002−3064−8225]

[1] RIMS, Kyoto University, Japan
kmuroya@kurims.kyoto-u.ac.jp
[2] Kyushu Institute of Technology, Japan
hamana@cs.gunma-u.ac.jp

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. As case studies, we prove contextual improvement for a computational lambda-calculus and its extension with effect handlers.

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 summarised in terms of *contexts*, cf. Fig. 1. Evaluation $\rightarrow_{\mathcal{E}}$ uses a rewrite rule $l \rightarrow r$ inside a Felleisen's *evaluation context* [9, 8] $E \in \mathit{Ectx}$ 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.

From the viewpoint of rewriting theory, the roles of evaluation and refinement are rather unusual. It is evaluation that *specifies* (the behaviour of) a programming language as operational semantics. Evaluation is not simply a deterministic restriction of refinement. Refinement which models optimisation should be *validated* with respect to

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

Fig. 1. Evaluation and refinement relations, where θ is a substitution

evaluation. Indeed, compiler optimisation is meant to preserve evaluation results and improve time efficiency of evaluation. This preservation and improvement deserve formal validation.

Such validation can be provided as *observational equivalence* [22], and its quantitative variant, *contextual improvement* [27]. 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 [23] 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 $l \to r$.

This paper aims at formalising this connection between observational equivalence proofs and critical pair analysis. In doing so, we introduce a new rewriting-theoretic formalisation of evaluation. Our contributions are:

- introducing a new formalisation 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,
- identifying sufficient conditions for contextual improvement that include a notion of *local coherence*,
- establishing critical pair analysis for local coherence, and
- demonstrating TERS with examples including a computational lambda-calculus and its extension with effect handlers.

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 [14]. Finally, local coherence is a notion from the rewriting literature; it is namely a sufficient condition for confluence in equational rewriting [15, 3]. We exploit the notion for TERS instead of equational rewriting[3].

1.1 Examples of TES and TERS

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 $\to$:

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

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*} [26] 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 $\to$:

$$\begin{gathered} v, v' ::= x \mid \lambda x.p, \qquad p, p' ::= \mathtt{return}(v) \mid \mathtt{let}\ x = p\ \mathtt{in}\ p' \mid v\ v', \\ E ::= \square \mid \mathtt{let}\ x = E\ \mathtt{in}\ p,\ (\lambda x.p)\ v \to p[v/x],\ \mathtt{let}\ x = \mathtt{return}(v)\ \mathtt{in}\ p \to p[v/x]. \end{gathered}$$

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

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

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

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

[3] TERS is *not* equational rewriting. Refinement is compile-time rewriting, and we do not evaluate modulo refinement.

$$\begin{tikzcd}
 & E[l\theta] \arrow[dl] \arrow[dr] & \\
s \arrow[dr, dotted, "*"'] & & E[r\theta] \arrow[dl, dotted, "*"] \\
 & s' &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & E[l\theta] \arrow[dl, Rightarrow] \arrow[dr] & \\
s \arrow[dr, dotted] & & E[r\theta] \arrow[dl, dotted, Rightarrow] \\
 & s' &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & E[l\theta] \arrow[dl, Rightarrow] \arrow[dr] & \\
s \arrow[d, dotted, "l"'] & & E[r\theta] \arrow[d, dotted, "k"] \\
s' & & u \arrow[ll, dotted, Rightarrow, "*"]
\end{tikzcd}
\quad (1 + k \geq l)$$

Fig. 2. Joinability for confluence, commutation and local coherence

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. 2 (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. 2 (right). To ensure improvement, our notion of local coherence asserts that the joining part satisfies the inequality $1+k \geq l$ about the number of evaluation steps.

To prove the joinability for local coherence, we need to be careful with evaluation contexts. We need to show that the $1 + k$ evaluation steps $E[l\theta] \to_{\mathcal{E}} E[r\theta] \overset{k}{\to}_{\mathcal{E}} u$ can be *simulated* by the l evaluation steps $s \overset{l}{\to}_{\mathcal{E}} s'$. 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. 12 (2)).

Note that local coherence can be seen as a generalisation of *commutation* [30]; see Fig. 2 (middle). Commutation is the case where $k = 0$, $l = 1$, and allowing only one step of refinement $\Rightarrow_{\mathcal{R}}$ instead of $\overset{*}{\Rightarrow}_{\mathcal{R}}$.

2 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).

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

3 First-order Term Evaluation and Refinement Systems

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 *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 $Pos(t)$ 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* $\square$. Let Ctx be the set of contexts. Let $C[t]$ denote the term where the hole $\square$ of $C \in Ctx$ is replaced by t. A set $\mathcal{C}$ of contexts is *closed under substitutions* if $C \in \mathcal{C}$ implies $C\theta \in \mathcal{C}$ for any $\mathsf{subst}\,\theta$, and *closed under composition* if $C, C' \in \mathcal{C}$ implies $C[C'] \in \mathcal{C}$. The set $\mathcal{C}$ is *inductive* if any $C \in \mathcal{C}$ is $\square$ 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 $f(t_1\theta, \ldots, t_{i-1}\theta, \square, t_{i+1}\theta, \ldots, t_n\theta) \in \mathcal{C}$ 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 1 (TES). A *term evaluation system* is a tuple $(\Sigma, \mathcal{E}, Ectx, Val)$ consisting of

- a signature Σ,
- a set $\mathcal{E}$ of *evaluation rules*, where $(l \to r) \in \mathcal{E}$ with $l, r \in \mathrm{T}_\Sigma$, such that (i) every free variable occuring in r also occurs in l and (ii) l is not a variable,
- a set $Ectx \subseteq Ctx$ of *evaluation contexts* that is closed under substitutions, closed under composition and inductive, and
- a set $Val \subseteq \mathsf{NF}(\to_{\mathcal{E}})$ of *values* that satisfies (i) $v \in Val$ implies $v\theta \in Val$ for any $\mathsf{subst}\,\theta$, and (ii) it comes with an equivalence relation $=_{Val} \subseteq Val \times Val$, where:
- the *evaluation relation* $\to_{\mathcal{E}} \subseteq \mathrm{T}_\Sigma \times \mathrm{T}_\Sigma$ is defined in Fig. 1.

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 $=_{Val}$ specifies *observations* of these results in terms of equivalence classes. For example, when the syntactic equality $\equiv$ is used, each value $v \in Val$ becomes a distinct observation. On the other hand, when the total relation $\top$ is used, values are all identified; this means that successful termination is the only possible observation.

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

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

Definition 2 (TERS). A *term evaluation and refinement system* is a tuple $(\Sigma, \mathcal{E}, \mathcal{R}, Ectx, Val)$ consisting of

- A TES $(\Sigma, \mathcal{E}, Ectx, Val)$, and
- a set $\mathcal{R}$ of *refinement rules* where $(l \Rightarrow r) \in \mathcal{R}$ with $l, r \in \mathrm{T}_\Sigma$, such that (i) every free variable occuring in r also occurs in l and (ii) l is not a variable.
- The *refinement relation* $\Rightarrow_\mathcal{R} \subseteq \mathrm{T}_\Sigma \times \mathrm{T}_\Sigma$ is defined by the inference rule in Fig. 1.

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

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

Example 3 (Nats [20, Ex. 8.19]). Let **Nats** be the TERS defined as follows.

Signature Σ	$\mathtt{nats}\colon 0,\ \mathtt{inc}\colon 1,\ \mathtt{hd}\colon 1,\ \mathtt{tl}\colon 1,\ \text{‘:’}\colon 2,\ \mathtt{s}\colon 1,\ 0\colon 0$
Values Val	$V ::= 0 \mid \mathtt{s}(V)$
Evaluation contexts $Ectx$	$E ::= \square \mid \mathtt{hd}(E) \mid \mathtt{tl}(E) \mid \mathtt{inc}(E)$
Evaluation rules $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$\mathtt{nats} \to 0 : \mathtt{inc}(\mathtt{nats})$	$\mathtt{tl}(\mathtt{inc}(\mathtt{nats})) \Rightarrow \mathtt{inc}(\mathtt{tl}(\mathtt{nats}))$
$\mathtt{inc}(x : y) \to \mathtt{s}(x) : \mathtt{inc}(y)$	
$\mathtt{hd}(x : y) \to x$	
$\mathtt{tl}(x : y) \to y$	

We define $=_{Val}$ by the syntactic equality $\equiv$, to allow each value to be observed separately. The refinement rule is reversed, compared to the original one [20, Ex. 8.19], so that it induces improvement; see Sec. 4 for details.

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 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 $k, l \in \mathbb{N}$ and u_1, u_2 such that $s_1 \overset{k}{\to}_\mathcal{E} u_1$, $s_2 \overset{l}{\to}_\mathcal{E} u_2$, $1 + l \geq k$ and $u_2 \overset{*}{\Rightarrow}_\mathcal{R} u_1$.

Definition 5 (Rewriting properties).
- A TES $(\mathcal{E}, Val)$ is *deterministic* if every $\mathcal{E}$-peak (s_1, t, s_2) is trivial.
- A TERS $(\mathcal{E}, \mathcal{R}, Val)$ is *locally coherent* if every $(\mathcal{R}, \mathcal{E})$-peak is joinable up to $\mathcal{R}$.

We say a TERS $(\mathcal{E}, \mathcal{R}, Val)$ is deterministic if $(\mathcal{E}, Val)$ is deterministic. We also simply say an $(\mathcal{R}, \mathcal{E})$-peak is *joinable*, omitting “up to $\mathcal{R}$.”

