Second-Order Term Evaluation and Refinement Systems

— Towards a Unified Theory of Rewriting and Operational Semantics

Makoto Hamana
Kyushu Institute of Technology, Japan

Workshop in Celebration of Marcelo Fiore's 60th Birthday, Cambridge, 2-3 July, 2026.

Happy Birthday Marcelo!

Rewriting, Semantics, and Abstract Syntax with Binding

My background

- Term rewriting systems – influence of Duch school
- Semantics?
- .. The last student of Nobuo Yoneda
- Visited LFCS, Edinburgh 1999-2001 as JSPS posdoc
- -- Semantics of **higher-order rewriting**
- LICS99 paper
- Met Marcelo

Semantics ⇨ Rewriting ⇨ Verification

Second-Order Abstract
- Initial algebra semant
  [Fiore, Plotkin, Turi, LI
- Simply-typed syntax
  [Fiore, PPDP'02]
- Dependently-typed sy
- **Second-order Equat**
  [Fiore, Hur CSL'10]
- Polymorphic syntax [H. FoSSaCS'11]
- **Polymorphic algebraic theories**
  [Fiore, H., LICS'13]

- **SOL** [H. ICFP'17]
  - Second-order rewriting
  - Confluence by critical pa
  - Strong normalization by
    - Interpretation [H. MS
    - Semantic labelling [H

- **ReCheck**
  [H., Muroya, Emoto, FLOPS'24 ,TACAS'26]
  - User-defined operational semantics
  - Optimization rewriting
  - Specified by second-order rules
  - Automatic verification of contextual improvement

Problem: Verifying Contextual Improvement

- ▷ Program optimizations by rewrite rules
- ▷ **Contextual improvement** by Sands — a directed form of contextual equivalence requires that replacing a program fragment never increases evaluation cost in **any** program context
- ▷ Existing proofs are manual and for one specific operational semantics
- ▷ This work: **user-defined operational semantics** by our framework TERS (Term Evaluation and Refinement Systems).

Goal: verify contextual improvement automatically

by a rewriting method: **critical-pair analysis** and tool **ReCheck**

Example: Verification Problem in Functional Programming

Functional program

```
[]     ++ ys ⟶ ys
(x:xs) ++ ys ⟶ x : (xs ++ ys)
```

Verify equality:

```
(xs ++ ys) ++ zs = xs ++ (ys ++ zs)
```

- ▷ Proved by induction on lists
- ▷ How to automate without induction proof?

Example: Verification Problem in Functional Programming

Functional program

```
[]     ++ ys ⟶ ys
(x:xs) ++ ys ⟶ x : (xs ++ ys)
```

Verify **contextual improvement**:

```
(xs ++ ys) ++ zs ⇒ xs ++ (ys ++ zs)
```

- ▷ Real programming languages use **deterministic evaluation** mechanisms
  - Call-by-value: OCaml
  - Call-by-name, call-by-need: Haskell

- ▷ Our framework TERS (Term Evaluation and Refinement Systems)
- ▷ Second-order TERS uses second-order abstract syntax

TERS: Term Evaluation and Refinement Systems

A TERS $(\mathcal{E}, \mathcal{R})$ specifies:

- a signature,
- syntax classes (values, computations, answers, handlers, . . .),
- evaluation contexts Ectx
- evaluation rules $\mathcal{E}$
- refinement rules $\mathcal{R}$

Example: Verification Problem in Functional Programming

Functional program

```
[]     ++ ys ⟶ ys
(x:xs) ++ ys ⟶ x : (xs ++ ys)
```

Verify **contextual improvement**:

```
(xs ++ ys) ++ zs ⇒ xs ++ (ys ++ zs)
```

Example: Append as TERS

Values	$V ::= [\,] \mid V : V$
Evaluation ctxts $Ectx$	$E ::= \Box \mid E +\!\!+ t \mid V +\!\!+ E \mid E : t \mid V : E$
Evaluation rules $\mathcal{E}$	$[\,] +\!\!+ ys \longrightarrow ys$
	$(x : xs) +\!\!+ ys \longrightarrow x : (xs +\!\!+ ys)$
Refinement rules $\mathcal{R}$	$(xs +\!\!+ ys) +\!\!+ zs \Rightarrow xs +\!\!+ (ys +\!\!+ zs)$

- Rewrite relations:

Evaluation

$$\frac{(l \longrightarrow r) \in \mathcal{E} \quad E \in Ectx}{E[l\theta] \longrightarrow_{\mathcal{E}} E[r\theta]}$$

