Subject Reduction and Confluence for the *reFLect* Language*

Sava Krstić and John Matthews

OGI School of Science & Engineering at OHSU

1 Introduction

This paper presents several technical results concerning the operational semantics of the reflective functional programming language *reFLect*, currently being developed at Intel [7].

The design of *reFLect* has been motivated by an existing functional language, FL, that Intel has used for a number of years as an aid for large-scale hardware verification [1]. FL is a statically typed, polymorphic programming language with a lazy evaluation semantics. A key language feature that distinguishes FL from existing lazy functional languages such as *Haskell* is its treatment of symbolic Boolean formulas as first-class entities. The FL interpreter compactly represents symbolic formulas as binary decision diagrams (bdd's), and efficiently manipulates the formulas through highly optimized bdd routines written in C.

FL's features make it possible to specify both circuits *and* circuit verification algorithms in a highly declarative form. The overhead of executing the verification algorithms in FL's interpreter is insignificant, since the vast majority of time and space is taken by the underlying bdd routines.

However, the circuits that need to be verified are still much too large to be handled by automatic circuit verification algorithms. Therefore FL has also been used to implement an interactive theorem prover based on a higher order logic. The theorem prover is used to decompose large verification problems into a series of smaller problems, each of which can be checked using existing circuit verification algorithms.

The theorem prover's term language is based on FL. In principle, this also allows verification algorithms as well as circuits to be formally reasoned about. For example, the theorem prover could prove that a model checking algorithm always returns the correct results for any circuit. This ability of a theorem prover to reason about its own logic and verification algorithms is called *reflection* in the literature [8].

Based on their experiences with FL, Grundy, O'Leary, and Melham introduce a successor language *reFLect* in [7]. It allows *any expression* of the language to be manipulated as a first-class data structure, not just Boolean formulas. Implementing model checkers, compilers, theorem provers, and other code-transforming tools should be significantly easier in the new language. Circuits can also be specified in *reFLect* at least as easily as they could be in FL.

* This research was supported by a grant from the Intel Corporation.

Since *reFLect* is meant to be the term language of a theorem prover, it is important that the semantics of the language be logically consistent. For example, it should not be possible for an expression to initially be assigned one type, but result in a final value of a different type. Nor should it be possible to reduce the same *reFLect* expression in two different ways to two semantically different values. These properties are called *subject reduction* and *confluence*, respectively. Proving them is the main goal of this paper.

The language considered in this paper is the core *reFLect*, as described in Sections 1–7 of [7]. The full language has three additional constructs that facilitate reflection ([7], Section 8), but these are left out in this initial semantical investigation of the language.

We use standard proof techniques, with appropriate modifications to account for the reflective nature of the language. For example, even though the language definition begins with a simple context-free description of raw expressions, the actual well-formed *reFLect* expressions exhibit a distinctive recursive structure that is more subtle. Proofs by structural induction refer to this structure rather than the one induced by the grammar.

Basic notions like free variable occurrences, α-equivalence, and substitution need to be overhauled too because they fundamentally bear upon the *reFLect*'s own concepts of patterns, quotations and antiquotations. For example, it takes some effort to verify the simple but vital fact that substitution is a well defined operation on α-equivalence classes.

This paper is self-contained, but is written in a rather dry style. It is meant to be a companion paper to [7], where the reader should find motivation for various design decisions and examples.

In Section 2, we define well-formed *reFLect* expressions (slightly varying the original definition) and several pertinent notions. Lemma 2 is the basis for proofs by structural induction. Also important is Lemma 5, showing preservation of well-formedness by replacement.

Section 3 gives a rather complete account of α-equivalence and substitution. We state a number of facts needed in the rest of the paper. Even though some proof details are omitted, all necessary definitions and main proof ideas are given.

In Section 4, we define the *reFLect* reduction system, and also a simpler auxiliary non-deterministic system. In Sections 5 and 7 we prove that subject reduction and strong normalization (termination) hold for the auxiliary system. These results immediately imply that the same properties hold for *reFLect* as well. In Section 6, we prove local confluence of *reFLect*, which with the already established termination result completes the proof of confluence for *reFLect*.

Acknowledgments We are grateful to Jim Grundy, John O'Leary and Tom Melham for their help in understanding *reFLect*; and to Andy Moran, Sylvain Conchon and Alexandre Miquel for helping us track down references to non-deterministic lambda calculus. We would especially like to thank Jim Grundy for carefully reading and debugging various versions of our paper.

2 Syntax

2.1 Types

The set of *reFLect* types is freely generated by a set of TYPE VARIABLES and a set of TYPE CONSTRUCTORS in the usual manner. The language requires only the binary constructor $\Rightarrow$ and a primitive type (nullary constructor) term.

A TYPE INSTANTIATION is a map ϕ from type variables to types, with finite "domain" $\mathsf{dom}(\phi) = \{\alpha \mid \phi(\alpha) \neq \alpha\}$. Every type instantiation extends recursively to a map, also denoted by ϕ, from types to types.

