Extensional rewriting with sums

Sam Lindley

Laboratory for Foundations of Computer Science,
School of Informatics, The University of Edinburgh
Sam.Lindley@ed.ac.uk*

Abstract. Inspired by recent work on normalisation by evaluation for sums, we propose a normalising and confluent extensional rewriting theory for the simply-typed λ-calculus extended with sum types. As a corollary of confluence we obtain decidability for the extensional equational theory of simply-typed λ-calculus extended with sum types. Unlike previous decidability results, which rely on advanced rewriting techniques or advanced category theory, we only use standard techniques.

1 Introduction

It is easy to add sum types to the equational theory of the simply-typed λ-calculus, in the presence of η-rules, or to add sum types to the rewriting theory of simply-typed λ-calculus, in the absence of η-rules. However, adding sum types to the rewriting theory is difficult in the presence of η-rules. Existing rewriting theories, with the exception of Ghani's [5], are either incomplete with respect to the equational theory or non-confluent. Quoting Altenkirch et al [1], Ghani's work involves 'intricate rewriting techniques whose details are daunting'. Our aim is to introduce a straightforward rewriting theory using standard techniques.

The essential reason why the problem with confluence arises is that reordering independent nested cases does not change the semantics of a term. For instance, let = be equivalence in the equational theory, writing $\delta(p, x_1.n_2, x_2.n_2)$ for $\mathsf{case}\ p\ \mathsf{of}\ \mathsf{in}_1(x_1) \Rightarrow n_2 \mid \mathsf{in}_2(x_2) \Rightarrow n_2$, then

$$\begin{aligned}&\delta(p_1, x_1.\delta(p_2, y_1.n_1, y_2.n_2), x_2.\delta(p_2, y_1.m_1, y_2.m_2))\\ &\quad = \delta(p_2, y_1.\delta(p_1, x_1.n_1, x_2.m_1), y_2.\delta(p_1, x_1.n_2, x_2.m_2))\end{aligned}$$

where $x_1, x_2, y_1, y_2, m_1, m_2, n_1, n_2, p_1, p_2$ are distinct object variables. The structure of the terms on each side of the equation is identical, so it is not possible to capture the equivalence with a rewrite rule.

This article explores several solutions to the case ordering problem, all suggested by the work of Altenkirch et al [1] and Balat et al [2] on normalisation by evaluation for the simply-typed λ-calculus extended with sums. The goal of normalisation by evaluation [4] is to find a unique normal form with respect to the equational theory. In contrast, we shall be interested in normal forms with respect to a rewriting theory.

In fact, the case ordering problem also manifests itself in the equational setting. In the example above, either the left hand side or the right hand side

* Supported by EPSRC grant number EP/D046769/1

of the equivalence should be a normal form. But the terms are structurally identical so some non-structural property must be used to define normal forms. One possibility is to define an ordering on terms via an ordering on variable names. The ordering on terms can then be used to assign an ordering to nested cases. Such an ordering is undesirable as it requires us to dispense with α-conversion.

Altenkirch et al solve the problem by adding a new construct to the object language — a parallel case that simultaneously eliminates a set of sums. Using the extended language both sides of the equivalence are represented by the same parallel case. A big advantage of this approach is that it leads to a syntax which much more closely captures the semantics of the calculus. The main disadvantage is that it drastically increases the complexity of the machinery used by the syntax of the language. In Altenkirch et al's presentation functions appear in the syntax. These functions can be represented more concretely using sets or lists, but the resulting syntax is still significantly more complex than the standard one.

Balat et al [2] build on Altenkirch et al's work. Instead of adding parallel cases, they define a congruence over terms which contains the equivalence given in the example above as a special case. They identify normal forms up to this congruence, leading to a rather elegant presentation.

We adopt an extension $\sim$ of Balat et al's congruence, and perform rewriting modulo $\sim$. Sect. 2 introduces the equational theory $\lambda^{\to\times+}$ of simply-typed lambda-calculus extended with products and sums, and decomposes the general η axiom for sums into a number of simpler axioms. Sect. 3 describes a non-local rewriting theory that generates the equational theory. Sect. 4 gives a reducibility proof of strong normalisation for a fragment of the rewriting theory following the approach of Lindley and Stark [8, Chaper 3][9]. Sect. 5 uses strong normalisation results for fragments of the rewriting theory to prove weak normalisation and confluence modulo $\sim$ for the full rewriting theory, and hence decidability for the equational theory. Sect. 6 describes three variations of the rewriting theory. Sect. 7 concludes.

2 The object language

The simply typed lambda calculus extended with products and sums is standard [5]. We write $\lambda^{\to\times+}$ for the equational theory.

$$\text{(Types)} \qquad A, B ::= O \mid A \to B \mid A \times B \mid A + B$$

Types are constructed from a base type O, functions $A \to B$ from type A to type B, products $A \times B$ of types A and B, and sums of types A and B. We omit the unit and empty types, but restoring them does not radically change our proofs (though the empty type requires a little more care in the handling of typing

contexts).

$$\text{(Terms)} \qquad m, n, p ::= x \mid \lambda x.m \mid m\,n \mid \langle m, n\rangle \mid \pi_1(m) \mid \pi_2(m) \mid \iota_1(m) \mid \iota_2(m) \mid \delta(m, x_1.n_1, x_2.n_2)$$

Terms are constructed from variables x, lambda abstractions $\lambda x.m$, applications $m\,n$, pairs $\langle m, n\rangle$, projections $\pi_i(m)$, injections $\iota_i(m)$ and cases $\delta(m, x_1.n_1, x_2.n_2)$. Terms are identified up to α-conversion.