We formalise the key concept that relates evaluation with refinement in TERS. It is the so-called *(contextual) improvement* [27]: that is, any refinement $t \Rightarrow_\mathcal{R} s$ cannot be distinguished by evaluation $\to_\mathcal{E}$ inside any contexts, and the refinement cannot increase the number of evaluation steps that are needed for termination. Observation is made according to the set Val of values and its associated equivalence relation $=_{Val}$.

Definition 6 (Value-invariance, improvement).

- A TERS $(\mathcal{E}, \mathcal{R}, \mathit{Val})$ is *value-invariant* if, for any $v \in \mathit{Val}$ and $s \in \mathrm{T}_\Sigma$, $v \Rightarrow_{\mathcal{R}} s$ implies $s \in \mathit{Val}$ and $v =_{\mathit{Val}} s$.
- 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, $C[t] \overset{k}{\rightarrow}_{\mathcal{E}} v$ implies $C[s] \overset{m}{\rightarrow}_{\mathcal{E}} v'$ for some $m \in \mathbb{N}$ and $v' \in \mathit{Val}$ such that $v =_{\mathit{Val}} v'$ and $k \geq m$.

Improvement is notoriously difficult to directly prove, because of the universal quantification over all contexts. The following is our first main theorem providing a rewriting-theoretic sufficient condition for improvement, for deterministic TERS.

Theorem 7 (Sufficient condition for improvement: first-order version). *If a TERS $(\mathcal{E}, \mathcal{R}, \mathit{Val})$ is deterministic, 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.*

This theorem requires to prove determinism, value-invariance and local coherence. When a TERS is orthogonal, determinism boils down 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. This makes it easy to verify value-invariance. We will show that local coherence can be shown by critical pair analysis in Sec. 4.

4 Critical pair analysis for local coherence

Critical pairs. The definition of critical pairs is standard; it resembles the definition of critical pairs for commutation [30]. Note that critical pairs are generated by two kinds of overlaps, due to asymmetry of $(\mathcal{R}, \mathcal{E})$-peaks.

Definition 8 (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 9 (Overlaps). Let $\mathcal{X}_1, \mathcal{X}_2 \in \{\mathcal{R}, \mathcal{E}\}$. Given rules $(l_1 \multimap_1 r_1) \in \mathcal{X}_1$, $(l_2 \multimap_2 r_2) \in \mathcal{X}_2$ and a substitution θ, a quadruple $(l_1 \multimap_1 r_1, l_2 \multimap_2 r_2, p, \theta)$ is an $(\mathcal{X}_1, \mathcal{X}_2)$-*overlap* if it satisfies the following.

- The rules $l_1 \multimap_1 r_1$ and $l_2 \multimap_2 r_2$ do not have common variables.
- If $p = \varepsilon$, the rules $l_1 \multimap_1 r_1$ and $l_2 \multimap_2 r_2$ 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 10 (Critical pairs).

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

Lemma 11. 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.

$$\begin{tikzcd}
 & \mathtt{tl}(\mathtt{inc}(\mathtt{nats})) \arrow[dl, Rightarrow] \arrow[dr] & \\
\mathtt{inc}(\mathtt{tl}(\mathtt{nats})) \arrow[d] & & \mathtt{tl}(\mathtt{inc}(0 : \mathtt{inc}(\mathtt{nats}))) \arrow[d] \\
\mathtt{inc}(\mathtt{tl}(0 : \mathtt{inc}(\mathtt{nats}))) \arrow[dr] & & \mathtt{tl}(\mathtt{s}(0) : \mathtt{inc}(\mathtt{inc}(\mathtt{nats}))) \arrow[dl] \\
 & \mathtt{inc}(\mathtt{inc}(\mathtt{nats})) &
\end{tikzcd}$$

Fig. 3. Joinability of the critical pair

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

Definition 12 (Well-behaved TERS). A TERS $(\Sigma, \mathcal{E}, \mathcal{R}, \mathit{Ectx}, \mathit{Val})$ is *well-behaved* if it satisfies the following.

1. For any $C_1, C_2 \in \mathit{Ctx}$, if $C_1[C_2] \in \mathit{Ectx}$ then $C_1, C_2 \in \mathit{Ectx}$.
2. For any $E \in \mathit{Ectx}$ and $C' \in \mathit{Ctx}$, if $E \Rightarrow_{\mathcal{R}} C'$ then $C' \in \mathit{Ectx}$.
3. For any $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$ and any variable x, the following holds.
   (a) The variable x appears at most once in l, and at most once in r.
   (b) Let p be the position of x in l. For the position q of x in r, if $l[\square]_p \in \mathit{Ectx}$ then $r[\square]_q \in \mathit{Ectx}$.
4. For any $(l \rightarrow_{\mathcal{E}} r) \in \mathcal{E}$, any variable x appears at most once in l.

The condition (1) is usually satisfied by inductively-defined evaluation contexts. The condition (2) was already discussed in Sec. 1.1. The other conditions are rather technical (see Sec. A.3 for some details), but these are easy to verify.

Theorem 13 (Critical pair theorem). A well-behaved TERS is locally coherent if and only if every critical pair is joinable.

The example. The TERS **Nats** in Example 3 is deterministic, value-invariant and locally coherent (Prop. 29 in Appendix). By Thm. 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. 3. The direction of refinement, which we reversed compared to the original [20], is crucial. Refinement must not increase the number of evaluation steps.

5 Second-Order Term Evaluation and Refinement Systems

Next we extend our framework to the second-order setting. By second-order we mean to use second-order abstract syntax [12, 10], 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 [11] and second-order computation systems [13, 14].

Meta-terms. In addition to the countably infinite set X of variables, let Z be a countably infinite set of *metavariables*. Each element $M \in Z$ comes with an *arity* $m \in \mathbb{N}$; we write $M \colon m$. Let $\equiv$ denote the syntactic equality on metavariables. Let Σ be a *signature*, each of whose element $f \in \Sigma$ comes with a sequence $\langle n_1, \ldots, n_l \rangle$ of natural numbers called a *binding arity*; we write $f \colon \langle n_1 \ldots, n_l \rangle$.

Let M_Σ denote the set of *meta-terms* defined using the signature Σ, the set X of variables and the set Z of metavariables. A judgement $\Theta \rhd \Gamma \vdash t$ consists of a set Θ of metavariables, a set Γ of variables, and a meta-term t. A *well-formed* meta-term is a meta-term t such that a judgement $\Theta \rhd \Gamma \vdash t$ is derivable by formation rules below for some Θ, Γ. We assume that meta-terms are well-formed.

$$\frac{x \in \Gamma}{\Theta \rhd \Gamma \vdash x} \qquad \frac{(M \colon m) \in \Theta \quad \{\Theta \rhd \Gamma \vdash t_i\}_{i \in [m]}}{\Theta \rhd \Gamma \vdash M[t_1, \ldots, t_m]}$$

$$\frac{(f \colon \langle n_1, \ldots, n_l \rangle) \in \Sigma \quad \{\Theta \rhd \Gamma, \overline{x_i} \vdash t_i\}_{i \in [l]} \quad \{|\overline{x_i}| = n_i\}_{i \in [l]}}{\Theta \rhd \Gamma \vdash f(\overline{x_1}.t_1, \ldots, \overline{x_l}.t_l)}$$

In a meta-term $f(\overline{x_1}.t_1, \ldots, \overline{x_l}.t_l)$, each $\overline{x_i}.t_i$ introduces bound variables $\overline{x_i}$. We assume the α-equivalence for bound variables. A meta-term $M[t_1, \ldots, t_m]$ is called a *meta-application*. The arguments $t_1, \ldots, t_m$ can be seen as explicit substitution for variables; when a meta-term s substitutes the metavariable M, free variables of s gets substituted by $t_1, \ldots, t_m$. A *term* is a meta-term that contains no meta-application. 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. Let $Pos(t)$ be the set of all positions in a meta-term t. 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 $\Theta \rhd \Gamma$ 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.

A meta-term is a *higher-order pattern* if each occurrence of meta-application has the form $M[x_1, \ldots, x_m]$ such that $x_1, \ldots, x_m$ are distinct bound variables.

A *context* C is a meta-term that involves exactly one *hole* $\square$. A context is *flat* if any prefix of the position of the hole is not a metavariable position; e.g. $f(x.\square)$ is a flat context, but $M[\square]$ and $M[f(x.\square)]$ are not flat contexts. Let *Ctx* be the set of contexts. Let $C[t]$ denote the term where the hole $\square$ 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 $\square$ 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 $f(t_1\theta, \ldots, t_{i-1}\theta, \square, t_{i+1}\theta, \ldots, t_n\theta) \in \mathcal{C}$ for any $\mathsf{subst}\,\theta$.

Syntax classes. We introduce a notion of syntactic classification for terms, typically used for distingushing values and non-values, following [14]. 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. In general, we define a set *Sclass* of names for *syntax classes*. Each syntax class is associated with a BNF grammar to define a set of meta-terms. Every metavariable is either associated with a syntax class and called "<syntax class name> metavariable", or not associated and called *general metavariable*. Moreover, we assume a default syntax class **value** to denote values.

For example, in the case of λ_{value}-calculus, we define $\mathit{Sclass} = \{\textbf{values}\ V ::= x \mid \lambda x.M,\ \textbf{non-values}\ P ::= M\,N\}$. The metavariable V is called a value metavariable, and M is a general metavariable.