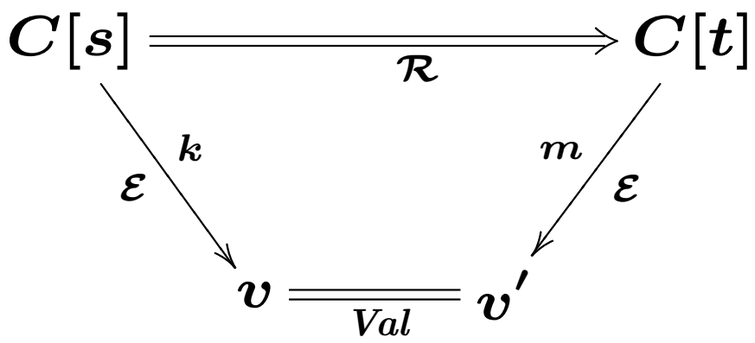
Refinement

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

- These define operational semantics and rewriting

Contextual Improvement

A set $\mathcal{R}$ of refinement rules is a **contextual improvement** w.r.t. a set $\mathcal{E}$ evaluation rules if $s \Rightarrow_{\mathcal{R}} t$



closed terms

with $k \geq m$

same observable values, with no extra evaluation steps

- **Key difficulty:** universal quantification over *all contexts* C

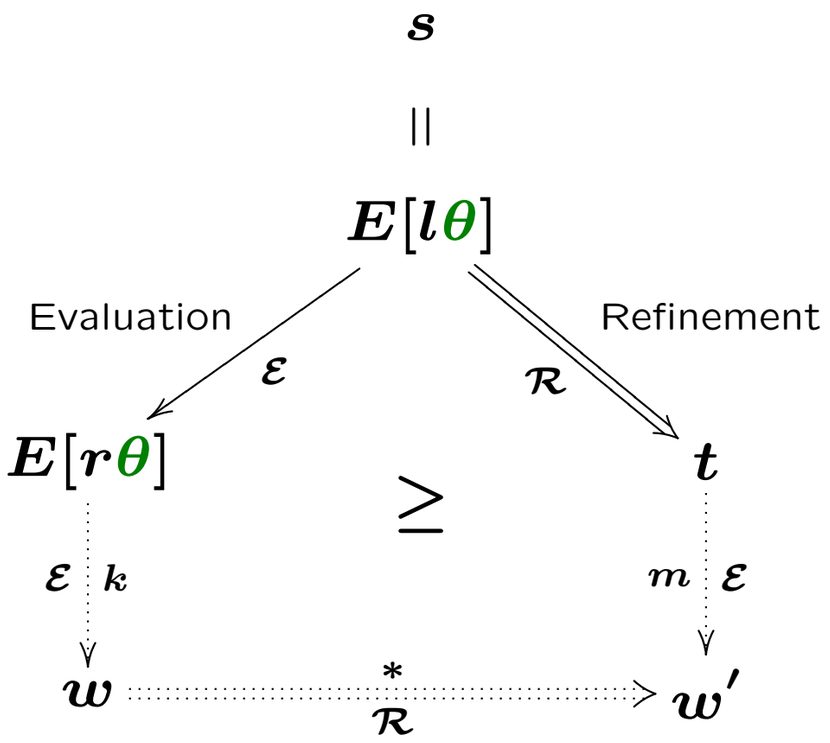
Main Theorem [Muroya&H., FLOPS'24]

If

1. $\mathcal{E}$ is deterministic (ensured by evaluation contexts)
2. $\mathcal{R}$ is value-invariant (a refinement of value is a value)
3. TERS $(\mathcal{E}, \mathcal{R})$ is **locally coherent**

then $\mathcal{R}$ is a **contextual improvement** wrt $\mathcal{E}$.

Local Coherence



with $1 + k \geq m$ $\quad$ $\mathcal{E}$: evaluation rules $\quad$ $\mathcal{R}$: refinement rules

- How to check?

Critical Pair Lemma

Lemma [FLOPS'24, Muroya&H.] A well-behaved TERS is **locally coherent** . iff all its **critical pairs** between $\mathcal{E}$ and $\mathcal{R}$ are joinable.