The free $fv(m)$ and bound variables $bv(m)$ are defined in the usual way. We write $m[x := n]$ for the capture-avoiding substition of n for x in m, and $m[x_1 := n_1, \ldots, x_k := n_k]$ for the simultaneous capture-avoiding substition of n_i for x_i in m $(1 \leqslant i \leqslant k)$. We write $size(m)$ for the size of the term m. The typing rules are standard. Each type constructor has an introduction rule and an elimination rule.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x.m : A \to B} \qquad \frac{\Gamma \vdash m : A \to B \quad \Gamma \vdash n : A}{\Gamma \vdash m\,n : B}$$

$$\frac{\Gamma \vdash m : A \quad \Gamma \vdash n : B}{\Gamma \vdash \langle m, n\rangle : A \times B} \qquad \frac{\Gamma \vdash m : A_1 \times A_2}{\Gamma \vdash \pi_i(m) : A_i}\ i \in \{1, 2\}$$

$$\frac{\Gamma \vdash m : A_i}{\Gamma \vdash \iota_i(m) : A_1 + A_2}\ i \in \{1, 2\}$$

$$\frac{\Gamma \vdash m : A_1 + A_2 \quad \Gamma, x_i : A_i \vdash n_i : B \quad i \in \{1, 2\}}{\Gamma \vdash \delta(m, x_1.n_1, x_2.n_2) : B}$$

Axioms The axioms for $\lambda^{\to\times+}$ consist of a β-axiom and an η-axiom for each type constructor.

$$(\to.\beta) \qquad (\lambda x.m)\,n = m[x := n]$$
$$(\times.\beta_i) \qquad \pi_i(\langle m_1, m_2\rangle) = m_i, \quad i \in \{1, 2\}$$
$$(+.\beta_i) \qquad \delta(\iota_i(m), x_1.n_1, x_2.n_2) = n_i[x_i := m], \quad i \in \{1, 2\}$$

$$(\to.\eta) \qquad m = \lambda x.m\,x, \quad x \notin fv(m)$$
$$(\times.\eta) \qquad m = \langle \pi_1(m), \pi_2(m)\rangle$$
$$(+.\eta^\dagger) \qquad n[x := p] = \delta(p, x_1.n[x := \iota_1(x_1)], x_2.n[x := \iota_2(x_2)])$$

The equation $m = n$ is shorthand for the equality judgement $\Gamma \vdash m = n : A$ where $\Gamma \vdash m : A$ and $\Gamma \vdash n : A$. The equational theory is given by the least (typed) congruence satisfying the axioms.

Alternative axioms The generalised η-axiom for sums $+.\eta^\dagger$ is non-local and it is not at all obvious how it might give rise to a confluent rewriting system. In particular, note that substitutions appear both on the left and the right hand

side of the axiom. We break $+.\eta^\dagger$ down into a number of simpler axioms.

$$(+.\eta) \quad p = \delta(p, x_1.\iota_1(x_1), x_2.\iota_2(x_2))$$

$$(\textit{move-case}) \quad F[\delta(p, x_1.n_1, x_2.n_2)] = \delta(p, x_1.F[n_1], x_2.F[n_2]),$$
$$x_1, x_2 \notin \mathit{fv}(F[\,]) \text{ and } \mathit{bv}(F[\,]) \cap \mathit{fv}(p) = \emptyset$$

$$(\textit{repeated-guard})$$
$$\delta(p, x_1.\delta(p, y_1.n_1, y_2.n_2), x_2.\delta(p, z_1.p_1, z_2.p_2))$$
$$= \delta(p, x_1.n_1[y_1 := x_1], x_2.p_2[z_2 := x_2]), \quad x_1, x_2 \notin \mathit{fv}(p)$$

$$(\textit{redundant-guard}) \quad \delta(p, x_1.n, x_2.n) = n, \quad x_1, x_2 \notin \mathit{fv}(n)$$

The local η axiom for sums $+.\eta$ is a special case of $+.\eta^\dagger$ in which n is just x. The *move-case* axiom is a generalisation of the usual commuting conversions for $\lambda^{\to\times+}$ [6,11]. As well as allowing cases to move across *elimination frames* ($F_1[\,]$), *move-case* also allows them to be moved across *neutral frames* ($F_2[\,]$), *lambda frames* ($F_3[\,]$) and *continuation frames* ($F_4[\,]$).

$$(\text{Frames}) \qquad F[\,] ::= F_1[\,] \mid F_2[\,] \mid F_3[\,] \mid F_4[\,]$$

$$F_1[\,] ::= [\,]\,n \mid \pi_1([\,]) \mid \pi_2([\,]) \mid \delta([\,], x_1.n_1, x_2.n_2)$$
$$F_2[\,] ::= m\,[\,] \mid \langle [\,], n\rangle \mid \langle m, [\,]\rangle \mid \iota_1([\,]) \mid \iota_2([\,])$$
$$F_3[\,] ::= \lambda x.[\,]$$
$$F_4[\,] ::= \delta(p, x_1.[\,], x_2.n_2) \mid \delta(p, x_1.n_1, x_2.[\,])$$

We write *move-case*$_i$ for the restriction of *move-case* to frames of the form F_i. Following Altenkirch et al, we use the word *guard* to refer to the first argument of a case. The axiom *repeated-guard* allows guards to be copied or deleted. The axiom *redundant-guard* is a special case of $+.\eta^\dagger$ in which x does not occur free in n.

Proposition 1. *Replacing the axiom $+.\eta^\dagger$ with the alternative axioms $+.\eta$, move-case, repeated-guard, redundant-guard yields the same equational theory.*

Proof. (sketch)
New axioms are sound:

- $+.\eta$ and *redundant-guard* are instances of $+.\eta^\dagger$
- *move-case*, *repeated-guard*: apply $+.\eta^\dagger$ from left to right using p as the substituted term, eliminate resulting $+.\beta_i$ redexes, then α-convert

New axioms are complete:

- η expand all instances of p in n
- use *move-case* to hoist all instances of p to the top
- use *repeated-guard* and *redundant-guard* to get rid of the multiple copies of p
- use *redundant-guard* for the case where $x \notin \mathit{fv}(n)$

The axioms *move-case*, *repeated-guard* and *redundant-guard*, are implicit in previous work on normalisation by evaluation with sums [1,2]. To the author's knowledge, they have not previously been used as the basis for a rewriting calculus.