Lemma 1. *Given types $\sigma_1, \ldots, \sigma_k$ and $\tau_1, \ldots, \tau_k$, there exists at most one type instantiation ϕ such that*

(1) $\sigma_i\phi = \tau_i$ for all $i = 1, \ldots, k$;
(2) all type variables in $\mathsf{dom}(\phi)$ occur in the σ_i. □

The type instantiation ϕ as in the lemma will be denoted $(\tau_1, \ldots, \tau_k)/(\sigma_1, \ldots, \sigma_k)$. This notation will be used only in specifying the reduction system in Section 4.

2.2 Expressions and Contexts

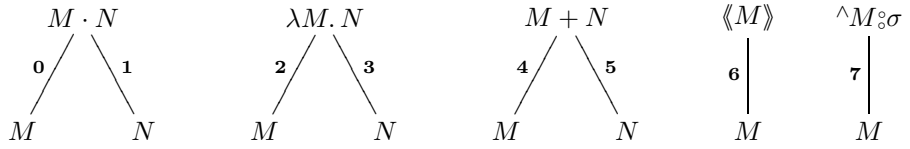
EXPRESSIONS of *reFLect* are described by the grammar

$$M \ ::= \ v^\sigma \ \mid \ M \cdot M \ \mid \ \lambda M.\, M \ \mid \ M + M \ \mid \ \langle\!\langle M \rangle\!\rangle \ \mid \ {}^\wedge M{\circ\!\!\circ}\sigma \qquad (1)$$

and we will use the terminology VARIABLE, APPLICATION, LAMBDA ABSTRACTION, ALTERNATION, QUOTATION, ANTIQUOTATION to classify the expressions according to the top symbol in them. The symbol σ in (1) stands for an arbitrary type.

Remark 1. The original *reFLect* paper [7] uses the alternation production $M ::= \lambda M.\, M + M$, so the set of legitimate expressions in [7] is smaller than ours. These STANDARD *reFLect* EXPRESSIONS can be characterized as expressions defined by (1) with the additional constraint that in every subexpression of the form $M+N$, the subexpression M must be an abstraction. We have adopted the less restrictive syntax only for reasons of convenience; every result we prove in this section and the next translates with trivial justification into the same result in the context of standard *reFLect* expressions. The same is true for the main results proved in the last three sections; see Remark 7 in Section 4.

It is convenient to view expressions as ordered trees with internal nodes labeled by symbols $\cdot$, λ, $+$, $\langle\!\langle\rangle\!\rangle$,${}^\wedge{\circ\!\!\circ}\sigma$, and with leaves labeled by variables. The edges are labeled by numbers between **0** and **7**, according to the symbol at the edges' source nodes, as follows:



Every node in the tree has its unique POSITION—the string over $\{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{7}\}$ describing the sequence of labels from the root to that node. We will use the notation ε for the empty string (the root position). Every expression M thus has an associated prefix-closed set of positions $\mathsf{Pos}(M)$. We will write $\pi \leq \pi'$ to indicate that π is a prefix of π' and $\pi \wedge \pi'$ for the largest common prefix of π and π'.

The subexpression of M at position π will be denoted $M|_\pi$. Clearly,

$$\mathsf{Pos}(M|_\pi) = \{\pi' \mid \pi\pi' \in \mathsf{Pos}(M)\} \quad \text{and} \quad M|_\pi|_{\pi'} = M|_{\pi\pi'} \tag{2}$$

The REPLACEMENT of $M|_\pi$ with N in M will be denoted $M[\pi \mapsto N]$.

The subexpression $E|_\pi$ is a variable if and only if π is a maximal (with respect to the prefix ordering) element of $\mathsf{Pos}(E)$; the set of variable positions will be denoted $\mathsf{VarPos}(E)$. Note that E is completely described by the set $\mathsf{Pos}(E)$ and by knowing $E|_\pi$ for all $\pi \in \mathsf{VarPos}(E)$.

CONTEXTS are defined using the same grammar (1) extended with the hole production $M ::= \sqcup$. The definition of replacement trivially extends to contexts.

2.3 Levels

Every subexpression of an expression M has an associated LEVEL; it is defined to be the number of the enclosing quotations of M minus the number of enclosing antiquotations. Thus, the level of the subexpression $M|_\pi$ is equal to

$$\mathsf{level}(\pi) = \#\{\text{occurrences of } \mathbf{6} \text{ in } \pi\} - \#\{\text{occurrences of } \mathbf{7} \text{ in } \pi\}.$$

Note the additivity of levels:

$$\mathsf{level}(\pi\pi') = \mathsf{level}(\pi) + \mathsf{level}(\pi'). \tag{3}$$

We will be interested only in LEVEL CONSISTENT EXPRESSIONS, defined as those in which every position has a non-negative level. More generally, a LEVEL CONSISTENT CONTEXT is one in which all positions have non-negative level and all hole positions have level zero.