Workflow

TERS $\Rightarrow$ Critical pairs analysis $\Rightarrow$ Local coherence $\Rightarrow$ Contextual improvement

- **Tool ReCheck**

Critical Pairs for TERS

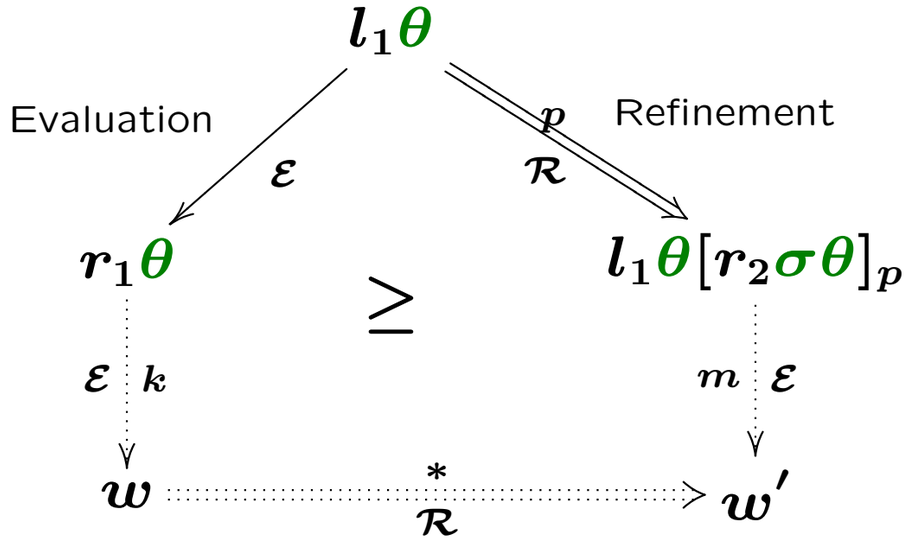
Given a TERS $(\mathcal{E}, \mathcal{R})$

$l_1 \longrightarrow r_1 \in \mathcal{E}$ Evaluation rules

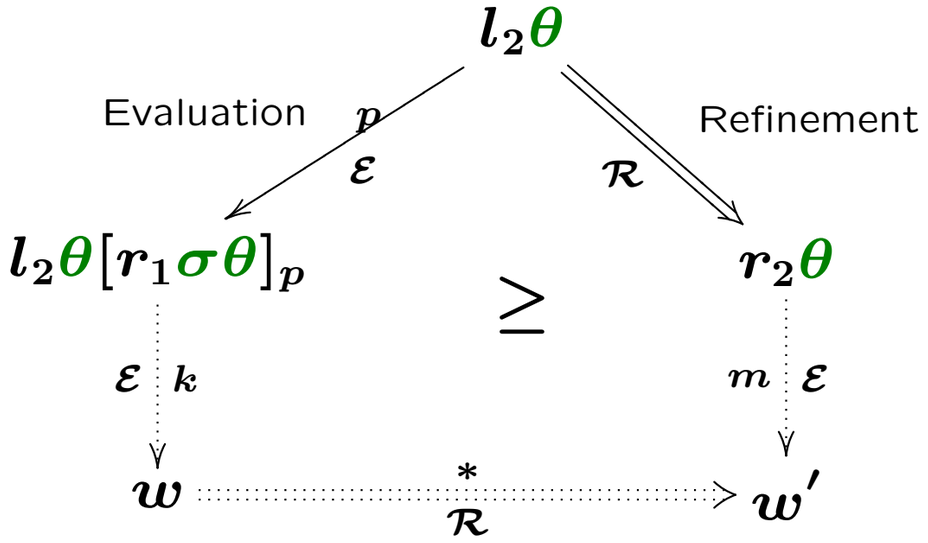
$l_2 \Rightarrow r_2 \in \mathcal{R}$ Refinement rules

- ▷ Enumerates all possible overlaps between $\mathcal{E}$ and $\mathcal{R}$ calculating most general unifiers θ
- ▷ Check their joinability with $1 + k \geq m$