3 A rewriting theory

As a first attempt at a rewriting theory consider defining a rewrite rule for each axiom by orienting from left to right (with the usual restrictions for η-expansion). Unfortunately the resulting theory has infinite reduction sequences arising from *move-case*$_4$. For instance, the following reductions can be applied indefinitely as m appears as a subterm of n.

$$\begin{aligned} &m = \delta(p, x.n, x.\delta(p, x.n, x.n)) \\ &\longrightarrow_{\textit{move-case}_4} \delta(p, x.\delta(p, x.n, x.n), x.\delta(p, x.n, x.n)) \\ &\longrightarrow_{\textit{move-case}_4} \delta(p, x.m, x.m) = n \end{aligned}$$

Ohta and Hasegawa [10] face a similar problem for a linear lambda calculus. Their solution is to separate the axioms of their equational theory into a reduction relation and an equivalence relation, and use Huet's technique for proving confluence of the reduction relation modulo the equivalence relation [7]. Balat et al use an equivalence for defining normal forms and implementing normalisation by evaluation with sums. Their equivalence is the least congruence satisfying the *move-case*$_4$ and *redundant-guard* axioms. We introduce a congruence that also includes the *repeated-guard* axiom.

Definition 2. *The relation $\sim$ is the least congruence satisfying the axioms move-case$_4$, repeated-guard and redundant-guard.*

Deciding equivalence modulo $\sim$ is straightforward. First we define some auxiliary functions.

Definition 3.

$$\textit{Guards}(m) = \begin{cases} p \cup \textit{Guards}(x_1.n_1) \cup \textit{Guards}(x_2.n_2), & \textit{if } m = \delta(p, x_1.n_1, x_2.n_2) \\ \emptyset, & \textit{otherwise} \end{cases}$$

$$\textit{Guards}(x.n) = \{m \in \textit{Guards}(n) \mid x \notin \textit{fv}(m)\}$$

$$\textit{Paths}(gs) = \{\rho \mid \rho \in \{1,2\}^{gs}\} \quad \nu(ps) = \{p^{x_1}_{x_2} \mid p \in ps_{/\sim} \textit{ and } x_1, x_2 \textit{ fresh}\}$$

$$\textit{Tail}_{\rho[p^{x_1}_{x_2} \mapsto i]}(\delta(p', x'_1.n'_1, x'_2.n'_2)) = \textit{Tail}_{\rho[p^{x_1}_{x_2} \mapsto i]}(n_i[x'_i := x_i]), \ \textit{if } p \sim p'$$

$$\textit{Tail}_\rho(m) = m$$

The function $Guards(m)$ gives the set of independent *guards* at the top-level of m. The definition of *Guards* is the same as that used by Balat et al. If $ps = Guards(m)$, then $Paths(\nu(ps))$ represents the set of possible *paths* through m dictated by ps. Given a path ρ through a term m, the subterm of m at the end of that path, the *tail* of ρ, is given by $Tail_\rho(m)$. We write $ps_{/\sim}$ for the quotient set of ps by $\sim$.

Proposition 4.

$$\begin{aligned} m_1 \sim m_2 \quad &\iff \\ &\forall \rho \in Paths(\nu(Guards(m_1) \cup Guards(m_2))). Tail_\rho(m_1) \sim Tail_\rho(m_2) \end{aligned}$$

To decide whether $m_1 \sim m_2$: if one of m_1, m_2 is a case, then use Prop. 4; otherwise compare the top-level constructors and if equal recurse on the immediate subterms of m_1, m_2. Having defined the decidable equivalence $\sim$, we now present the rewrite rules. The β- and η-rules are standard.

β*-rules*

$$\begin{array}{lrcl}
(\to.\beta) & \lambda x.m\,n & \longrightarrow & m[x := n] \\
(\times.\beta_1) & \pi_1(\langle m_1, m_2 \rangle) & \longrightarrow & m_1 \\
(\times.\beta_2) & \pi_2(\langle m_1, m_2 \rangle) & \longrightarrow & m_2 \\
(+.\beta_1) & \delta(\iota_1(m), x_1.n_1, x_2.n_2) & \longrightarrow & n_1[x_1 := m] \\
(+.\beta_2) & \delta(\iota_2(m), x_1.n_1, x_2.n_2) & \longrightarrow & n_2[x_2 := m]
\end{array}$$

η*-rules* The η-rules are type-directed expansions.

$$\begin{array}{lrcl}
(\to.\eta) & m^{A \to B} & \longrightarrow & \lambda x.m\,x, \qquad \text{if } x \notin fv(m) \\
(\times.\eta) & m^{A \times B} & \longrightarrow & \langle \pi_1(m), \pi_2(m) \rangle \\
(+.\eta) & m^{A+B} & \longrightarrow & \delta(m, x_1.\iota_1(x_1), x_2.\iota_2(x_2))
\end{array}$$

The annotation m^A means m has type A. The η-rules are applicable only if expansion does not create a new redex. More precisely, only variables, applications and projections (*pure neutral terms*) can be η-expanded, and only in a non-elimination frame.

In order to instantiate the *move-case* axiom as a rewrite rule we read it from left to right. This corresponds to *hoisting* a case over a frame. Of course, we do not generally need to allow hoisting over continuation frames, as this is captured by $\sim$. However, for confluence it is necessary to allow hoisting over several continuation frames followed by a non-continuation frame.

Frames and contexts

$$
\begin{array}{lrl}
\text{(Hoisting frames)} & \mathsf{H}[\,] & ::= \mathsf{F}_1[\,] \mid \mathsf{F}_2[\,] \mid \mathsf{F}_3[\,] \\
\text{(Discriminator contexts)} & \mathsf{D}[\,] & ::= [\,] \mid \delta(p, x_1.\mathsf{D}[\,], x_2.n_2) \\
 & & \mid \delta(p, x_1.n_1, x_2.\mathsf{D}[\,]) \\
\text{(Hoisting contexts)} & \mathsf{HD}[\,] & ::= \mathsf{H}[\mathsf{D}[\,]]
\end{array}
$$

γ-rules We refer to reduction rules that arise from the *move-case*-, *repeated-guard*- and *redundant-guard*-axioms as *γ-rules*.

$$
\begin{array}{l}
(\textit{hoist-case}) \\
\quad \mathsf{HD}[\delta(p, x_1.n_1, x_2.n_2)] \longrightarrow \delta(p, x_1.\mathsf{HD}[n_1], x_2.\mathsf{HD}[n_2]), \\
\quad\quad x_1, x_2 \notin \mathit{fv}(\mathsf{HD}) \text{ and } \mathit{bv}(\mathsf{HD}) \cap \mathit{fv}(p) = \emptyset
\end{array}
$$

The *hoist-case*-rule is obtained from the *move-case* axiom. It is a generalisation of the usual commuting conversions. Note that continuation frames are not hoisting frames, as naïvely hoisting over continuation frames would lead to non-termination. However, the equivalence $\sim$ includes the possibility of moving cases over continuation frames, and the *hoist-case*-rule does allow a case to be hoisted over a hoisting frame from inside a discrimination context.