Level consistent expressions exhibit a block structure determined by the partition of the set of positions into those of level 0 (ACTIVE), and those of level greater than 0 (PASSIVE). This structure is depicted in Figure 1. Informally, starting at the root and walking down a level consistent expression's tree, one goes through active nodes for some time (the top block ① in the picture), until encountering the first quote node. This node is the entry point to the first layer

of passive positions, grouped into blocks ②, ③, and ④ in the picture. One can pass through any number of quote or antiquote positions while staying in the passive block, as long as the total number of visited quote positions remains greater than the total number of visited antiquote positions. Getting into an antiquote node that balances the two totals is the entry point into the new layer of active nodes. In the figure, that new layer cannot be reached from block ②, while from block ③ there are two antiquote "doors" to active blocks ⑤ and ⑥.

Fig. 1. Structure of level consistent expressions.

A convenient formalization of the inductive structure of the set of level consistent expressions is given by the following lemma. It will be repeatedly used to justify recursive definitions of functions and relations whose argument is a level consistent expression.

Lemma 2. *If $M, N, M_1, \ldots, M_k$ are level consistent expressions and $\mathcal{C}$ is a level consistent context, then the expressions*

$$v^\sigma \qquad M \cdot N \qquad \lambda M.\, N \qquad M + N \qquad \langle\!\langle \mathcal{C}[{}^\wedge M_1 {}^\circ_\circ \sigma_1, \ldots, {}^\wedge M_k {}^\circ_\circ \sigma_k] \rangle\!\rangle$$

are all level consistent. Moreover, every level consistent expression can be uniquely written in one of these five forms.

Proof. Straightforward. □

Note that the uniqueness part of the lemma says in particular that for every level consistent quotation M there exist a unique level consistent context $\mathcal{C}$ (with, say, k holes), unique types $\sigma_1, \ldots, \sigma_k$, and unique level consistent expressions $M_1, \ldots, M_k$ such that $M = \langle\!\langle \mathcal{C}[{}^\wedge M_1 {}^\circ_\circ \sigma_1, \ldots, {}^\wedge M_k {}^\circ_\circ \sigma_k] \rangle\!\rangle$. This equation will be referred to as the FACTORIZATION of the quotation M, and the expressions $M_1, \ldots, M_k$ will be called FACTORS of M.

In the sequel we will also use the term ACTIVE SUBEXPRESSION to mean a subexpression occurring at an active position.

2.4 Typing

The following rules define the set of WELL-TYPED expressions.

Typing Rules 1

$$\frac{}{v^\sigma : \sigma} \qquad \frac{M : \sigma \Rightarrow \tau \qquad N : \sigma}{M \cdot N : \tau}$$

$$\frac{L : \sigma \qquad M : \tau}{\lambda L.\, M : \sigma \Rightarrow \tau} \qquad \frac{M : \sigma \qquad N : \sigma}{M + N : \sigma}$$

$$\frac{M_i : \mathsf{term} \quad (1 \le i \le k) \qquad \mathcal{C}[z_1^{\sigma_1}, \ldots, z_k^{\sigma_k}] : \sigma \quad (\mathcal{C} \text{ level consistent, } z_i \text{ fresh})}{\langle\!\langle \mathcal{C}[\hat{}M_1 \mathbin{\wr} \sigma_1, \ldots, \hat{}M_k \mathbin{\wr} \sigma_k] \rangle\!\rangle : \mathsf{term}}$$

Note that the last rule has a meaning when $k = 0$. It says then than we can derive $\langle\!\langle M \rangle\!\rangle : \mathsf{term}$ whenever we have $M : \sigma$, for any σ. Such expressions (namely, well-typed quotations without factors) will be called PURE QUOTATIONS.

Lemma 3. *If M is a well-typed expression, then M is level consistent and there is a unique type σ such that $M : \sigma$.*

Proof. Induction on the structure of M, using Lemma 2. □

Remark 2. A standard *reFLect* expression is well-typed in our system if and only if it is well-typed in [7].

2.5 Well-formedness

We define a PATTERN to be a level consistent expression that is either a variable or a quote all of whose factors are variables. If L is a pattern, we will denote by $\mathsf{PatVar}(L)$ the set of variables occurring at active positions in L.

By definition, an expression is WELL-FORMED if it is well-typed and satisfies the following PATTERN CONDITION: in every active subexpression of the form $\lambda L.\, M$, the expression L is a pattern.

We also define PATTERN POSITIONS to be the active postions which contain occurrences of **2**; ROOT PATTERN POSITIONS are those that end in **2**. Thus, E satisfies the pattern condition if and only if $E|_\pi$ is a pattern for every root pattern position π of E.

Lemma 4. *All active subexpressions of a well-formed expression are well-formed.*

Proof. Assume E is a well-formed expression and π an active position in it.

From equations (2) and (3) it follows that the root pattern positions of $E|_\pi$ are precisely the positions π' such that $\pi\pi'$ is a root pattern position of E; thus, $E|_\pi$ satisfies the pattern condition.

It remains to prove that $E|_\pi$ is well-typed. The proof is by induction on the structure of the expression E and the base case $E = v^\sigma$ is evident. Suppose that E is of the form $L \cdot M$, $\lambda L.\, M$, or $L + M$. The typing rules imply that L and M are well-typed, and since $E|_\pi$ is an active subexpression of L or M, it must be well-typed by induction hypothesis.