Critical Pair

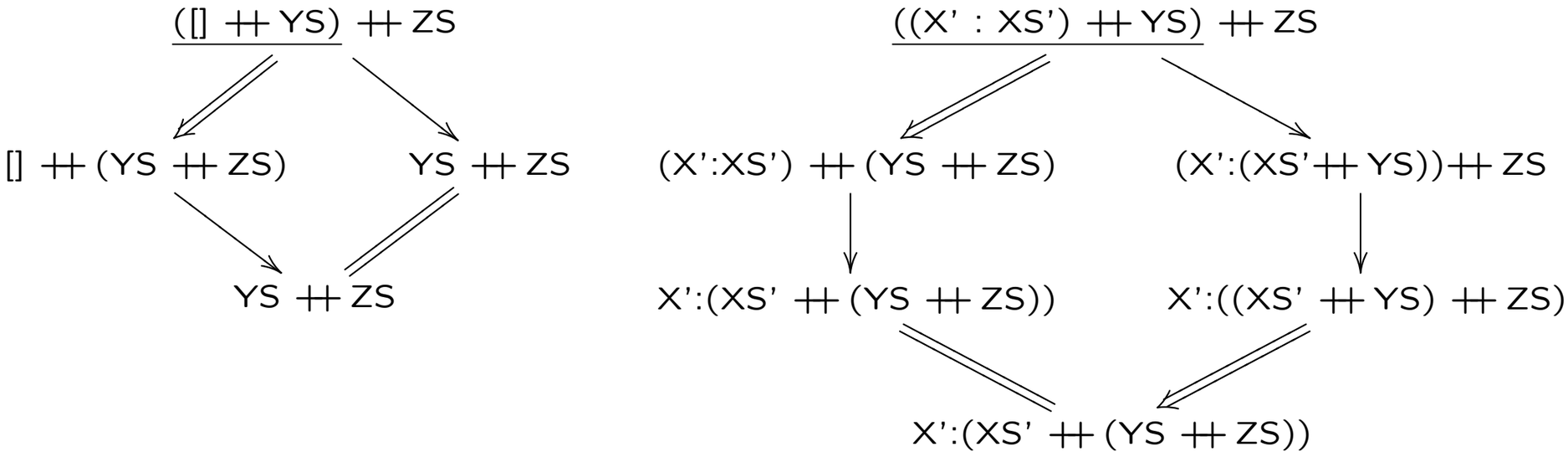


Critical Pair



Example: Append as TERS

Values	$V ::= [\,] \mid V : V$
Evaluation ctxts $Ectx$	$E ::= \Box \mid E +\!\!+ t \mid V +\!\!+ E \mid E : t \mid V : E$
Evaluation rules $\mathcal{E}$	$[\,] +\!\!+ ys \to ys$
	$(x : xs) +\!\!+ ys \to x : (xs +\!\!+ ys)$
Refinement rules $\mathcal{R}$	$(xs +\!\!+ ys) +\!\!+ zs \Rightarrow xs +\!\!+ (ys +\!\!+ zs)$



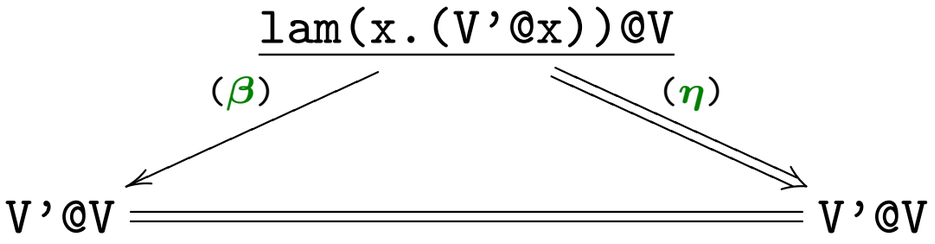
Both critical pairs are joinable, hence $\mathcal{R}$ is a **contextual improvement**.

Example: The call-by-value λ-calculus as TERS

Syntax classes

Values	$V, V' ::= x \mid \lambda(x.M)$
Evaluation contexts $\mathcal{E}ctx$	$E ::= \Box \mid E@M \mid V@E$
Evaluation rules $\mathcal{E}$	**Refinement rules** $\mathcal{R}$
$\lambda(x.M[x])@V \to M[V]$	$\lambda(x.V@x) \Rightarrow V$

- 1 Joinable critical pair $\Rightarrow$ **contextual improvement**



Example 3: A simplified computational λ-calculus λ_{ml*} as TERS

Syntax classes

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

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

Evaluation contexts $\mathcal{E}ctx$ $\quad E ::= \Box \mid \mathsf{let}(E, x.P)$