The discrimination context is necessary in the *hoist-case*-rule because it is only sound to hoist a case over a lambda abstraction if the lambda-bound variable does not occur free in the guard. If hoisting from within a discrimination context is disallowed, then some cases become blocked from being hoisted outside the lambda by outer cases that depend on the bound variable. For instance, suppose D is restricted to be the empty context $[\,]$, then the terms

$$\lambda x.\delta(x, x_1.\delta(z, y_1.y_1, y_2.y_2), x_2.x_2)$$

and

$$\delta(z, y_1.\lambda x.\delta(x, x_1.y_1, x_2.x_2), y_2.\lambda x.\delta(x, x_1.y_2, x_2.x_2))$$

become distinct normal forms, despite the fact that these terms are identified in the equational theory.

Remark For confluence it is not necessary to have a discrimination context in the *hoist-case*$_1$- and *hoist-case*$_2$-rules, but here we gave the single general *hoist-case*-rule for the sake of uniformity.

Definition 5 (Reduction relations).

$$
\begin{array}{lll}
\longrightarrow_{\beta} & = & \textit{the compatible closure of the } \beta\textit{-rules} \\
\longrightarrow_{\eta} & = & \textit{the restricted compatible closure of the } \eta\textit{-rules} \\
\longrightarrow_{\gamma} & = & \textit{the compatible closure of the } \gamma\textit{-rules} \\
\longrightarrow_{\gamma_E} & = & \textit{the compatible closure of hoist-case with } \mathsf{HD} \textit{ restricted to } \mathsf{F}_1 \\
 & & \textit{(i.e. the standard commuting conversion reduction relation)} \\
\longrightarrow_{\gamma'} & = & \longrightarrow_{\gamma} \setminus \longrightarrow_{\gamma_E} \\
\longrightarrow_{c} & = & \longrightarrow_{\beta} \cup \longrightarrow_{\eta} \cup \longrightarrow_{\gamma_E} \\
\longrightarrow & = & \longrightarrow_{\beta} \cup \longrightarrow_{\eta} \cup \longrightarrow_{\gamma}
\end{array}
$$

4 Strong normalisation for $\beta\eta\gamma_E$-reduction

Strong normalisation is standard for $\beta\eta\gamma_E$-reduction [5,11]. In this section we present an adaptation of the strong normalisation proof given in the author's thesis [8, Chapter 3]. Our use of frame stacks alleviates difficulties with γ_E-reduction, and leads to a significantly simpler proof than Prawitz's original one [11].

Definition 6. *A term* m *is strongly normalising with respect to a reduction relation* R*, or* R-SN*, if all* R*-reduction sequences starting from* m *are finite. A reduction relation* R *is* strongly normalising*, or* SN*, if all terms* m *are* R*-SN. If* m *is* R*-SN, then we write* $max_R(m)$ *for the maximum length of a reduction sequence starting from* m*.*

Definition 7 (frame stacks).

$$
\begin{array}{lrl}
(\textit{elimination frames}) & E ::= & F_1 \\
(\textit{frame stacks}) & S ::= & Id \mid S \circ E \\
(\textit{stack length}) & |Id| = & 0 \\
 & |S \circ E| = & |S| + 1 \\
(\textit{plugging}) & Id[m] = & m \\
 & (S \circ E)[m] = & S[(E[m])]
\end{array}
$$

Following Girard et al [6] we assume variables are annotated with types (it is straightforward, albeit somewhat tedious, to adapt the proof to use local typing contexts instead). We write $A \multimap B$ for the type of frame stack S, if $S[m] : B$ for all terms $m : A$.

Definition 8 (frame stack reduction).

$$S \longrightarrow_c S' \quad \stackrel{def}{\Longleftrightarrow} \quad \forall m. S[m] \longrightarrow_c S'[m]$$

A frame stack S is c-*strongly normalising* if all c-reduction sequences starting from S are finite.

Lemma 9.

1. $S \longrightarrow_c S'$ *iff* $S \neq Id$ *and* $S[x] \longrightarrow_c S'[x]$.
2. *If* $S \longrightarrow_c S'$*, for frame stacks* S, S'*, then* $|S'| \leqslant |S|$.
3. *If there exists* m *such that* $S[m]$ *is* c*-SN, then* $S[x]$ *is* c*-SN.*

Proof. Induction on the structure of S.

Definition 10 (reducibility).

– *Id is reducible.*

- $S \circ [\,]\, n : (A \to B) \multimap C$ *is reducible if* S *and* n *are reducible.*
- $S \circ \pi_i([\,]) : (A \times B) \multimap C$ *is reducible if* S *is reducible.*
- $S : (A + B) \multimap C$ *is reducible if* $S[\iota_1(m)]$ *is* c*-SN for all reducible* $m : A$*, and* $S[\iota_2(n)]$ *is* c*-SN for all reducible* $n : B$*.*
- $m{:}A$ *is reducible if* $S[m]$ *is* c*-SN for all reducible* $S : A \multimap C$*.*

Lemma 11. *If* $m : A$ *is reducible then* m *is* c*-SN.*

Proof. Follows immediately from reducibility of *Id* and the definition of reducibility on terms.

Lemma 12. $x : A$ *is reducible.*

Proof. By induction on A using Lemma 9 and Lemma 11.

Corollary 13. *If* $S : A \multimap C$ *is reducible then* S *is* c*-SN.*

Each type constructor has an associated β-rule. Each β-rule gives rise to an SN-closure property.

Lemma 14 (SN-closure).