Finally, suppose $E = \langle\!\langle \mathcal{C}[\hat{}M_1 \circ \sigma_1, \ldots, \hat{}M_k \circ \sigma_k] \rangle\!\rangle$. Again by typing rules, the factors M_i must be well-typed and $E|_\pi$ must be an active subexpression of one of them, so the induction hypothesis applies. □

Corollary 1. *The factors of well-formed quotations are well-formed.* □

Well-formedness is preserved by replacement of active subexpressions with other well-formed expressions of the same type. This is made precise in the following key lemma, to be used in defining substitution action (Section 3) and in the proof of subject reduction (Section 5).

Lemma 5. *Suppose E is a well-formed expression of type σ and π is an active non-pattern position in E. If $E|_\pi : \tau$ and N is any well-formed expression of type τ, then $E[\pi \mapsto N]$ is a well-formed expression of type σ.*

Proof. Every root pattern position of $E[\pi \mapsto N]$ is either a root pattern position of E or a root pattern positions of N prefixed with π. Moreover the subexpression of $E[\pi \mapsto N]$ corresponding to that position is equal to the corresponding subexpression of E or N, and so is a pattern Thus, $E[\pi \mapsto N]$ satisfies the pattern condition, and it remains to prove that it is well-typed. This is done by induction on the type derivation of E.

Case 1: $E = v^\sigma$. Evident.

Case 2: $E = L \cdot M$, where $L : \upsilon \Rightarrow \sigma$ and $M : \upsilon$. We have $\pi = \mathbf{0}\pi'$ or $\pi = \mathbf{1}\pi'$ for some π'. Suppose $\pi = \mathbf{0}\pi'$; the other case is similar and will be omitted. We have $E|_\pi = L|_{\pi'}$ and π' is a non-pattern active position in L. From Lemma 4 we know that L is well-formed, so the induction hypothesis implies $L[\pi' \mapsto N] : \upsilon \Rightarrow \sigma$ and the typing rule for application allows us to derive $L[\pi' \mapsto N] \cdot M : \sigma$. Thus $E[\pi \mapsto N]$, being equal to $L[\pi' \mapsto N] \cdot M$, is well-typed.

Case 3: $E = \lambda L.\, M$, where $\sigma \equiv \upsilon \Rightarrow \omega$, $L : \upsilon$, and $M : \omega$. Since π is a non-pattern position, we must have $\pi = \mathbf{3}\pi'$ and $E[\pi \mapsto N] = \lambda L.\, M[\pi' \mapsto N]$, where π' is a non-pattern active position in M. The induction hypothesis implies $M[\pi' \mapsto N] : \omega$, so the typing rule for abstractions gives $E[\pi \mapsto N] : \upsilon \Rightarrow \omega$.

Case 4: $E = L + M$. Similar to Case 2.

Case 5: $E = \langle\!\langle \mathcal{C}[\hat{}M_1 \circ \sigma_1, \ldots, \hat{}M_k \circ \sigma_k] \rangle\!\rangle$, where $\sigma = \mathsf{term}$, and the other notation is as in Table 1. As in the corresponding case of Lemma 4, we have that $E|_\pi$ is a subexpression of some M_i. More precisely, $M_i = E|_{\pi'}$ and $E|_\pi = M_i|_{\pi''}$,

where $\pi = \pi'\pi''$. Since $\mathsf{level}(\pi') = 0$, it follows that $\mathsf{level}(\pi'') = 0$. Also, π'', being a suffix of π, is a non-pattern position. The factors M_i are well-formed (Corollary 1), so the induction hypothesis yields $M_i[\pi'' \mapsto N] : \mathsf{term}$. Thus, by the typing rule for quotes, $E[\pi \mapsto N]$ has a factorization just like that of E, but with $M_i[\pi'' \mapsto N]$ in place of M_i, and so $E[\pi \mapsto N]$ is well-typed. □

Remark 3. All results of this subsection remain valid when "well-typed expression" is replaced everywhere with "well-formed expression". The same proofs apply; one just needs to ignore the parts where the pattern condition is checked.

2.6 Type Instantiation

The inductive definition of the action of a type instantiation ϕ on a well-formed expression L, denoted $L\phi$, is given by the following set of equations.

Type Instantiation 2

$$
\begin{aligned}
v^{\sigma}\phi &= v^{\phi(\sigma)} \\
(L \cdot M)\phi &= (L\phi) \cdot (M\phi) \\
(\lambda L.\, M)\phi &= \lambda(L\phi).\, (M\phi) \\
(L + M)\phi &= (L\phi) + (N\phi) \\
\langle\!\langle \mathcal{C}[{}^{\wedge}M_1{:}\sigma_1, \ldots, {}^{\wedge}M_k{:}\sigma_k] \rangle\!\rangle\phi &= \langle\!\langle \mathcal{C}[{}^{\wedge}(M_1\phi){:}\sigma_1, \ldots, {}^{\wedge}(M_k\phi){:}\sigma_k] \rangle\!\rangle
\end{aligned}
$$

Type instantiation can be applied to contexts as well; we just need to add $\_\phi = \_$ to the definition above.