Substitutions must also be consistent with syntax classes. A substitution θ is *valid* if for each assignment $(M \mapsto \overline{x}.s) \in \theta$, M and s's syntax class are the same, or if M is a general metavariable then s can be arbitrary. We write $\mathsf{valid}\,\theta$ when θ is a valid substitution.

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.

Evaluation and refinement. Second-order TES and TERS can now be defined, in an analogous way to the first-order setting.

Definition 14 (Second-order TES). A *second-order term evaluation system* is a tuple $(\Sigma, \mathcal{E}, \mathit{Ectx}, \mathit{Sclass})$ consisting of

- a signature Σ,
- a set $\mathcal{E}$ of *evaluation rules*, where $(l \to r) \in \mathcal{E}$ with $l, r \in \mathrm{M}_\Sigma$, such that (i) every free metavariable occuring in r also occurs in l and (ii) l is not a variable nor a metavariable,
- 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 includes a *value* class (V, Val) that satisfies (i) $\mathit{Val} \subseteq \mathsf{NF}(\to_{\mathcal{E}})$, (ii) $v \in \mathit{Val}$ implies $v\theta \in \mathit{Val}$ for any $\mathsf{valid}\,\theta$, and (iii) it comes with an equivalence relation $=_{\mathit{Val}} \subseteq \mathit{Val} \times \mathit{Val}$ where:

– the *evaluation relation* $\rightarrow_{\mathcal{E}} \subseteq \mathrm{M}_\Sigma \times \mathrm{M}_\Sigma$ is defined in Fig. 1 where θ is valid.

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

Definition 15 (Second-order TERS). A *second-order term evaluation and refinement system* is a tuple $(\Sigma, \mathcal{E}, \mathcal{R}, Ectx, Sclass)$ consisting of

– A second-order TES $(\Sigma, \mathcal{E}, Ectx, Sclass)$, and
– a set $\mathcal{R}$ of *refinement rules* where $(l \Rightarrow r) \in \mathcal{R}$ with $l, r \in \mathrm{M}_\Sigma$, such that (i) every free metavariable occuring in r also occurs in l and (ii) l is not a variable nor a metavariable.
– The *refinement relation* $\Rightarrow_{\mathcal{R}} \subseteq \mathrm{M}_\Sigma \times \mathrm{M}_\Sigma$ is defined in Fig.1 where θ is valid.

The refinement relation $\Rightarrow_{\mathcal{R}}$ is closed under arbitrary contexts in *Ctx*. It is not closed under substitutions per se, but it satisfies the following: $t \Rightarrow_{\mathcal{R}} u$ implies $t\theta \overset{*}{\Rightarrow}_{\mathcal{R}} u\theta$ for any valid θ.

We also assume that the lhs of every rule is a Miller's higher-order pattern [21] to make unification decidable.

Joinability and improvement. The definitions of peaks, joinability (see Def. 4), rewriting properties (Def. 5), and improvement (Def. 6) are inherited from the first-order case. Finally, the first main theorem (Thm. 7) also holds in the second-order setting:

Theorem 16 (Sufficient condition for improvement: second-order version). *If a second-order TERS* $(\Sigma, \mathcal{E}, \mathcal{R}, Ectx, Sclass)$ *is deterministic, value-invariant and locally coherent, then* $\mathcal{R}$ *is improvement w.r.t.* $\mathcal{E}$.

Examples. In the remainder of this section, we present three examples of TERS.

Example 17 (Left-to-right call-by-value lambda-calculus). A TERS **CBV**λ of the left-to-right call-by-value lambda-calculus is defined as follows, where t is a term, v is a value, M is a general metavariable, and V is a value metavariable. We use syntactic sugar $\lambda x.t \equiv \lambda(x.t), t\ u \equiv @(t, u)$.

Signature Σ	$\lambda : \langle 1 \rangle,\ @ : \langle 0, 0 \rangle$
Syntax class *Sclass*	**values** $V ::= \lambda x.t$
Evaluation contexts *Ectx*	$E ::= \Box \mid E\ t \mid v\ E$
Evaluation rules $\mathcal{E}$	**Refinement rules** $\mathcal{R}$
$(\lambda x.M[x])\ V \rightarrow M[V]$	$(\lambda x.M[x])\ V \Rightarrow M[V]$
	$\lambda x.V\ x \Rightarrow V$

We define $=_{Val}$ by the total relation $\top$, namely $\lambda x.t =_{Val} \lambda y.u$ for arbitrary $t, u \in \mathrm{M}_\Sigma$. This means that we observe only termination (since $Val \subseteq \mathsf{NF}(\rightarrow_{\mathcal{E}})$), identifying all values.

Example 18 (A simplified computational lambda-calculus λ_{ml*} [26,7]). A notion of evaluation for Sabry and Wadler's computational lambda-calculus λ_{ml*} [26] has been studied [7]. A TERS **Comp**λ_{ml*} of the computational lambda-calculus is defined in Fig. 4. We use syntactic sugar $\lambda x.t \equiv \lambda(x.t), t\ u \equiv @(t, u)$. We define $=_{Val}$ by the total relation $\top$. This means we observe only termination (since $Val \subseteq \mathsf{NF}(\rightarrow_{\mathcal{E}})$), identifying all values.

Signature Σ $\quad \lambda\colon \langle 1\rangle,\ @\colon \langle 0,0\rangle,\ \mathtt{let}\colon \langle 0,1\rangle,\ \mathtt{return}\colon \langle 0\rangle$

Syntax class *Sclass*

Values $\quad V, V' ::= x \mid \lambda x.P$

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

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

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 $\mathcal{R}$

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

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

$$\lambda x.V\ x \Rightarrow V \tag{r3}$$

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

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

Fig. 4. The TERS **Comp**λ_{ml*}

Example 19 (Effect handlers [25]). A TERS **Hndl** is defined in Fig. 5, where V, V_1, V_2 are value metavariables, H is a handler metavariable, and $P, P_1, P_2, \ldots$ are computation metavariables.

We only consider two operations $\mathtt{op}_1, \mathtt{op}_2$ and two handlers: $\mathtt{handler}_1$ for catching the first operation $\mathtt{op}_1$ and $\mathtt{handler}_0$ for catching no operation, for simplicity. We change the evaluation rule (7) to be the so-called *shallow handling*; the original, *deep handling*, rule [25] can be accommodated to a TERS, but this TERS would not be well-behaved[4].

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[5]. The refinement rules are numbered according to the original presentation [25, Fig. 7].

We define $=_{Val}$ for the TERS **Hndl** as follows, where v is any value.

$$x =_{Val} v \qquad \mathtt{true} =_{Val} \mathtt{true} \qquad \mathtt{false} =_{Val} \mathtt{false} \qquad \mathtt{fun}(x.p) =_{Val} \mathtt{fun}(x'.p')$$
$$\mathtt{handler}_1(x.p, x.k.p_1) =_{Val} \mathtt{handler}_1(x'.p', x'.k'.p'_1)$$
$$\mathtt{handler}_0(x.p) =_{Val} \mathtt{handler}_0(x'.p')$$

This means that we distinguish each ground type value (i.e. boolean value), and observe merely termination for other values (i.e. functions and handlers), although the TERS **Hndl** is untyped.

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

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

Signature Σ

$\texttt{true}\colon\langle 0\rangle$, $\texttt{false}\colon\langle 0\rangle$, $\texttt{fun}\colon\langle 1\rangle$, $@\colon\langle 0,0\rangle$, $\texttt{return}\colon\langle 0\rangle$, $\texttt{op}_1\colon\langle 0,1\rangle$, $\texttt{op}_2\colon\langle 0,1\rangle$, $\texttt{handler}_1\colon\langle 1,2\rangle$, $\texttt{handler}_0\colon\langle 1\rangle$, $\texttt{do}\colon\langle 0,1\rangle$, $\texttt{if}\colon\langle 0,0,0\rangle$, $\texttt{with\_handle}\colon\langle 0,0\rangle$

Syntax class *Sclass*

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

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

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

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

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

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

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

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

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

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

$\texttt{fun}(x.P[x])\ V \to P[V]$ (5)

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

$\texttt{with\_handle}(h_1, \texttt{return}(V)) \to P[V]$ (6)

$\texttt{with\_handle}(h_1, \texttt{op}_1(V, y.P'[y])) \to P_1[V, \texttt{fun}(y.P'[y])]$ (7)

$\texttt{with\_handle}(h_1, \texttt{op}_2(V, y.P'[y])) \to \texttt{op}_2(V, y.\texttt{with\_handle}(h_1, P'[y]))$ (8)

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

$\texttt{with\_handle}(h_0, \texttt{return}(V)) \to P[V]$ (9)

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

Refinement rules $\mathcal{R}$

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

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

$\texttt{fun}(x.F\ x) \Rightarrow F$ (r9)

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

Fig. 5. The TERS **Hndl**

6 Second-order critical pair analysis for local coherence

Critical pairs. The following definitions are analogous to those of first-order TERS. The definition of critical pairs is again standard, akin to commutation.

Definition 20 (Unifiers).

- 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 21 (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 quadruple $(l_1 \twoheadrightarrow_1 r_1, l_2 \twoheadrightarrow_2 r_2, p, \theta)$ 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.
- The substitution θ is a most general unifier between $l_1|_p$ and l_2.