$\to$ *If* $S[m[x := n]]$ *and* n *are* c*-SN then* $S[(\lambda x.m)\, n]$ *is* c*-SN.*
$\times.1$ *If* $S[m]$ *and* n *are* c*-SN then* $S[\pi_1(\langle m, n\rangle)]$ *is* c*-SN.*
$\times.2$ *If* $S[n]$ *and* m *are* c*-SN then* $S[\pi_2(\langle m, n\rangle)]$ *is* c*-SN.*
$+.1$ *If* $S[n_1[x_1 := m]]$*,* $S[n_2]$ *and* m *are* c*-SN then* $S[\delta(\iota_1(m), x_1.n_1, x_2.n_2)]$ *is* c*-SN.*
$+.2$ *If* $S[n_2[x_2 := m]]$*,* $S[n_1]$ *and* m *are* c*-SN then* $S[\delta(\iota_2(m), x_1.n_1, x_2.n_2)]$ *is* c*-SN.*

Proof.

$\to, \times.1, \times.2$: By induction on $max_c(S) + max_c(m) + max_c(n)$.
$+.1$: By induction on $|S| + max_c(S[n_1[x_1 := m]]) + max(S[n_2]) + max_c(m)$.
$+.2$: By induction on $|S| + max_c(S[n_2[x_2 := m]]) + max_c(S[n_1]) + max_c(m)$.

Now we obtain reducibility-closure properties for each type constructor.

Lemma 15 (reducibility-closure).

$\to$ *If* $m[x := n]$ *is reducible for all reducible* n*, then* $\lambda x.m$ *is reducible.*
$\times$ *If* m, n *are reducible, then* $\langle m, n\rangle$ *is reducible.*
$+$ *If* m *is reducible,* $n_1[x_1 := l]$ *is reducible for all reducible* l*, and* $n_2[x_2 := p]$ *is reducible for all reducible* p*, then* $\delta(m, x_1.n_1, x_2.n_2)$ *is reducible.*

Proof. Each property follows from the corresponding part of Lemma 14 using Lemma 11 and Corollary 13.

Theorem 16. *Let* m *be any term. Suppose* $x_1 : A_1, \ldots, x_k : A_k$ *includes all the free variables of* m*. If* $p_1 : A_1, \ldots, p_k : A_n$ *are reducible then* $m[x_1{:=}p_1, \ldots, x_k{:=} p_k]$ *is reducible.*

Proof. By induction on the structure of terms using Lemma 15.

Theorem 17 (strong normalisation). *All terms are* c*-SN.*

Proof. Let m be a term with free variables $x_1, \ldots, x_k$. By Lemma 12, $x_1, \ldots, x_k$ are reducible. Hence, by Thm. 16, m is c-SN.

5 Weak normalisation and confluence

It is straightforward to prove that γ-reduction is strongly normalising.

Lemma 18. *If* $D[x], m$ *are* γ*-SN, then* $D[m]$ *is* γ*-SN.*

Proof. By induction on $max_\gamma(D[x]) + max_\gamma(m)$.

Lemma 19. *If* p, n_1, n_2 *are* γ*-SN, then* $\delta(p, x_1.n_1, x_2.n_2)$ *is* γ*-SN.*

Proof. By induction on $\langle max_\gamma(p), size(p), max_\gamma(n_1) + max_\gamma(n_2)\rangle$. The only interesting case is

$$\begin{aligned}&\delta(D[\delta(p, x_1.p_1, x_2.p_2)], y_1.n_1, y_2.n_2)\\ &\quad\longrightarrow_\gamma \delta(p, x_1.D[\delta(p_1, y_1.n_1, y_2.n_2)], x_2.D[\delta(p_2, y_1.n_1, y_2.n_2)])\end{aligned}$$

By the induction hypothesis, $\delta(p_1, y_1.n_1, y_2.n_2)$ and $\delta(p_2, y_1.n_1, y_2.n_2)$ are both γ-SN. Then by Lemma 18 and the induction hypothesis

$$\delta(p, x_1.D[\delta(p_1, y_1.n_1, y_2.n_2)], x_2.D[\delta(p_2, y_1.n_1, y_2.n_2)])$$

is γ-SN.

Lemma 20.

1. *If* m, n *are* γ*-SN then* $m\,n$ *is* γ*-SN.*
2. *If* m, n *are* γ*-SN then* $\langle m, n\rangle$ *is* γ*-SN.*
3. *If* m *is* γ*-SN then* $\lambda x.m$ *is* γ*-SN.*
4. *If* m *is* γ*-SN then* $\pi_i(m)$ *is* γ*-SN.*
5. *If* m *is* γ*-SN then* $\iota_i(m)$ *is* γ*-SN.*

Proof.

1-2 By induction on $\langle max_\gamma(m) + max_\gamma(n), size(m) + size(n))\rangle$.
3-5 By induction on $\langle max_\gamma(m), size(m)\rangle$.

Theorem 21. γ *is strongly normalising.*

We now obtain weak normalisation for $\beta\eta\gamma$-reduction. The key observation is that γ-reduction following $\beta\eta\gamma_E$-normalisation cannot introduce new $\beta\eta$-redexes.

Lemma 22. *If* m *is in* $\beta\eta\gamma_E$*-normal form and* $m \longrightarrow_\gamma^* m'$*, then* m' *is in* $\beta\eta$*-normal form.*

Lemma 22 is easily proved by a straightforward syntactic analysis of the structure of the term m'. The details are omitted due to lack of space.

Theorem 23. $\longrightarrow$ *is weakly normalising.*

Proof. To normalise a term of m, first reduce to a $\beta\eta\gamma_E$-normal form m', then reduce m' to γ-normal form m''. By Lemma 22, m'' must be a $\beta\eta\gamma$-normal form.

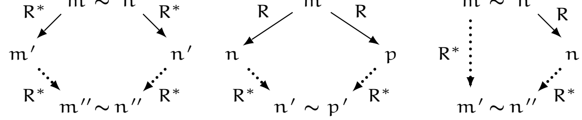
We could obtain confluence by appealing to correctness of normalisation by evaluation for sums [1]. Instead, we give a direct proof of confluence modulo $\sim$ using the strong normalisation results for $\longrightarrow_c$ and $\longrightarrow_\gamma$.

We write R^* for the transitive reflexive closure of the relation R.