Lemma 6. *Suppose ϕ is a type instantiation and E is a well-formed expression of type σ. Then $E\phi$ is a well-formed expression of type $\phi(\sigma)$. Also, if $E = \mathcal{C}[M_1, \ldots, M_k]$, where $\mathcal{C}$ is a level consistent context, then $E\phi = (\mathcal{C}\phi)[M_1\phi, \ldots, M_k\phi]$.*

Proof. Induction on the structure of E. □

3 Substitution

A SUBSTITUTION θ is a map from variables to expressions such that for every variable x the expression $\theta(x)$ is well-formed and of the same type as x. Moreover, as for type instantiations, we require that the "domain" $\mathsf{dom}(\theta) = \{x \mid \theta(x) \neq x\}$ be finite. The goal of this section is to define the *reFLect* notion of α-equivalence and the capture-avoiding action of substitutions on α-equivalence classes.

3.1 Free and Bound

Suppose E is a well-formed expression. The BINDING POSTITIONS in E are variable positions that are also pattern positions in E. (It follows that every binding position is active.) Suppose now π is a non-binding active variable position in E. Let us say that a position π' is an ABSTRACTION PREFIX of π if π' is active and $\pi'\mathbf{3} < \pi$. Let $\{\pi_1, \ldots, \pi_k\}$ be the set of all abstraction prefixes of π. We allow $k = 0$ here—the case when π has no abstraction prefixes. Each subexpression $E|_{\pi_i}$ is an abstraction $\lambda L_i.\, M_i$. We will say that π is BOUND or FREE occurrence of the variable $E|_\pi$ in E depending on whether the variable $E|_\pi$ occurs as a pattern variable in some L_i or not.

In the case when π is bound in E, the variable $E|_\pi$ may occur in several L_i, but only the occurrences of $E|_\pi$ in the innermost of those L_i will be binding: by definition, a binding position μ BINDS the position π in E when $E|_\pi = E|_\mu$ and $\pi' = \mu \wedge \pi$ is the largest abstraction prefix of π such that $E|_\pi \in \mathsf{PatVar}(E|_{\pi'\mathbf{2}})$.

Thus, we have a partition

$$\mathsf{VarPos}(E) = \mathsf{FreePos}(E) \uplus \mathsf{BdPos}(E) \uplus \mathsf{PassPos}(E),$$

where $\mathsf{FreePos}$ and $\mathsf{PassPos}$ denote the sets of free and passive positions, and BdPos stands for the (disjoint) union of binding and bound positions. We will also write $\mathsf{FreePos}(E, x)$ for the set of all $\pi \in \mathsf{FreePos}(E)$ such that $E|_\pi = x$.

3.2 Alpha-Equivalence

The set $\mathsf{BdPos}(E)$ is partitioned into BINDING CLUSTERS. There is a bijective correspondence between the set of clusters of E and the set of pairs (π, x), where π is an active abstraction position in E (say, $E|_\pi = \lambda L.\, M$) and $x \in \mathsf{PatVar}(L)$. By definition, the cluster corresponding to the pair (π, x) is the set

$$\{\pi\mathbf{2}\mu \mid E|_{\pi\mathbf{2}\mu} = x\} \uplus \{\pi\mathbf{3}\pi' \mid \pi' \in \mathsf{FreePos}(E|_{\pi\mathbf{3}}, x)\}.$$

The type of x will also be called the type of this cluster.

It is easy to check that μ binds π if and only if μ and π are a binding and non-binding positions belonging to the same cluster.

Two expressions E and E' will be called α-EQUIVALENT (written $E \sim_\alpha E'$) when

(α_1) $\mathsf{Pos}(E) = \mathsf{Pos}(E')$
(α_2) The clusters of E and E' are equal and of the same type
(α_3) $E|_\pi = E'|_\pi$ for all $\pi \in \mathsf{VarPos}(E) - \mathsf{BdPos}(E)$

The condition (α_1) implies $\mathsf{PassPos}(E) = \mathsf{PassPos}(E')$ and (α_2) implies $\mathsf{BdPos}(E) = \mathsf{BdPos}(E')$. Thus, a position in E is free, binding, bound, or passive if and only if it is of the same kind in E'. Moreover, it follows from (α_3) that $\mathsf{FreePos}(E, x) = \mathsf{FreePos}(E', x)$ for every x.

Remark 4. Alpha-equivalence is clearly an equivalence relation, and every expression α-equivalent to a well-formed expression is well-formed itself, and of the same type.

Remark 5. Alpha-equivalent patterns must be equal. Also, α-equivalence preserves the inductive structure of expressions. More precisely, if $L \sim_\alpha M$ and $L' \sim_\alpha M'$, then $L \cdot M \sim_\alpha L' \cdot M'$, $L + M \sim_\alpha L' + M'$, and $\lambda L.\, M \sim_\alpha \lambda L.\, M'$. Moreover, $\langle\!\langle \mathcal{C}[\hat{}M_1 \circ \sigma_1, \ldots, \hat{}M_k \circ \sigma_k] \rangle\!\rangle \sim_\alpha \langle\!\langle \mathcal{C}[\hat{}M'_1 \circ \sigma_1, \ldots, \hat{}M'_k \circ \sigma_k] \rangle\!\rangle$ holds provided $M_i \sim_\alpha M'_i$.