Definition 22 (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 23. 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.

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. 12), except for the last two conditions which ensure that evaluation and refinement are consistent with syntax classes.

Definition 24 (Well-behaved TERS). A TERS $(\Sigma, \mathcal{E}, \mathcal{R}, \mathit{Ectx}, \mathit{Sclass})$ is *well-behaved* if it satisfies the following.

1. For any $C_1, C_2 \in \mathit{Ctx}$, if $C_1[C_2] \in \mathit{Ectx}$ then $C_1, C_2 \in \mathit{Ectx}$.
2. For any $E \in \mathit{Ectx}$ and $C' \in \mathit{Ctx}$, if $E \Rightarrow_{\mathcal{R}} C'$ then $C' \in \mathit{Ectx}$.
3. For any $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$ and any metavariable N that is not a value metavariable, the following holds.
   (a) The metavariable N appears at most once in l, and at most once in r.
   (b) Let p be the position of N in l. For the position q of N in r, if $l[\square]_p \in \mathit{Ectx}$ then $r[\square]_q \in \mathit{Ectx}$.
4. For any $(l \to_{\mathcal{E}} r) \in \mathcal{E}$, any metavariable N appears at most once in l.
5. For any $(l \Rightarrow_{\mathcal{R}} r) \in \mathcal{R}$ and any $\mathsf{valid}\,\theta$, if $l\theta$ belongs to a class then $r\theta$ belongs to the same class.
6. For any $(l \to_{\mathcal{E}} r) \in \mathcal{R}$ and any $\mathsf{valid}\,\theta$, if $l\theta$ belongs to a class then $r\theta$ belongs to the same class.

Theorem 25 (Critical pair theorem). A well-behaved TERS is locally coherent if and only if every critical pair is joinable.

The three examples. The TERSs **CBV**λ, **Comp**λ_{ml*} and **Hndl** are deterministic, value-invariant and locally coherent (Prop. 34, Prop. 35 & Prop. 36 in Appendix). By Thm. 7, every refinement $\mathcal{R}$ in the examples is *improvement* w.r.t. the corresponding evaluation $\mathcal{E}$.

7 Related work

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, 6] 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.

In the first-order setting, Lucas' *context-sensitive rewriting* [20] is capable of restricting where rewriting may happen, by means of a *replacement map* $\mu\colon \Sigma \to \mathbb{N}^*$. It is possible to encode any replacement map into evaluation contexts. 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. This can be encoded into evaluation contexts as $E ::= \square \mid \mathtt{if}(E, s_1, s_2)$. Another example is $\mu(+) = \{1, 2\}$ that specifies both of the two arguments of + can be rewritten. This can be encoded as $E ::= \square \mid E + t \mid t + E$. Every context-sensitive rewriting system can be simulated by a first-order TES in this way. Advantages of TERS are that (1) TES can also control the evaluation order easily (e.g. the left-to-right evaluation of function application), and (2) we have also formulated second-order TES with refinements as TERS.

Another term-rewriting alternative to evaluation contexts is *rewriting strategies* [16] that provide a way of determinising rewriting. Evaluation typically comes with a convenient inductive structure, which is lacking in strategies.

There is rich literature on methodologies for proving observational equivalence [1, 18, 24, 28]. Some methodologies have been applied to effect handlers [5, 4]. 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 [13, 14]. Our prototype analyzer based on this technology could automatically check the joinablity of all the critical pairs in the examples (see Appendix C for **Hndl**). There are few works on automating observational equivalence proofs for functional programs. Known examples, including the tool SyTeCi [17], are based on or inspired by algorithmic game semantics [2].

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 [29] and time improvement [19]. This line of work, however, does not come with any form of automation.

8 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.

Acknowledgement. The authors are supported by JSPS, KAKENHI Project No. 20H04164, Japan. K.M. is also supported by JSPS, KAKENHI Project No. 22K17850, Japan.

References

1. Abramsky, S.: The lazy lambda-calculus, pp. 65–117. Addison Wesley (1990)
2. Abramsky, S.: Algorithmic game semantics: a tutorial introduction. In: NATO Advanced Study Institute 2001. pp. 21–47 (2001)
3. Aoto, T., Toyama, Y.: A reduction-preserving completion for proving confluence of non-terminating term rewriting systems. Log. Methods Comput. Sci. **8**(1) (2012). https://doi.org/10.2168/LMCS-8(1:31)2012
4. Biernacki, D., Lenglet, S., Polesiuk, P.: A complete normal-form bisimilarity for algebraic effects and handlers. In: Ariola, Z.M. (ed.) 5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference). LIPIcs, vol. 167, pp. 7:1–7:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPICS.FSCD.2020.7, `https://doi.org/10.4230/LIPIcs.FSCD.2020.7`
5. Biernacki, D., Piróg, M., Polesiuk, P., Sieczkowski, F.: Handle with care: relational interpretation of algebraic effects and handlers. Proc. ACM Program. Lang. **2**(POPL), 8:1–8:30 (2018). https://doi.org/10.1145/3158096, `https://doi.org/10.1145/3158096`
6. Faggian, C., Guerrieri, G., de'Liguoro, U., Treglia, R.: On reduction and normalization in the computational core. Math. Struct. Comput. Sci. **32**(7), 934–981 (2022). https://doi.org/10.1017/S0960129522000433, `https://doi.org/10.1017/S0960129522000433`
7. Faggian, C., Guerrieri, G., Treglia, R.: Evaluation in the computational calculus is non-confluent. In: 10th International Workshop of Confluence, IWC 2021. pp. 31–36 (2021), `http://www.lix.polytechnique.fr/iwc2021/papers/IWC_2021_paper_6.pdf`
8. Felleisen, M.: lambda-v-cs: An extended lambda-calculus for scheme. In: Chailloux, J. (ed.) Proceedings of the 1988 ACM Conference on LISP and Functional Programming, LFP 1988, Snowbird, Utah, USA, July 25-27, 1988. pp. 72–85. ACM (1988). https://doi.org/10.1145/62678.62686, `https://doi.org/10.1145/62678.62686`
9. Felleisen, M.: The theory and practice of first-class prompts. In: Ferrante, J., Mager, P. (eds.) Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988. pp. 180–190. ACM Press (1988). https://doi.org/10.1145/73560.73576, `https://doi.org/10.1145/73560.73576`
10. Fiore, M.: Second-order and dependently-sorted abstract syntax. In: Proc. of LICS'08. pp. 57–68 (2008)
11. Fiore, M., Mahmoud, O.: Second-order algebraic theories. In: Proc. of MFCS'10. pp. 368–380. LNCS 6281 (2010)
12. Hamana, M.: Free Σ-monoids: A higher-order syntax with metavariables. In: Chin, W. (ed.) Programming Languages and Systems: Second Asian Symposium, APLAS 2004, Taipei, Taiwan, November 4-6, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3302, pp. 348–363. Springer (2004). https://doi.org/10.1007/978-3-540-30477-7_23
13. Hamana, M.: How to prove decidability of equational theories with second-order computation analyser SOL. J. Funct. Program. **29**, e20 (2019). https://doi.org/10.1017/S0956796819000157, `https://doi.org/10.1017/S0956796819000157`
14. Hamana, M., Abe, T., Kikuchi, K.: Polymorphic computation systems: Theory and practice of confluence with call-by-value. Sci. Comput. Program. **187**, 102322 (2020). https://doi.org/10.1016/J.SCICO.2019.102322, `https://doi.org/10.1016/j.scico.2019.102322`
15. Huet, G.P.: Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. J. ACM **27**(4), 797–821 (1980). https://doi.org/10.1145/322217.322230

16. Huet, G.P., Lévy, J.: Computations in orthogonal rewriting systems. In: Lassez, J., Plotkin, G.D. (eds.) Computational Logic - Essays in Honor of Alan Robinson. pp. 395–443. The MIT Press (1991)
17. Jaber, G.: Syteci: automating contextual equivalence for higher-order programs with references. Proc. ACM Program. Lang. **4**(POPL), 59:1–59:28 (2020). https://doi.org/10.1145/3371127, `https://doi.org/10.1145/3371127`
18. Koutavas, V., Levy, P., Sumii, E.: From applicative to environmental bisimulation. Elect. Notes in Theor. Comp. Sci. **276**, 215–235 (2011). https://doi.org/10.1016/j.entcs.2011.09.023
19. Lago, U.D., Gavazzo, F.: Effectful normal form bisimulation. In: Caires, L. (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. Lecture Notes in Computer Science, vol. 11423, pp. 263–292. Springer (2019). https://doi.org/10.1007/978-3-030-17184-1_10, `https://doi.org/10.1007/978-3-030-17184-1_10`
20. Lucas, S.: Context-sensitive rewriting. ACM Comput. Surv. **53**(4), 78:1–78:36 (2021). https://doi.org/10.1145/3397677, `https://doi.org/10.1145/3397677`
21. Miller, D.: A logic programming language with lambda-abstraction, function variables, and simple unification. J. Log. Comput. **1**(4), 497–536 (1991). https://doi.org/10.1093/logcom/1.4.497, `https://doi.org/10.1093/logcom/1.4.497`
22. Morris Jr, J.H.: Lambda-calculus models of programming languages. Ph.D. thesis, Massachusetts Institute of Technology (1969)
23. Muroya, K.: Hypernet semantics of programming languages. Ph.D. thesis, University of Birmingham (2020), `https://etheses.bham.ac.uk/id/eprint/10433/`
24. Plotkin, G.D.: Lambda-definability and logical relations (1973), memorandum SAI-RM-4
25. Pretnar, M.: An introduction to algebraic effects and handlers. invited tutorial paper. In: Ghica, D.R. (ed.) The 31st Conference on the Mathematical Foundations of Programming Semantics, MFPS 2015, Nijmegen, The Netherlands, June 22-25, 2015. Electronic Notes in Theoretical Computer Science, vol. 319, pp. 19–35. Elsevier (2015). https://doi.org/10.1016/J.ENTCS.2015.12.003, `https://doi.org/10.1016/j.entcs.2015.12.003`
26. Sabry, A., Wadler, P.: A reflection on call-by-value. In: Harper, R., Wexelblat, R.L. (eds.) Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming, ICFP 1996, Philadelphia, Pennsylvania, USA, May 24-26, 1996. pp. 13–24. ACM (1996). https://doi.org/10.1145/232627.232631, `https://doi.org/10.1145/232627.232631`
27. Sands, D.: Total correctness by local improvement in the transformation of functional programs. ACM Trans. Program. Lang. Syst. **18**(2), 175–234 (1996). https://doi.org/10.1145/227699.227716
28. Statman, R.: Logical relations and the typed lambda-calculus. Information and Control **65**(2/3), 85–97 (1985). https://doi.org/10.1016/S0019-9958(85)80001-2
29. Sumii, E.: A bisimulation-like proof method for contextual properties in untyped lambda-calculus with references and deallocation. Theor. Comput. Sci. **411**(51-52), 4358–4378 (2010). https://doi.org/10.1016/J.TCS.2010.09.009, `https://doi.org/10.1016/j.tcs.2010.09.009`
30. Toyama, Y.: Commutativity of term rewriting systems, pp. 393–407. North-Holland (1988)