Evaluation rules $\mathcal{E}$

$\lambda(x.P[x])@V \longrightarrow P[V]$

$\mathsf{let}(\mathsf{return}(V), x.P[x]) \longrightarrow P[V]$

Refinement rules $\mathcal{R}$

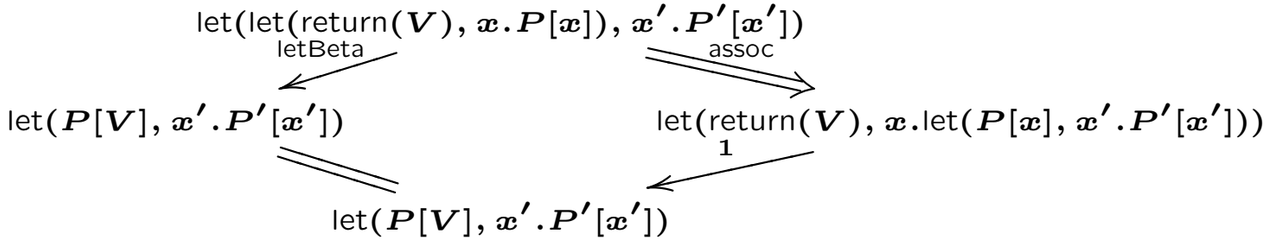
$\lambda(x.V@x) \Rightarrow V$

$\mathsf{let}(P, x.\mathsf{return}(x)) \Rightarrow P$

$\mathsf{let}(\mathsf{let}(P_1, x_1.P_2[x_1]), x_2.P_3[x_2])$

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

- 3 Joinable critical pairs $\Rightarrow$ **contextual improvement**



Example 4: Effect Handlers

Syntax classes

Functions $F ::= x \mid \mathsf{fun}(x.P)$

Values $V ::= \mathsf{true} \mid \mathsf{false} \mid F \mid H$

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

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

Evaluation contexts $Ectx$

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

Example 4: Effect Handlers

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

$\mathsf{do}(\mathsf{return}(\boldsymbol{V}), \boldsymbol{x}.\boldsymbol{P}[\boldsymbol{x}]) \longrightarrow \boldsymbol{P}[\boldsymbol{V}]$

$\mathsf{do}(\mathsf{op}_i(\boldsymbol{V}, \boldsymbol{y}.\boldsymbol{P_1}[\boldsymbol{y}]), \boldsymbol{x}.\boldsymbol{P_2}[\boldsymbol{x}]) \longrightarrow \mathsf{op}_i(\boldsymbol{V}, \boldsymbol{y}.\mathsf{do}(\boldsymbol{P_1}[\boldsymbol{y}], \boldsymbol{x}.\boldsymbol{P_2}[\boldsymbol{x}]))$

$\mathsf{if}(\mathsf{true}, \boldsymbol{P_1}, \boldsymbol{P_2}) \longrightarrow \boldsymbol{P_1} \quad \mathsf{if}(\mathsf{false}, \boldsymbol{P_1}, \boldsymbol{P_2}) \longrightarrow \boldsymbol{P_2} \quad \mathsf{fun}(\boldsymbol{x}.\boldsymbol{P}[\boldsymbol{x}])@\boldsymbol{V} \longrightarrow \boldsymbol{P}[\boldsymbol{V}]$

$\mathsf{with\_handle}(\boldsymbol{h_1}, \mathsf{return}(\boldsymbol{V})) \longrightarrow \boldsymbol{P}[\boldsymbol{V}]$

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

$\mathsf{with\_handle}(\boldsymbol{h_1}, \mathsf{op}_1(\boldsymbol{V}, \boldsymbol{y}.\boldsymbol{P'}[\boldsymbol{y}])) \longrightarrow \boldsymbol{P_1}[\boldsymbol{V}, \mathsf{fun}(\boldsymbol{y}.\boldsymbol{P'}[\boldsymbol{y}])]$

$\mathsf{with\_handle}(\boldsymbol{h_1}, \mathsf{op}_2(\boldsymbol{V}, \boldsymbol{y}.\boldsymbol{P'}[\boldsymbol{y}])) \longrightarrow \mathsf{op}_2(\boldsymbol{V}, \boldsymbol{y}.\mathsf{with\_handle}(\boldsymbol{h_1}, \boldsymbol{P'}[\boldsymbol{y}]))$

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

$\mathsf{with\_handle}(\boldsymbol{h_0}, \mathsf{return}(\boldsymbol{V})) \longrightarrow \boldsymbol{P}[\boldsymbol{V}]$

$\mathsf{with\_handle}(\boldsymbol{h_0}, \mathsf{op}_i(\boldsymbol{V}, \boldsymbol{y}.\boldsymbol{P'}[\boldsymbol{y}])) \longrightarrow \mathsf{op}_i(\boldsymbol{V}, \boldsymbol{y}.\mathsf{with\_handle}(\boldsymbol{h_0}, \boldsymbol{P'}[\boldsymbol{y}]))$