Suppose C is a cluster in E corresponding to some pair (π, x). If $E \sim_\alpha E'$, then in E' the same cluster C corresponds to the pair (π, y) for some variable y of the same type as x. On the other hand, if z is an arbitrary variable of the same type as x, the expression $E' = E[\pi \mapsto z]^{\pi \in C}$ need not be α-equivalent to E. The cluster C' of E' at (π, z) can be strictly larger than C. There are two ways for this to happen. First, it is possible that the pair (π, z) corresponds to a cluster C_1 of E (which is to say that $z \in \mathsf{BdPos}(E|_\pi)$), and then we have $C' = C \uplus C_1$. The second possibility is that z occurs free in $E|_\pi$; in this case $C' = C \uplus \pi\mathsf{FreePos}(E|_\pi, z)$.

Lemma 7. *For every expression E and every finite set X of variables, there exists an expression that is α-equivalent to E and contains no bound occurrences of any variable in X.*

Proof. For each cluster C of E, let z_C be a distinct variable that does not occur in either E or X. By the discussion in the previous paragaph, the expression $E' = E[\pi \mapsto z]^{\pi \in C \in \mathcal{C}}$, where $\mathcal{C}$ is the set of all clusters of E, is α-equivalent to E and its bound variables are the z_C's. □

3.3 Substitution Action

If θ is a substitution and E is a well-formed expression, the result of BLIND ACTION (not capture-avoiding) of θ on E is the expression obtained by replacing all occurrences of free variables with their θ-values:

$$E^\theta = E[\pi \mapsto \theta(E|_\pi)]^{\pi \in \mathsf{FreePos}(E)}$$

Remark 6. We can think of E^θ as being obtained from E through a sequence of steps each of which consists of replacing an occurrence of a variable with a well-formed expression of the same type. Thus, by Lemma 5, E^θ is a well-formed expression of the same type as E.

A complete description of E^θ is given by

$$\mathsf{Pos}(E^\theta) = \mathsf{Pos}(E) \cup \textstyle\bigcup_{x \in \mathsf{dom}(\theta)} \{\pi\pi' \mid \pi \in \mathsf{FreePos}(E, x),\ \pi' \in \mathsf{Pos}(\theta(x))\} \quad (4)$$

$$\begin{aligned} \mathsf{VarPos}(E^\theta) = {} & \mathsf{VarPos}(E) - \mathsf{FreePos}(E) \\ & \cup \textstyle\bigcup_{x \in \mathsf{dom}(\theta)} \{\pi\pi' \mid \pi \in \mathsf{FreePos}(E, x),\ \pi' \in \mathsf{Pos}(\theta(x))\} \end{aligned} \quad (5)$$

and

$$E^\theta|_\pi = E|_\pi \qquad \text{if } \pi \in \mathsf{VarPos}(E) - \mathsf{FreePos}(E) \tag{6}$$

$$E^\theta|_{\pi\pi'} = \theta(x)|_{\pi'} \quad \text{if } \pi \in \mathsf{FreePos}(E,x) \text{ and } \pi' \in \mathsf{Pos}(\theta(x)) \tag{7}$$

If $\pi \in \mathsf{FreePos}(E,x)$ and $\pi' \in \mathsf{FreePos}(\theta(x))$, then $\pi\pi'$ will be in $\mathsf{FreePos}(E^\theta)$ unless the variable $\theta(x)|_{\pi'}$ occurs as a pattern variable in some patterns associated to abstraction prefixes of π in E. A simple condition that clearly eliminates this possibility is that E is FREE FOR θ in the sense that no binding variable of E occurs freely in any of the expressions $\theta(x)$, where $x \in \mathsf{dom}(\theta)$.

An active abstraction position in E^θ is either (a) an active abstraction position in E; or (b) of the form $\pi\pi'$ where $\pi \in \mathsf{FreePos}(E,x)$ and π' is an active abstraction position in $\theta(x)$. In the case (a), the cluster in E^θ corresponding to a pair (π, y) contains the cluster of E corresponding to the pair (π, y), and the two are equal if E is free for x. In the case (b), the cluster in E^θ corresponding to a pair $(\pi\pi', y)$ is equal πC, where C is the cluster of $\theta(x)$ corresponding to the pair (π', y). In both cases, the types of the associated clusters are obviously the same.

Lemma 8. *If $E \sim_\alpha E'$ and both E and E' are free for θ, then $E^\theta \sim_\alpha E'^\theta$.*

Proof. The condition (α_1) needed for $E^\theta \sim_\alpha E'^\theta$ follows immediately from (4) and the fact $\mathsf{FreePos}(E,x) = \mathsf{FreePos}(E',x)$. The condition (α_2) follows from the above analysis of binding clusters in E^θ.