A Omitted proofs

A.1 Proofs for Sec. 3 and Sec. 4

Theorem 26 (Thm. 7: sufficient condition for improvement). If a TERS is deterministic, value-invariant and locally coherent, then it supports improvement.

Proof. Take arbitrary $k \in \mathbb{N}$ and $t, u \in \mathrm{T}_\Sigma$ such that $t \Rightarrow_{\mathcal{R}} u$ and $t \overset{k}{\rightarrow}_{\mathcal{E}} v \in \mathit{Val}$. We first prove that $t \Rightarrow_{\mathcal{R}} u$ and $t \overset{k}{\rightarrow}_{\mathcal{E}} v$ imply $u \overset{m}{\rightarrow}_{\mathcal{E}} v'$, $v =_{\mathit{Val}} v'$ 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 $t \rightarrow_{\mathcal{E}} t' \overset{k-1}{\rightarrow}_{\mathcal{E}} v$. 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 $t' \overset{l}{\rightarrow}_{\mathcal{E}} t''$, $u \overset{n}{\rightarrow}_{\mathcal{E}} u'$, $t'' \overset{m}{\Rightarrow}_{\mathcal{R}} u'$ and $1 + l \geq n$. Because the TERS $(\mathcal{E}, \mathcal{R})$ is deterministic, t'' must appear in the sequence $t' \overset{k-1}{\rightarrow}_{\mathcal{E}} v$, and hence $t \rightarrow_{\mathcal{E}} t' \overset{l}{\rightarrow}_{\mathcal{E}} t'' \overset{k-l-1}{\rightarrow}_{\mathcal{E}} v$. We prove that we have the following situation:

$$\begin{tikzcd}
& t \arrow[dl, Rightarrow] \arrow[dr] & \\
u \arrow[d, "n"'] & & t' \arrow[d, "l"] \\
u' \arrow[d, "n'"'] & & t'' \arrow[ll, Rightarrow, "m"] \arrow[d, "k-l-1"] \\
v' \arrow[rr, equal, "\mathit{Val}"'] & & v
\end{tikzcd}$$

namely that there exist $n' \in \mathbb{N}$ and $v' \in \mathit{Val}$ such that $u' \overset{n'}{\rightarrow}_{\mathcal{E}} v'$ 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 $u'' \overset{n''}{\rightarrow}_{\mathcal{E}} v''$ such that $v'' =_{\mathit{Val}} v$ and $k - l - 1 \geq n''$. Furthermore, by I.H. of the outer induction on n'', we have $u' \overset{n'}{\rightarrow}_{\mathcal{E}} v'$ such that $v'' =_{\mathit{Val}} v'$ and $n'' \geq n'$. We finally have $k \geq n + n'$.

As a result, we have $u \overset{n+n'}{\rightarrow}_{\mathcal{E}} v'$ 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 $C[t] \overset{k}{\rightarrow}_{\mathcal{E}} v$ imply $C[u] \overset{m}{\rightarrow}_{\mathcal{E}} v'$ such that $k \geq m$ and $v =_{\mathit{Val}} v'$, for any $v \in \mathit{Val}$. □

Lemma 27 (Lem. 11). 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. $\square$

Theorem 28 (Thm. 13: 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 $p \in Pos(s)$, $(l \Rightarrow r) \in \mathcal{R}$ and $\mathsf{subst}\,\theta$ such that $s|_p = l\theta$, $t_1 = s[r\theta]_p$ and $s[\Box]_p \in Ctx$. 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 $p' \in Pos(l\theta)$, $(l' \to r') \in \mathcal{E}$ and $\mathsf{subst}\,\theta'$ such that $(l\theta)|_{p'} = l'\theta'$, $t_2 = (l\theta)[r'\theta']_{p'}$ and $(l\theta)[\Box]_{p'} \in Ectx$. 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, 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: $q_1 \in Pos(l)$, $l|_{q_1} = y$, $q_2 \in Pos(y\theta)$, and $p' = q_1q_2$. Because of the condition (1) of Def. 12, $(l\theta)[\Box]_{p'} \in Ectx$ 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 (3a) of Def. 12. 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 (3b) of Def. 12, 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 $p' \in Pos(s)$, $(l' \to r') \in \mathcal{E}$ 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 $p' \in Pos(s)$.

- When $p' = \varepsilon$, we have $s = l'\theta'$ and $t_2 = r'\theta'$.
  - If p is a non-variable position of 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: $q_1 \in Pos(l')$, $l'|_{q_1} = y$, $q_2 \in Pos(y\theta')$, and $p = q_1q_2$. The variable y appears at most once in l', due to the condition (4) of Def. 12. 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 $f(\dots, \overline{x_i}.u_i[\square]_{p_t}, \dots) \in \mathit{Ectx}$, we have $f(\dots, \overline{x_i}.\square, \dots) \in \mathit{Ectx}$ too, thanks to the condition (1) of Def. 12. Therefore, joinability of the $(\mathcal{R}, \mathcal{E})$-peak Q implies joinability of the $(\mathcal{R}, \mathcal{E})$-peak P'.
  - When $i' \neq i$, we can assume that $i' < i$ without loss of generality. The $(\mathcal{R}, \mathcal{E})$-peak

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

    is joinable (to $f(\dots, \overline{x_{i'}}.u_{i'}[r'\theta']_{p'_t}, \dots, \overline{x_i}.u_i[r\theta]_{p_t}, \dots)$), thanks to the condition (2) of Def. 12.

□

A.2 The TERS Nats

Proposition 29. The TERS **Nats** is deterministic, value-invariant and locally coherent.

Proof. The TERS **Nats** is deterministic, because evaluation rules concern distinct symbols.

To prove value-invariance, we assume $v \Rrightarrow_{\mathcal{R}} u$ for some $v \in \mathit{Val}$ and $u \in \mathrm{T}_\Sigma$. It must hold that $v = \mathtt{s}^n(0)$, and in this case, the refinement $v \Rrightarrow_{\mathcal{R}} u$ is impossible. The TERS **Nats** is trivially value-invariant.

To prove local coherence, we use Thm. 13. We first show that the TERS **Nats** is well-behaved. The condition (1) of Def. 24 is trivially satisfied. As for the condition (2), each evaluation context $E \in \mathit{Ectx}$ never includes the constant `nats`, and hence the refinement rule cannot be applied to $E[t]$ to obtain $C'[t]$. Therefore the condition (2) is trivially satisfied. The other conditions of well-behavedness are easy to check. We then show that any critical pair is joinable. There is only one critical pair, and it is indeed joinable as follows.

$$\begin{tikzcd}
 & \mathtt{tl}(\mathtt{inc}(\mathtt{nats})) \arrow[dl, Rightarrow] \arrow[dr] & \\
\mathtt{inc}(\mathtt{tl}(\mathtt{nats})) \arrow[d] & & \mathtt{tl}(\mathtt{inc}(0 : \mathtt{inc}(\mathtt{nats}))) \arrow[d] \\
\mathtt{inc}(\mathtt{tl}(0 : \mathtt{inc}(\mathtt{nats}))) \arrow[dr] & & \mathtt{tl}(\mathtt{s}(0) : \mathtt{inc}(\mathtt{inc}(\mathtt{nats}))) \arrow[dl] \\
 & \mathtt{inc}(\mathtt{inc}(\mathtt{nats})) &
\end{tikzcd}$$

□

A.3 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. 12). 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	$+: 2,\ -: 2,\ \overset{?}{\equiv}: 2,\ \mathsf{s}: 1,\ 0: 0$
Values *Val*	$V ::= 0 \mid \mathsf{s}(V)$
Evaluation contexts *Ectx*	$E ::= \square \mid \mathsf{s}(E) \mid E + t \mid E - t \mid v - E$
Evaluation rules $\mathcal{E}$	**Refinement rule** $\mathcal{R}$
$0 + x \to x$	$x - x \Rightarrow 0$
$\mathsf{s}(x) + y \to \mathsf{s}(x + y)$	$0 \Rightarrow x - x$
$0 - x \to 0$	
$\mathsf{s}(x) - \mathsf{s}(y) \to x - y$	
$x \overset{?}{\equiv} x \to 0$	

We define $=_{Val}$ by the syntactic equality $\equiv$. The operation $\overset{?}{\equiv}$ checks syntactic equality.