Definition 24. *A reduction relation* R *is:*

- *confluent modulo* $\sim$ *iff for all* m, n, m', n' *with* $m \sim n$, $m \longrightarrow_R^* m'$ *and* $n \longrightarrow_R^* n'$, *there exist* m'', n'' *with* $m' \longrightarrow_R^* m''$, $n' \longrightarrow_R^* n''$ *and* $m'' \sim n''$.
- *weakly confluent modulo* $\sim$ *iff for all* m, n, p *with* $m \longrightarrow_R n$ *and* $m \longrightarrow_R p$, *there exist* n', p' *with* $n \longrightarrow_R^* n'$, $p \longrightarrow_R^* p'$ *and* $n' \sim p'$.
- *weakly coherent modulo* $\sim$ *iff for all* m, n, n' *with* $m \sim n$ *and* $n \longrightarrow_R n'$, *there exist* m', n'' *with* $m \longrightarrow_R^* m'$, $n' \longrightarrow_R^* n''$ *and* $m' \sim n''$.

Confluence, weak confluence, and weak coherence, all modulo $\sim$



Theorem 25 (Huet's Theorem [7]). *If the reduction relation* R *is strongly normalising, weakly confluent modulo* $\sim$ *and weakly coherent modulo* $\sim$, *then* R *is also confluent modulo* $\sim$.

Proposition 26.
$\longrightarrow_\beta, \longrightarrow_\eta, \longrightarrow_c, \longrightarrow_\gamma, \longrightarrow$ *are all weakly confluent modulo* $\sim$.

Proposition 27.
$\longrightarrow_\beta, \longrightarrow_\eta, \longrightarrow_c, \longrightarrow_\gamma, \longrightarrow$ *are all weakly coherent modulo* $\sim$.

Proposition 28.
$\longrightarrow_\beta, \longrightarrow_\eta, \longrightarrow_c, \longrightarrow_\gamma$ *are all confluent modulo* $\sim$.

Proof. By Huet's Theorem using Prop. 26, Prop. 27, Thm. 17 and Thm. 21.

We now show confluence of $\longrightarrow$ modulo $\sim$ using some intermediate Lemmas. The only non-trivial interaction is between β- and γ'-reduction. Following Barendregt [3, Chapter 11] we allow β-redexes to be marked. A redex is marked by overlining it. Notice that γ'-reduction can *hide* β-redexes inside a γ_E-redex. In such cases, we allow the γ_E-redex to be marked. For instance

$$\overline{(\lambda x.\delta(p, x_1.n_1, x_2.n_2))\, m} \longrightarrow_{\gamma'} \overline{\delta(p, x_1.\lambda x.n_1, x_2.\lambda x.n_2)\, m}$$

Definition 29.

$$\begin{aligned}
\varphi(\overline{(\lambda x.m)\,n}) &= \varphi(m)[x := \varphi(n)] \\
\varphi(\overline{\pi_i(\langle m, n\rangle)}) &= \varphi(m) \\
\varphi(\overline{\delta(\iota_i(m), x_1.n_1, x_2.n_2)}) &= \varphi(n_i)[x_i := \varphi(m)] \\
\varphi(\overline{E[\delta(p, x_1.n_1, x_2.n_2)]}) &= \delta(\varphi(p), x_1.\varphi(\overline{E[n_1]}), x_2.\varphi(\overline{E[n_2]}))
\end{aligned}$$

φ *commutes with all the other syntax constructors.*

The φ function contracts all of the marked β-redexes in a term.

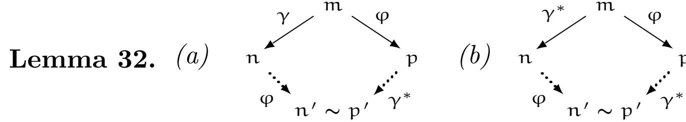
Lemma 30. $\varphi(m[x := n]) = \varphi(m)[x := n]$

Proof. By induction on the structure of m.

Lemma 31. *If* $m \sim m'$ *then* $\varphi(m) \sim \varphi(m')$.

Proof. By induction on the derivation of $m \sim m'$.

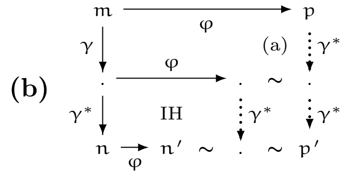
Lemma 32. *(a)* *(b)*



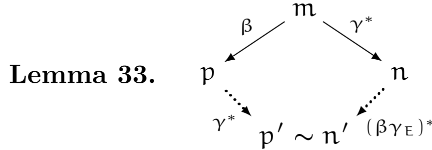
Proof.

(a) By induction on the derivation of γ using Lemma 30 and Lemma 31.

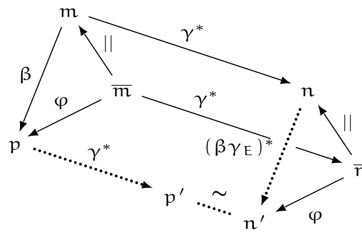
(b)



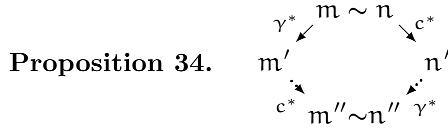
Lemma 33.



Proof. Let $\overline{m}$ be m with the β-redex marked, and $||$ be an operator that erases marked redexes, but otherwise leaves a term unchanged.



The front triangle is proved by induction on the structure of $\overline{n}$. The bottom rectangle is proved by Lemma 32.

Proposition 34. 

Proof. Using Lemma 33.

Theorem 35. $\longrightarrow$ *is confluent modulo* $\sim$.

Proof. By a diagram chase using Prop. 28 and Prop. 34.

Theorem 36. $\lambda^{\rightarrow\times+}$ *is decidable.*

Proof. By Thm. 35, and Thm. 23, every $\lambda^{\rightarrow\times+}$-term has a unique normal form obtained by reducing to $\beta\eta\gamma_E$-normal form and then to γ-normal form. To decide whether terms m, n are equal simply reduce them to normal forms m', n' and then compute whether $m' \sim n'$.

6 Variations

Unblocking cases It would be nice if it was possible to remove discrimination contexts from the *move-case*$_3$-rule, and so allow all the rewrite rules to be local. One way of doing so is to mark a case as blocked when it is adjacent to a lambda abstraction on whose bound variable the guard depends. Then unblocked cases can be lifted over blocked cases. The resulting calculus is somewhat fiddly, though, as blocked cases can subsequently become unblocked by β-reductions inside the guard. We omit the details, and instead consider some more well-behaved alternatives.