From equation (7) we get $E^\theta|_{\pi\pi'} = E'^\theta|_{\pi\pi'}$ for every $\pi \in \mathsf{FreePos}(E,x)$ and $\pi' \in \mathsf{Pos}(\theta(x))$. Thus, for the proof of (α_3), it only remains to check that $E^\theta|_\pi = E'^\theta|_\pi$ holds for all $\pi \in \mathsf{VarPos}(E) - \mathsf{FreePos}(E)$ such that $\pi \notin \mathsf{BdPos}(E^\theta)$. Such π is not in $\mathsf{BdPos}(E)$, so indeed $E^\theta|_\pi = E|_\pi = E'|_\pi = E'^\theta|_\pi$, by $E \sim_\alpha E'$ and equation (6). □

The CAPTURE-AVOIDING ACTION of substitutions is defined on α-equivalence classes. If E is a well-formed expression and $[E]$ its α-equivalence class, we define $[E]\theta$ to be the class $[E'^\theta]$, where E' is a representative of $[E]$ that is free for θ. Existence if E' follows from Lemma 7, and Lemma 8 shows correctness of our definition (independence of the chosen representative E').

From now on, we will consider α-equivalent expressions as equal; that is, by "a well-formed expression" we will really mean "an α-equivalence class". By Remark 5, the notation $L \cdot M$, $L + M$, $\lambda L.\, M$ and $\langle\!\langle \mathcal{C}[\hat{}M_1 \circ \sigma_1, \ldots, \hat{}M_k \circ \sigma_k] \rangle\!\rangle$ makes sense when L, M, M_i are α-equivalence classes. (Note however, that $\mathcal{C}$ is a context just as before.)

In particular, we will write $E\theta$ for the capture-avoiding substitution. In view of remarks 4 and 6, and Lemma 8, well-formedness is a property of α-classes, and is preserved by substitution actions:

Lemma 9. *For every well-formed expression E and every substitution θ, the expression $E\theta$ is well-formed and of the same type as E.* □

Note that $E\theta$ is really an abbreviation for $[E]\theta$ and that $E\theta = [E^\theta]$ when E is free for θ.

Lemma 10. *Suppose π is an active position in E, and E is well-formed and free for θ. Then $(E\theta)|_\pi = (E|_\pi)\theta$.*

Proof. $E|_\pi$ is also free for θ, so the lemma follows from the obvious equality of expressions (not α-equivalence classes) $(E^\theta)|_\pi = (E|_\pi)^\theta$. □

Lemma 11. *The capture-avoiding substitution action satisfies the following properties:*

$$\begin{aligned} x\theta &= \theta(x) \qquad (8)\\ (L \cdot M)\theta &= (L\theta) \cdot (M\theta)\\ (\lambda L.\, M)\theta &= \lambda L.\, M\theta\\ (L + M)\theta &= L\theta + M\theta\\ \langle\!\langle \mathcal{C}[{}^\wedge M_1 {}^\circ_\circ \sigma_1, \ldots, {}^\wedge M_k {}^\circ_\circ \sigma_k] \rangle\!\rangle \theta &= \langle\!\langle \mathcal{C}[{}^\wedge (M_1\theta) {}^\circ_\circ \sigma_1, \ldots, {}^\wedge (M_k\theta) {}^\circ_\circ \sigma_k] \rangle\!\rangle \end{aligned}$$

where, in the case of abstraction, the representative expression $\lambda L.\, M$ is assumed to be free for θ.

Proof. Immediate from Lemma 10. □

As usual, we write $[M/x]$ for the substitution which maps x to M and fixes all other variables.

Lemma 12. *Suppose $x \notin \mathsf{dom}(\theta)$ and x does not occur freely in any $\theta(y)$ for $y \in \mathsf{dom}(\theta)$. Then $E[M/x]\theta = (E\theta)[M\theta/x]$.*

Proof. Let L and R be the left-hand and the right-hand sides of the desired equality. It is straightforward (though laborious) to check that $\mathsf{Pos}(L) = \mathsf{Pos}(R)$ and that $L|_\pi = R|_\pi$ for every $\pi \in \mathsf{VarPos}(L)$. □

3.4 Pattern Matching

We will use the notation $\theta :\ L \Rightarrowtail M$ to say that the pattern L MATCHES the expression M via the substitution θ. The matching relation is described by the following two rules.

Pattern Matching 3

$$\frac{M :\ \sigma \qquad \theta = [v^\sigma \mapsto M]}{\theta : v^\sigma \Rightarrowtail M}$$

$$\frac{\theta = [v_1^{\mathsf{term}} \mapsto \langle\!\langle M_1 \rangle\!\rangle, \ldots, v_k^{\mathsf{term}} \mapsto \langle\!\langle M_k \rangle\!\rangle] \qquad M_i :\ \tau_i \qquad \tau_i = \sigma_i\phi \qquad (i = 1, \ldots, k)}{\theta :\ \langle\!\langle \mathcal{C}[{}^\wedge (v_1^{\mathsf{term}}) {}^\circ_\circ \sigma_1, \ldots, {}^\wedge (v_k^{\mathsf{term}}) {}^\circ_\circ \sigma_k] \rangle\!\rangle \Rightarrowtail \langle\!\langle (\mathcal{C}\phi)[M_1, \ldots, M_k] \rangle\!\rangle}$$

Note that in the second case the substitution θ maps every variable in its domain to a pure quotation.