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

$$\begin{tikzcd}
& (\mathsf{s}(x) + y) - (\mathsf{s}(x) + y) \arrow[dl, Rightarrow] \arrow[dr] & \\
0 & & \mathsf{s}(x + y) - (\mathsf{s}(x) + y)
\end{tikzcd}$$

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

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

$$\begin{tikzcd}
& 0 \arrow[dl, Rightarrow] \arrow[dr, equal] & \\
0 - 0 \arrow[rr] & & 0
\end{tikzcd}$$

This $(\mathcal{R}, \mathcal{E})$-peak is not joinable with respect to our definition of joinability (see Def. 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.

$$\begin{tikzcd}
& (\mathsf{s}(x) + y) \overset{?}{\equiv} (\mathsf{s}(x) + y) \arrow[dl, Rightarrow] \arrow[dr] & \\
\mathsf{s}(x + y) \overset{?}{\equiv} (\mathsf{s}(x) + y) & & 0
\end{tikzcd}$$

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

A.4 Proofs for Sec. 5 and Sec. 6

Lemma 30 (Lem. 23). 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. □

Theorem 31 (Thm. 25: 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 $p \in \mathit{Pos}(s)$, $(l \Rightarrow r) \in \mathcal{R}$ 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 $l\theta \rightarrow_{\mathcal{E}} t_2$, there exist $p' \in \mathit{Pos}(l\theta)$, $(l' \rightarrow r') \in \mathcal{E}$ 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 $l \rightarrow r$ 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, 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: $q_1 \in \mathit{Pos}(l)$, $l|_{q_1} = N[\overline{y}]$, $q_2 \in \mathit{Pos}((N[\overline{y}])\theta)$, and $p' = q_1 q_2$. Because of the condition (1) of Def. 24, $(l\theta)[\Box]_{p'} \in \mathit{Ectx}$ implies $l[\Box]_{q_1}, (N[\overline{y}])\theta[\Box]_{q_2} \in \mathit{Ectx}$. In particular, the latter means that $(N[\overline{y}])\theta \notin \mathsf{NF}(\rightarrow_{\mathcal{E}})$, 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 (3a) of Def. 24. 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 $l' \rightarrow r'$ can be applied to t_1 thanks to the condition (3b) of Def. 24, and the rule $l \Rightarrow r$ can be applied to t_2, thanks to the condition (6) of Def. 24. 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 $s \rightarrow_{\mathcal{E}} t_2$, there exist $p' \in \mathit{Pos}(s)$, $(l' \rightarrow r') \in \mathcal{E}$ 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 $p' \in \mathit{Pos}(s)$.

– When $p' = \varepsilon$, we have $s = l'\theta'$ and $t_2 = r'\theta'$.
  - If p is a non-metavariable position of 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 metavariable M such that: $q_1 \in Pos(l')$, $l'|_{q_1} = M[\overline{y}]$, $q_2 \in Pos(M[\overline{y}]\theta')$, and $p = q_1 q_2$. The metavariable M appears at most once in l', due to the condition (4) of Def. 24. We can apply the rule $l' \to r'$ to t_1. The substitution θ' is valid, thanks to the condition (5) of Def. 24. 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 $f(\ldots, \overline{x_i}.u_i[\square]_{p_t}, \ldots) \in Ectx$, we have $f(\ldots, \overline{x_i}.\square, \ldots) \in Ectx$ too, thanks to the condition (1) of Def. 24. 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

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

    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 (2) of Def. 24.

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 $p' \in Pos(s)$, $(l' \to r') \in \mathcal{E}$ and valid θ' such that $s|_{p'} = l'\theta'$, $t_2 = s[r'\theta']_{p'}$ and $s[\square]_{p'} \in Ectx$. We proceed by case analysis on $p' \in Pos(s)$.

– 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

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

$\square$

A.5 The TERS CBVλ and Hndl

B Determinism

We will use a sufficient condition for a TES to be deterministic, namely *decisiveness*.

Definition 32 (decisiveness). A TES $(\Sigma, \mathcal{E}, Ectx, Val)$ is *decisive* if each $t \in \mathrm{T}_\Sigma$ satisfies either of the following:

1. $t \in Val$,
2. there uniquely exist $(l \to r) \in \mathcal{E}$, subst θ and $E \in Ectx$ such that $t = E[l\theta]$,

3. there uniquely exist a variable x and $E \in \mathit{Ectx}$ such that $t = E[x]$.

Proposition 33 (sufficient condition for determinism). If a TES is decisive, then it is deterministic.

Proof. Let $t \to_{\mathcal{E}} s_1$ and $t \to_{\mathcal{E}} s_2$. Because the TES is decisive, t satisfies either the three conditions in Def. 32. Since the left-hand side of each evaluation rule is not a variable, to make the evaluation $t \to_{\mathcal{E}} s_1$ and $t \to_{\mathcal{E}} s_2$ happen, only the case (2) is possible. In this case, both s_1 and s_2 must be $E[r\theta]$. □

Proposition 34. The TERS **CBV**λ is deterministic, value-invariant and locally coherent.

Proof. The TERS **CBV**λ is deterministic, because it is decisive.

To prove value-invariance, we assume $\lambda x.t \Rightarrow_{\mathcal{R}} u$. There are two possible cases.

– When $u = \lambda x.t'$ for some t' such that $t \Rightarrow_{\mathcal{R}} t'$, we have $\lambda x.t =_{\mathit{Val}} \lambda x.t'$.

– When $t = \lambda x.(\lambda y.t')\ x$, it must be that $u = \lambda y.t'$, and we have $\lambda x.(\lambda y.t')\ x =_{\mathit{Val}} \lambda y.t'$.

Therefore the TERS **CBV**λ is value-invariant.

To prove local coherence, we use Thm. 13.

Firstly, the TERS **CBV**λ is well-behaved. The condition (1) of Def. 24 is trivially satisfied. We can show that the condition (2) is satisfied by straightforward induction on $E \in \mathit{Ectx}$. The condition (6) is satisfied, because any instance of the lhs of the evaluation rule never belongs to a syntax class (i.e. the value class). The condition (5) is also satisfied; the second refinement rule always turns a value into a value. The other conditions of well-behavedness are easy to check.

We then show that any critical pair is joinable. There are two critical pairs, which are for the second refinement rule (the η-rule) and the evaluation rule. These critical pairs are joinable as follows.

$$\begin{tikzcd}
& (\lambda x.V\ x)\ V' \arrow[dl, Rightarrow] \arrow[dr] & \\
V\ V' \arrow[rr, equal] & & V\ V'
\end{tikzcd}$$

$$\begin{tikzcd}
& (\lambda x'.M'[x'])\ (\lambda x.V\ x) \arrow[dl, Rightarrow] \arrow[dr] & \\
(\lambda x'.M'[x'])\ V \arrow[dr] & & M'[\lambda x.V\ x] \arrow[dl, Rightarrow] \\
& M'[V] &
\end{tikzcd}$$

□

Proposition 35. The TERS **Comp**λ_{ml*} is deterministic, value-invariant and locally coherent.

Proof. Firstly, the TERS **Comp**λ_{ml*} is deterministic, because the two evaluation rules consume different head symbols. Secondly, the TERS is value-invariant, thanks to the equivalence $=_{\mathit{Val}}$ being the total order.

To prove local coherence, we use Thm. 13.

The TERS **Comp**λ_{ml*} is well-behaved. The condition (1) of Def. 24 is trivially satisfied. We can show that the condition (2) is satisfied by induction on $E \in \mathit{Ectx}$ as follows.

– When $E = \square$, no refinement rule applies to x, so this case is impossible.
– When $E = \mathtt{let}(E', x.P)$, we have $\mathtt{let}(E'[t], x.P) \Rightarrow_{\mathcal{R}} C'[t]$. There are four possibilities.
  - If $E'[z] \Rightarrow_{\mathcal{R}} C''[z]$ such that $C' = \mathtt{let}(C'', x.P)$, we have $C'' \in \mathit{Ectx}$ by I.H., and hence $C' \in \mathit{Ectx}$.
  - If $P \Rightarrow_{\mathcal{R}} P'$ such that $C' = \mathtt{let}(E', x.P')$, we have $C' \in \mathit{Ectx}$.
  - If the refinement rule (r4) is applied at the root position of $E[z]$, we have $\mathtt{let}(E'[z], x.\mathtt{return}(x)) \Rightarrow_{\mathcal{R}} E'[z]$. We have $C' = E' \in \mathit{Ectx}$.
  - If the refinement rule (r5) is applied at the root position of $E[z]$, we have $\mathtt{let}(\mathtt{let}(E''[z], x.P[x]), y.P'[y]) \Rightarrow_{\mathcal{R}} \mathtt{let}(E''[z], x.\mathtt{let}(P[x], y.P'[y]))$. We have $C' = \mathtt{let}(E'', x.\mathtt{let}(P[x], y.P'[y])) \in \mathit{Ectx}$.