Parallel cases Altenkirch et al [1] use parallel cases in order to define normalisation by evaluation for sums. We write

$$\Delta([(x_0, p_0), \ldots, (x_{l-1}, p_{l-1})], [e_0, \ldots, e_{2^l-1}])$$

for the parallel case over the guards $p_0, \ldots, p_{l-1}$ with binders $x_0, \ldots, x_{l-1}$ and *tails* $e_0, \ldots, e_{2^l-1}$.

An easy way to comprehend the syntax is via the erasure *ser* from parallel cases to a tree of nested serial cases.

$$\begin{aligned} \mathit{ser}(\Delta((x, p) :: gs, es_1 +\!\!+ es_2)) &= \delta(p, x.\mathit{ser}(\Delta(gs, es_1)), x.\mathit{ser}(\Delta(gs, es_2))) \\ \mathit{ser}(\Delta([], [e])) &= e \end{aligned}$$

The *ser* function commutes with all other syntax constructors. The operator :: appends an element to the front of a list. The operator ++ concatenates two lists of equal length. The translation *par* from a $\lambda^{\rightarrow\times+}$-term to a term with parallel cases, simply converts each serial case into a parallel case with one guard.

$$\mathit{par}(\delta(p, x.n_1, x.n_2)) = \Delta((x, \mathit{par}(p)), [\mathit{par}(n_1), \mathit{par}(n_2)])$$

The *par* function commutes with all other syntax constructors.

Definition 37. *The relation $\approx$ is the least congruence such that for all permutations perm of the integers $1, \dots, n$:*

$$\Delta([], [e]) \approx e \qquad \Delta(gs, [e_0, \dots, e_{2^l - 1}]) \approx \Delta(gs', [e'_0, \dots, e'_{2^l - 1}])$$

where

$$gs = (x_0, p_0) \dots (x_{l-1}, p_{l-1}) \qquad e'_i = e_{perm*(i)}$$
$$gs' = (x'_0, p'_0) \dots (x'_{l-1}, p'_{l-1}) \text{ with } x'_i = x_{perm(i)} \text{ and } p'_i = p_{perm(i)}$$
$$perm*(i) = \uparrow (perm_2(\downarrow i)) \quad perm_2(b_{l-1} \dots b_0) = b_{perm(l-1)} \dots b_{perm(0)}$$
$$\downarrow, \uparrow \text{ convert natural numbers to and from binary}$$

Given two terms m_1 and m_2, then $m_1 \approx m_2$ iff m_2 can be obtained from m_1 by permuting guards (and adjusting binders and tails accordingly).

β- and η-rules The β- and η-rules are as in the serial rewriting theory, except for sums, where they are translated in the obvious way.

$$
\begin{array}{rrl}
(+.\beta_1) & \Delta((x, \iota_1(m)) :: gs, es_1 ++ es_2) & \longrightarrow \Delta(gs, es_1[x := m]) \\
(+.\beta_2) & \Delta((x, \iota_2(m)) :: gs, es_1 ++ es_2) & \longrightarrow \Delta(gs, es_2[x := m]) \\
(+.\eta) & m^{A+B} & \longrightarrow \Delta((x, m), [\iota_1(x), \iota_2(x)])
\end{array}
$$

For sum types, β-rules are only needed for the first guard of a parallel elimination. Other guards can just be eliminated by first applying $\approx$.

γ-rules

(*hoist-case*)

$$\mathsf{HP}[\Delta((x,p) :: gs, es_1 +\!\!+ es_2)]$$
$$\longrightarrow \Delta([x,p],[\mathsf{HP}[\Delta(gs,es_1)],\mathsf{HP}[\Delta(gs,es_2)]]]),$$
$$x \notin fv(\mathsf{HP}[\,]) \text{ and } bv(\mathsf{HP}[\,]) \cap fv(p) = \emptyset$$

(*redundant-guard*)

$$\Delta((x,p) :: gs, es_1 +\!\!+ es_2) \longrightarrow \Delta(gs, es_1),$$
$$es_1 \approx es_2 \text{ and } x \notin fv(es_1 +\!\!+ es_2)$$

(*repeated-guard*)

$$\Delta((x_1,p_1) :: (x_2,p_2) :: gs, (es_1 +\!\!+ es_2) +\!\!+ (es_3 +\!\!+ es_4))$$
$$\longrightarrow \Delta((x_1,p_1) :: gs, es_1 +\!\!+ es_4), \quad p_1 \approx p_2$$

(*join-cases*)

$$\Delta(gs,[e_1,\ldots,e_k,\ldots e_{2^l}]) \longrightarrow$$
$$\Delta((x,p) :: gs, [e'_{1,j} | 1 \leqslant j \leqslant 2^l] +\!\!+ [e'_{2,j} | 1 \leqslant j \leqslant 2^l])$$

where

$$e_k = \Delta((x,p) :: gs', es_1 +\!\!+ es_2)$$
$$\{x\} \notin \mathit{Binders}(gs) \text{ and } (\mathit{Binders}(gs) \cap fv(p)) = \emptyset$$
$$e'_{i,j} = \begin{cases} e_j, & \text{if } j \neq k \\ \Delta(gs', es_i), & \text{otherwise} \end{cases}$$
$$\mathit{Binders}(\Delta([(x_0,p_0),\ldots,(x_{l-1},p_{l-1})],[e_0,\ldots,e_{2^l-1}])) = \{x_0,\ldots,x_{l-1}\}$$

$$\begin{aligned}\mathsf{HP}[\,] ::= {} & [\,]\,n \mid \pi_1([\,]) \mid \pi_2([\,]) \mid \Delta((x,[\,]) :: gs, es) \\ & \mid \lambda x.[\,] \mid m\,[\,] \mid \langle [\,], n\rangle \mid \langle m, [\,]\rangle \mid \iota_1([\,]) \mid \iota_2([\,])\end{aligned}$$

The *redundant-guard*- and *repeated-guard*-rules are both obtained by reading the corresponding axioms from left to right. The *move-case*$_4$ axiom is captured by the combination of: parallel cases, the relation $\approx$ and the *join-cases*-rule; which allows a guard of a tail to be merged with the guards of its parent parallel case, providing the guard is independent of the guards of the parent.