Example 4: Effect Handlers

Refinement rules $\mathcal{R}$

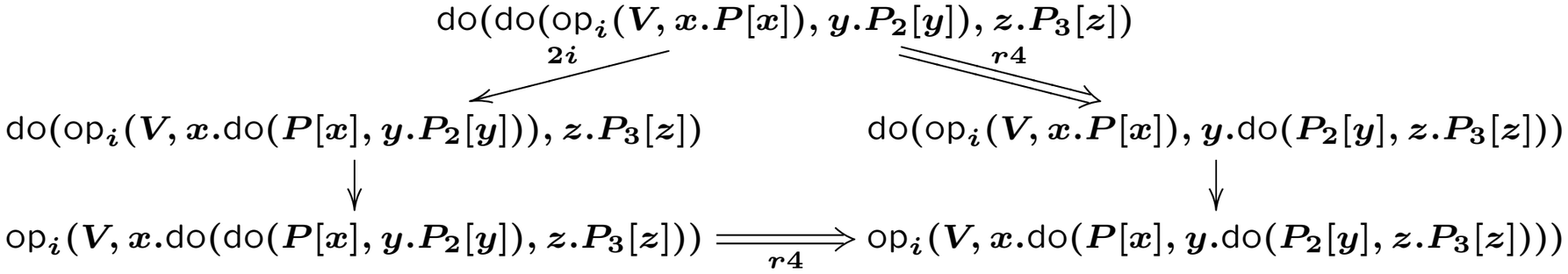
$$\mathsf{do}(\boldsymbol{P}, \boldsymbol{x}.\mathsf{return}(\boldsymbol{x})) \Rightarrow \boldsymbol{P}$$

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

$$\mathsf{fun}(\boldsymbol{x}.\boldsymbol{F}@\boldsymbol{x}) \Rightarrow \boldsymbol{F}$$

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

- 10 Joinable critical pairs $\Rightarrow$ **contextual improvement**



Example 2: Lazy program as TERS

Values Val $\quad \boldsymbol{V ::= n \mid V : t}$

Evaluation contexts Ectx $\quad \boldsymbol{E ::= \Box \mid \mathsf{ns}(E) \mid E : t}$

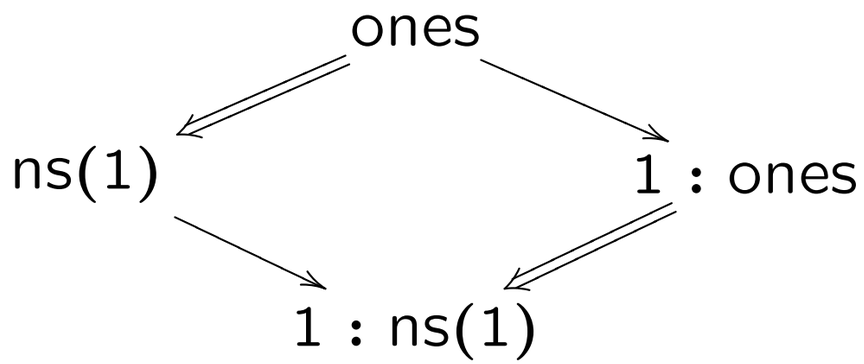
Evaluation rules $\mathcal{E}$

$\mathsf{ones} \longrightarrow 1 : \mathsf{ones}$

$\mathsf{ns}(\boldsymbol{n}) \longrightarrow \boldsymbol{n} : \mathsf{ns}(\boldsymbol{n})$

Refinement rule $\mathcal{R}$

$\mathsf{ones} \Rightarrow \mathsf{ns}(1)$



- $\mathcal{R}$ is a contextual improvement

Summary

- Secon-order TERS for evaluation and refinement
- **ReCheck: contextual improvement verifier** by critical pair analysis
- Future Directions
  - Unified theory of operational semantics and rewriting
  - Automatic verifier of Haskell's rewrite rules
  - Inductive equational proving without termination
  - Computation for Quotient algebraic datatypes
  - Other costs for evaluation:
    Weighted rewriting [Avanzini, Yamada 2025],
    Quantitative rewriting [POPL'23]
  - Relationship to Quantitative equational logic [Mardare, Pangaren and Plotkin, LICS'16]