The conditions (6) and (5) are also satisfied; note that instances of $P[V]$ are all computations. The other conditions of well-behavedness are easy to check.

We finally show that any critical pair is joinable. There are the following three critical pairs, which are all joinable. In the following, arrows $\rightarrow, \Rightarrow$ are labelled by a number that indicates which evaluation/refinement rule is applied.

$$\begin{tikzcd}
& (\lambda x.V\ x)\ V' \arrow[dl, Rightarrow, "r3"'] \arrow[dr, "1"] & \\
V\ V' \arrow[rr, equal] & & V\ V'
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{let}(\mathtt{return}(V), x.\mathtt{return}(x)) \arrow[dl, Rightarrow, "r4"'] \arrow[dr, "2"] & \\
\mathtt{return}(V) \arrow[rr, equal] & & \mathtt{return}(V)
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{let}(\mathtt{let}(\mathtt{return}(V), x.P[x]), x'.P'[x']) \arrow[dl, Rightarrow, "r5"'] \arrow[dr, "2"] & \\
\mathtt{let}(\mathtt{return}(V), x.\mathtt{let}(P[x], x'.P'[x'])) \arrow[dr, "2"'] & & \mathtt{let}(P[V], x'.P'[x']) \arrow[dl, equal] \\
& \mathtt{let}(P[V], x'.P'[x']) &
\end{tikzcd}$$

□

Proposition 36. The TERS **Hndl** is deterministic, value-invariant and locally coherent.

Proof. Firstly, to establish that the TERS **Hndl** is deterministic, we show that, for any $t \in \mathrm{M}_\Sigma$, if $t = E[l\theta] = E'[l'\theta']$ for some $E, E' \in \mathit{Ectx}$, $(l \to r), (l' \to r') \in \mathcal{E}$ and $\mathsf{valid}\,\theta, \theta'$, then the decomposition is unique, namely $E = E'$ and the rules $l \to r$, $l' \to r'$ are variants of each other. This can be proved by induction on $E \in \mathit{Ectx}$.

– When $E = \square$, we have $t = l\theta$. By definition of evaluation rules, $E' = \square$ must hold, and $l \to r$ and $l' \to r'$ must be variants.

- When $E = \mathtt{do}(E', x.P)$, we have $t = \mathtt{do}(E'[l\theta], P)$. By I.H., $E'[l\theta]$ is the only possible decomposition. The meta-term t itself cannot be an instance of any lhs of evaluation rules. Therefore, the decomposition $E[l\theta]$ is unique.
- When $E = \mathtt{with\_handle}(H, E')$, we have $t = \mathtt{with\_handle}(H, E'[l\theta])$. The proof is the same as the previous case.

Consequently, the TERS **Hndl** is deterministic.

Secondly, by definition of $=_{Val}$, the TERS **Hndl** is value-invariant. In particular, the refinement rule (r9) turns a function into a variable or a function; the original function is identified with the resulting variable or function by $=_{Val}$.

To prove local coherence, we use Thm. 13.

The TERS **Hndl** is well-behaved. The condition (1) of Def. 24 is trivially satisfied. We can show that the condition (2) is satisfied by induction on $E \in Ectx$ as follows.

- When $E = \square$, no refinement rule applies to x, so this case is impossible.
- When $E = \mathtt{do}(E', x.P)$, we have $\mathtt{do}(E'[z], x.P) \Rightarrow_{\mathcal{R}} C'[z]$. There are four possibilities.
  - If $E'[z] \Rightarrow_{\mathcal{R}} C''[z]$ such that $C' = \mathtt{do}(C'', x.P)$, we have $C'' \in Ectx$ by I.H., and hence $C' \in Ectx$.
  - If $P \Rightarrow_{\mathcal{R}} P'$ such that $C' = \mathtt{do}(E', x.P')$, we have $C' \in Ectx$.
  - If the refinement rule (r3) is applied at the root position of $E[z]$, we have $\mathtt{do}(E'[z], x.\mathtt{return}(x)) \Rightarrow_{\mathcal{R}} E'[z]$. We have $C' = E' \in Ectx$.
  - If the refinement rule (r4) is applied at the root position of $E[z]$, we have $\mathtt{do}(\mathtt{do}(E''[z], x.P[x]), y.P'[y]) \Rightarrow_{\mathcal{R}} \mathtt{do}(E''[z], x.\mathtt{do}(P[x], y.P'[y]))$. We have $C' = \mathtt{do}(E'', x.\mathtt{do}(P[x], y.P'[y])) \in Ectx$.
- When $E = \mathtt{with\_handle}(H, E')$, we have $\mathtt{with\_handle}(H, E'[z]) \Rightarrow_{\mathcal{R}} C'[z]$. There are three possibilities.
  - If $E'[z] \Rightarrow_{\mathcal{R}} C''[z]$ such that $C' = \mathtt{with\_handle}(H, C'')$, we have $C'' \in Ectx$ by I.H., and hence $C' \in Ectx$.
  - If $H \Rightarrow_{\mathcal{R}} H'$ such that $C' = \mathtt{with\_handle}(H', E')$, we have $C' \in Ectx$.
  - If the refinement rule (r13) is applied at the root position of $E[z]$, we have $\mathtt{with\_handle}(\mathtt{handler}_0(x.P[x]), E'[z]) \Rightarrow_{\mathcal{R}} \mathtt{do}(E'[z], x.P[x])$. We have $C' = \mathtt{do}(E', x.P[x]) \in Ectx$.

The conditions (6) and (5) are also satisfied; note that instances of $P[V]$ are all computations. The other conditions of well-behavedness are easy to check.

We finally show that any critical pair is joinable. There are the following seven critical pairs, which are all joinable. In the following, arrows $\rightarrow$, $\Rightarrow$ are labelled by a number that indicates which evaluation/refinement rule is applied, and we set $h_0 \equiv$

$\mathtt{handler}_0(x.P[x])$, and $i \in [2]$.

$$\begin{tikzcd}
& \mathtt{do}(\mathtt{return}(V), x.\mathtt{return}(x)) \arrow[dl, Rightarrow, "r3"'] \arrow[dr, "1"] & \\
\mathtt{return}(V) \arrow[rr, equal] & & \mathtt{return}(V)
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{do}(\mathtt{op}_i(V, y.P[y]), x.\mathtt{return}(x)) \arrow[dl, Rightarrow, "r3"'] \arrow[dr, "2"] & \\
\mathtt{op}_i(V, y.P[y]) & & \mathtt{op}_i(V, y.\mathtt{do}(P[y], x.\mathtt{return}(x))) \arrow[ll, Rightarrow, "r3"']
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{do}(\mathtt{do}(\mathtt{return}(V), x.P[x]), x'.P'[x']) \arrow[dl, Rightarrow, "r4"'] \arrow[dr, "1"] & \\
\mathtt{do}(\mathtt{return}(V), x.\mathtt{do}(P[x], x'.P'[x'])) \arrow[dr, "1"'] & & \mathtt{do}(P[V], x'.P'[x']) \arrow[dl, equal] \\
& \mathtt{do}(P[V], x'.P'[x']) &
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{do}(\mathtt{do}(\mathtt{op}_i(V, x.P[x]), y.P_2[y]), z.P_3[z]) \arrow[dl, Rightarrow, "r4"'] \arrow[dr, "2"] & \\
\mathtt{do}(\mathtt{op}_i(V, x.P[x]), y.\mathtt{do}(P_2[y], z.P_3[z])) \arrow[d, "2"'] & & \mathtt{do}(\mathtt{op}_i(V, x.\mathtt{do}(P[x], y.P_2[y])), z.P_3[z]) \arrow[d, "2"] \\
\mathtt{op}_i(V, x.\mathtt{do}(P[x], y.\mathtt{do}(P_2[y], z.P_3[z]))) & & \mathtt{op}_i(V, x.\mathtt{do}(\mathtt{do}(P[x], y.P_2[y]), z.P_3[z])) \arrow[ll, Rightarrow, "r4"']
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{fun}(x.V\ x)\ V' \arrow[dl, Rightarrow, "r9"'] \arrow[dr, "5"] & \\
V\ V' \arrow[rr, equal] & & V\ V'
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{with\_handle}(h_0, \mathtt{return}(V)) \arrow[dl, Rightarrow, "r13"'] \arrow[dr, "9"] & \\
\mathtt{do}(\mathtt{return}(V), x.P[x]) \arrow[dr, "1"'] & & P[V] \arrow[dl, equal] \\
& P[V] &
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{with\_handle}(h_0, \mathtt{op}_i(V, y.P'[y])) \arrow[dl, Rightarrow, "r13"'] \arrow[dr, "10"] & \\
\mathtt{do}(\mathtt{op}_i(V, y.P'[y]), x.P[x]) \arrow[dr, "2"'] & & \mathtt{op}_i(V, y.\mathtt{with\_handle}(h_0, P'[y])) \arrow[dl, Rightarrow, "r13"] \\
& \mathtt{op}_i(V, y.\mathtt{do}(P'[y], x.P[x])) &
\end{tikzcd}$$