Definition 38. $\longrightarrow_{\mathsf{P}} =$ *the union of the compatible closure of the above* β- *and* γ- *rules, and the restricted compatible closure of the above* η-*rules.*

The proofs of Sections 4 and 5 are easily adapted to handle parallel cases.

Proposition 39. $\longrightarrow_{\mathsf{P}/\approx}$ *is weakly normalising and confluent.*

Simulating parallel cases It is possible to simulate parallel cases using plain $\lambda^{\rightarrow\times+}$-syntax. The key to avoiding non-termination is to define a congruence such that guards can only be duplicated if in normal form.

Definition 40. *The relation* $\approx'$ *is the least congruence such that*

$$\begin{aligned}
&\delta(p_1, x_1.\delta(p_2, y_1.n_1, y_2.n_2), x_2.\delta(p_2, y_1.n_3, y_2.n_4)) \\
&\quad \approx' \delta(p_2, y_1.\delta(p_1, x_1.n_1, x_2.n_3), y_2.\delta(p_1, x_1.n_2, x_2.n_4)), \\
&\qquad x_1, x_2, y_1, y_2 \notin fv(p_1) \cup fv(p_2) \\
&\delta(p_1, x_1.\delta(p_2, y_1.n_1, y_2.n_2), x_2.n_3) \\
&\quad \approx' \delta(p_1, x_1.\delta(p_2, y_1.n_1, y_2.n_2), x_2.\delta(p_2, y_1.n_3, y_2.n_3)), \\
&\qquad x_2 \notin fv(p_2) \text{ and } y_1, y_2 \notin fv(n_3) \\
&\delta(p_1, x_1.n_1, x_2.\delta(p_2, y_1.n_2, y_2.n_3)) \\
&\quad \approx' \delta(p_1, x_1.\delta(p_2, y_1.n_1, y_2.n_1), x_2.\delta(p_2, y_1.n_2, y_2.n_3)) \\
&\qquad x_1 \notin fv(p_2) \text{ and } y_1, y_2 \notin fv(n_1)
\end{aligned}$$

where in each case p_1, p_2 *must be in normal form.*

γ*-rules*

$$\begin{aligned}
&(\textit{hoist-case}) \\
&\quad H[\delta(p, x_1.n_1, x_2.n_2)] \longrightarrow \delta(p, x_1.H[n_1], x_2.H[n_2]), \\
&\qquad x_1, x_2 \notin fv(H) \text{ and } bv(H) \cap fv(p) = \emptyset \\
&(\textit{duplicate-guard}) \\
&\quad \delta(p, x_1.\delta(p, y_1.m_1, y_2.m_2), x_2.\delta(p, y_1.n_1, y_2.n_2)) \\
&\quad \longrightarrow \delta(p, x_1.m_1[y_1 := x_1], x_2.n_2[y_2 := x_2]), \quad x_1, x_2 \notin fv(p) \\
&(\textit{redundant-guard}) \\
&\qquad C[\delta(p, x_1.n, x_2.n)] \longrightarrow C[n], \quad x_1, x_2 \notin fv(n) \text{ and} \\
&\qquad\quad C \equiv [\,]; \text{ or} \\
&\qquad\quad C \text{ is a hoisting frame; or} \\
&\qquad\quad C \text{ is a continuation frame with } (bv(C) \cap fv(p)) \neq \emptyset
\end{aligned}$$

The constraints on the context in which *redundant-guard* can be applied are necessary in order to prevent cycles with $\approx'$.

Definition 41. $\longrightarrow_p =$ *the union of* $\longrightarrow_\beta \cup \longrightarrow_\eta$ *and the compatible closure of the above* γ*-rules.*

Proposition 42. $\longrightarrow_{p/\sim'}$ *is weakly normalising and confluent.*

Conjecture 43. $\longrightarrow, \longrightarrow_{p/\approx}, \longrightarrow_{p/\sim'}$ are all strongly normalising.

Intuitively, it seems that $\longrightarrow$ should be strongly normalising. Both c-reduction and γ-reduction are strongly normalising, and γ'-reduction only interacts with c-reduction in such a way as to expose existing redexes, rather than actually creating new ones. If we could prove strong normalisation for $\longrightarrow$, then the confluence proof could be simplified.

7 Conclusion

We have proposed a confluent extensional rewriting theory for simply-typed lambda-calculus extended with sums. The key contribution is confluence and decidability for a conventional rewriting theory. This contrasts with the two previous approaches to decidability. Ghani [5] uses intricate rewriting techniques, whereas Altenkirch et al [1] use normalisation by evaluation and category theory.

Acknowledgements Thanks to Philip Wadler and the anonymous reviewers for helpful feedback.

References

1. T. Altenkirch, P. Dybjer, M. Hofmann, and P. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 303–310, Boston, Massachusetts, June 2001.
2. V. Balat, R. D. Cosmo, and M. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In *31st Symposium on Principles of Programming Languages (POPL 2004)*, pages 64–76. ACM Press, Jan. 2004.
3. H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Number 103 in Studies in Logics and the Foundations of Mathmatics. North Holland, 1984.
4. U. Berger, M. Eberl, and H. Schwichtenberg. Normalization by evaluation. In *Prospects for Hardware Foundations (NADA)*, number 1546 in Lecture Notes in Computer Science, pages 117–137, 1998.
5. N. Ghani. Beta-eta equality for coproducts. In *Proceedings of TLCA'95*, number 902 in Lecture Notes in Computer Science, pages 171–185. Springer-Verlag, 1995.
6. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.
7. G. P. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980.
8. S. Lindley. *Normalisation by Evaluation in the Compilation of Typed Functional Programming Languages*. PhD thesis, University of Edinburgh, 2005.
9. S. Lindley and I. Stark. Reducibility and $\top\top$-lifting for computation types. In P. Urzyczyn, editor, *TLCA*, volume 3461 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2005.
10. Y. Ohta and M. Hasegawa. A terminating and confluent linear lambda calculus. In *RTA*, pages 166–180, 2006.
11. D. Prawitz. Ideas and results in proof theory. In *Proceedings of the 2nd Scandinavian Logic Symposium*, number 63 in Studies in Logics and the Foundations of Mathmatics, pages 235–307. North Holland, 1971.