□

C Critical pair analysis of Hndl by our prototype analyzer

C.1 Definition of TERS Hndl

```
sigh = [signature|
  true : T
  false : T
  fun : (T -> T) -> T
  app : T,T -> T
  return : T -> T
  op1 : T, (T -> T) -> T
  op2 : T, (T -> T) -> T
  handler1 : (T -> T), (T,T -> T) -> T
  handler0 : (T -> T) -> T
  do : T, (T -> T) -> T
  if : T,T,T -> T
  with : T,T -> T
 |]

evals = [rule|
 (1-v) do(return(V),x.P[x]) => P[V]
 (2-1-v) do(op1(V, y.P1[y]), x.P2[x]) => op1(V, y.do(P1[y],x.P2[x]))
 (2-2-v) do(op2(V, y.P1[y]), x.P2[x]) => op2(V, y.do(P1[y],x.P2[x]))
 (3-v) if(true, P1, P2) => P1
 (4-v) if(false, P1, P2) => P2
 (5-v) fun(x.P[x])@V => P[V]
 (6-v) with(handler1(x.P[x],x.k.P1[x,k]), return(V)) => P[V]
 (7-v) with(handler1(x.P[x],x.k.P1[x,k]),op1(V,y.P'[y])) => P1[V,fun(y.P'[y])]
 (8-v) with(handler1(x.P[x],x.k.P1[x,k]),op2(V,y.P'[y])) => op2(V,y.with(handler1(x.P[x],x.k.P1[x,k]),P'[y]))
 (9-v)    with(handler0(x.P[x]),return(V)) => P[V]
 (10-1-v) with(handler0(x.P[x]),op1(V,y.P'[y])) => op1(V,y.with(handler0(x.P[x]),P'[y]))
 (10-2-v) with(handler0(x.P[x]),op2(V,y.P'[y])) => op2(V,y.with(handler0(x.P[x]),P'[y]))
 |]

refis = [rule|
 (r3-v) do(P,x.return(x)) => P
 (r4-v) do(do(P1,x.P2[x]),x.P3[x]) => do(P1, x1.do(P2[x1],x2.P3[x2]))
 (r9-v) fun(x.V@x) => V
 (r13-v) with(handler0(x.P[x]),P') => do(P',x.P[x])
 |]
```

C.2 Local coherence check

```
*Ex> lcoh evals refis

1: Overlap (1-v)-(r3-v)--- P'|-> return(V), P|-> z1.return(z1) ----------------
   (1-v) |do(return(V),x.P[x])| => P[V]
   (r3-v) do(P',x'.return(x')) => P'
                         do(return(V),x.return(x))
                  return(V) <-(1-v)-^-(r3-v)-> return(V)
                    ---> return(V) =OK= return(V) <---

2: Overlap (r4-v)-(1-v)--- P1|-> return(V'), P'|-> z1.P2[z1] ------------------
   (r4-v) do(|do(P1,x.P2[x])|,x.P3[x]) => do(P1,x1.do(P2[x1],x2.P3[x2]))
   (1-v) do(return(V'),x'.P'[x']) => P'[V']
                    do(do(return(V'),x.P2[x]),x.P3[x])
do(return(V'),x17.do(P2[x17],x27.P3[x27])) <-(r4-v)-^-(1-v)-> do(P2[V'],x.P3[x])
       ---> do(P2[V'],x27.P3[x27]) =OK= do(P2[V'],x.P3[x]) <---

3: Overlap (2-1-v)-(r3-v)--- P'|-> op1(V,y.P1[y]), P2|-> z1.return(z1) --------
   (2-1-v) |do(op1(V,y.P1[y]),x.P2[x])| => op1(V,y.do(P1[y],x.P2[x]))
   (r3-v) do(P',x'.return(x')) => P'
                    do(op1(V,y.P1[y]),x.return(x))
op1(V,y14.do(P1[y14],x14.return(x14))) <-(2-1-v)-^-(r3-v)-> op1(V,y.P1[y])
---> op1(V,y14.do(P1[y14],x14.return(x14))) =OK= op1(V,y.P1[y]) <---
```

```
4: Overlap (r4-v)-(2-1-v)--- P1|-> op1(V’,y’.P1’[y’]), P2’|-> z1.P2[z1] -------
   (r4-v) do(|do(P1,x.P2[x])|,x.P3[x]) => do(P1,x1.do(P2[x1],x2.P3[x2]))
   (2-1-v) do(op1(V’,y’.P1’[y’]),x’.P2’[x’]) => op1(V’,y’.do(P1’[y’],x’.P2’[x’]))
                 do(do(op1(V’,y’.P1’[y’]),x.P2[x]),x.P3[x])
do(op1(V’,y’.P1’[y’]),x121.do(P2[x121],x1.P3[x1])) <-(r4-v)-∧-(2-1-v)-> do(op1(V’,yd.do(P1’[yd],xd.P2[xd])),x.P3[x])
---> op1(V’,y23.do(P1’[y23],x23.do(P2[x23],x1.P3[x1]))) =OK= op1(V’,y26.do(do(P1’[y26],xd.P2[xd]),x26.P3[x26])) <---

5: Overlap (2-2-v)-(r3-v)--- P’|-> op2(V,y.P1[y]), P2|-> z1.return(z1) --------
   (2-2-v) |do(op2(V,y.P1[y]),x.P2[x])| => op2(V,y.do(P1[y],x.P2[x]))
   (r3-v) do(P’,x’.return(x’)) => P’
                       do(op2(V,y.P1[y]),x.return(x))
op2(V,y33.do(P1[y33],x33.return(x33))) <-(2-2-v)-∧-(r3-v)-> op2(V,y.P1[y])
---> op2(V,y33.do(P1[y33],x33.return(x33))) =OK= op2(V,y.P1[y]) <---

6: Overlap (r4-v)-(2-2-v)--- P1|-> op2(V’,y’.P1’[y’]), P2’|-> z1.P2[z1] -------
   (r4-v) do(|do(P1,x.P2[x])|,x.P3[x]) => do(P1,x1.do(P2[x1],x2.P3[x2]))
   (2-2-v) do(op2(V’,y’.P1’[y’]),x’.P2’[x’]) => op2(V’,y’.do(P1’[y’],x’.P2’[x’]))
                 do(do(op2(V’,y’.P1’[y’]),x.P2[x]),x.P3[x])
do(op2(V’,y’.P1’[y’]),x140.do(P2[x140],x240.P3[x240])) <-(r4-v)-∧-(2-2-v)-> do(op2(V’,yd.do(P1’[yd],xd.P2[xd])),x.P3[x])
---> op2(V’,y42.do(P1’[y42],x42.do(P2[x42],x240.P3[x240]))) =OK= op2(V’,y45.do(do(P1’[y45],xd.P2[xd]),x45.P3[x45])) <---

7: Overlap (5-v)-(r9-v)--- P|-> z1.app(V’,z1) --------------------------------
   (5-v) |fun(x.P[x])|@V => P[V]
   (r9-v) fun(x’.app(V’,x’)) => V’
                           app(fun(x.app(V’,x)),V)
                   app(V’,V) <-(5-v)-∧-(r9-v)-> app(V’,V)
                     ---> app(V’,V) =OK= app(V’,V) <---

8: Overlap (9-v)-(r13-v)--- P’|-> z1.P[z1], Pd’|-> return(V) -----------------
   (9-v) |with(handler0(x.P[x]),return(V))| => P[V]
   (r13-v) with(handler0(x’.P’[x’]),Pd’) => do(Pd’,x’.P’[x’])
                      with(handler0(x.P[x]),return(V))
                        P[V] <-(9-v)-∧-(r13-v)-> do(return(V),xd66.P[xd66])
                          ---> P[V] =OK= P[V] <---

9: Overlap (10-1-v)-(r13-v)--- P’|-> z1.P[z1], Pd’|-> op1(V,y.Pd[y]) ----------
   (10-1-v) |with(handler0(x.P[x]),op1(V,y.Pd[y]))| => op1(V,y.with(handler0(x.P[x]),Pd[y]))
   (r13-v) with(handler0(x’.P’[x’]),Pd’) => do(Pd’,x’.P’[x’])
                    with(handler0(x.P[x]),op1(V,y.Pd[y]))
op1(V,y72.with(handler0(x72.P[x72]),Pd[y72])) <-(10-1-v)-∧-(r13-v)-> do(op1(V,y.Pd[y]),xd73.P[xd73])
---> op1(V,y72.with(handler0(x72.P[x72]),Pd[y72])) =OK= op1(V,y76.do(Pd[y76],x76.P[x76])) <---

10: Overlap (10-2-v)-(r13-v)--- P’|-> z1.P[z1], Pd’|-> op2(V,y.Pd[y]) ----------
   (10-2-v) |with(handler0(x.P[x]),op2(V,y.Pd[y]))| => op2(V,y.with(handler0(x.P[x]),Pd[y]))
   (r13-v) with(handler0(x’.P’[x’]),Pd’) => do(Pd’,x’.P’[x’])
                    with(handler0(x.P[x]),op2(V,y.Pd[y]))
op2(V,y86.with(handler0(x86.P[x86]),Pd[y86])) <-(10-2-v)-∧-(r13-v)-> do(op2(V,y.Pd[y]),xd87.P[xd87])
---> op2(V,y86.with(handler0(x86.P[x86]),Pd[y86])) =OK= op2(V,y90.do(Pd[y90],x90.P[x90])) <---

#Joinable! (Total 10 CPs)
YES
```

In the proof of Prop. 36, op_1 and op_2 were summed in op_i, so the number of critical pairs in the proof matches this output.