4 Reduction

The *reFLect* small step reduction semantics is defined as the rewriting system $\rightarrow_R$ induced by the following four kinds of redex-contractum pairs.

R	redex	contractum	conditions
(β)	$(\lambda L.\, M) \cdot N$	$M\theta$	$\theta :\ L \Rrightarrow N$
(γ)	$(\lambda L.\, M + L') \cdot N$	$M\theta$	$\theta :\ L \Rrightarrow N$
(ξ)	$(\lambda L.\, M + L') \cdot N$	$L' \cdot N$	N is a pure quotation L does not match N
(ψ)	$\langle\!\langle \mathcal{C}[\hat{}\langle\!\langle M_1 \rangle\!\rangle \circ \sigma_1, \ldots, \hat{}\langle\!\langle M_k \rangle\!\rangle \circ \sigma_k] \rangle\!\rangle$	$\langle\!\langle (\mathcal{C}\phi)[M_1, \ldots, M_k] \rangle\!\rangle$	$\mathcal{C}$ is a level consistent context $M_1 :\ \tau_1 \quad \cdots \quad M_k :\ \tau_k$ $\phi = (\tau_1, \ldots, \tau_k)/(\sigma_1, \ldots, \sigma_k)$

We say that an expression E' is obtained from E by β-reduction, written $E \rightarrow_\beta E'$, when

(β_1) $E' = E[\pi \mapsto R]$,
(β_2) π is an active non-pattern position in E,
(β_3) $(E|_\pi, R)$ is a β-redex-contractum pair.

The relations $\rightarrow_\gamma, \rightarrow_\xi, \rightarrow_\psi$ are defined analogously. By definition, the reduction relation $\rightarrow_R$ is the union of these four reduction relations.

Remark 7. Clearly, if $E \rightarrow_R E'$ and E is a standard *reFLect* expression, then so is E'. The reduction system $\rightarrow_R$ restricted to the set of standard *reFLect* expressions coincides with the reduction system defined in [7]. Thus, subject reduction and confluence, once established for $\rightarrow_R$, immediately follow for the original *reFLect* system.

Lemma 13. *Let ω be any of the reduction symbols β, γ, ξ, ψ.*

(a) If $E \rightarrow_\omega E'$, then $E[M/x] \rightarrow_\omega E'[M/x]$.
(b) If $M \rightarrow_\omega M'$, then $E[M/x] \rightarrow_\omega^* E[M'/x]$.

Proof. Easy. □

Corollary 2. *If $E \to_\omega E'$, then $E\theta \to_\omega E'\theta$, for every substitution θ.*

Proof. Follows immediately from Lemma 13(a). □

Now we introduce a reduction system $\to_S$ that is closely related to $\to_R$ but is often easier to reason about. For this system we will prove subject reduction and termination and the corresponding results for $\to_R$ will follow immediately since $\to_R \subseteq \to_S^*$ (Lemma 14 below).

By definition, $\to_S$ is the union of five reduction systems $\to_{\beta'}$, $\to_{\beta''}$, $\to_\zeta$, $\to_{\zeta'}$, $\to_\psi$, where $\to_\psi$ is as before and the four new types of redexes are defined as follows.

S	redex	contractum	conditions
(β')	$(\lambda x.\, M) \cdot N$	$M[N/x]$	
(β'')	$(\lambda L.\, M) \cdot N$	$M\theta$	$\theta :\ L \Rightarrow N$ N is a pure quotation
(ζ)	$M + N$	M	
(ζ')	$M + N$	N	

Lemma 14. *If $M \to_R^* N$, then $M \to_S^* N$.*

Proof. Every β-redex is either a β'- or a β''-redex. Every ξ-redex is a ζ'-redex. Finally, every γ-redex has a two-step derivation in $\to_S$ as follows: $(\lambda L.\, M + L') \cdot N \to_\zeta (\lambda L.\, M) \cdot N \to_{\beta''} M\theta$. □

Thus, the power of (β) together with (γ) and (ξ) is matched by the power of (β) together with (ζ) and (ζ'); the last two rules are simpler and symmetric, and that is what makes $\to_S$ a more convenient system to reason about. The reasons for splitting (β) into two rules (β') and (β'') will be clear in the proof of termination (Section 7).

5 Subject Reduction

We prove subject reduction for the reduction system $\to_S$. By Lemma 14, subject reduction for $\to_R$ will immediately follow.

Theorem 1. *Suppose E is a well-formed expression, $\vdash E : \sigma$, and $E \to_S E'$. Then E' is well-formed and $\vdash E' : \sigma$.*

Proof. By Lemma 5, it suffices to prove that for each redex-contractum pair (E, E') with E well-formed, one must have that E' is also well-formed and of the same type as E.

If our redex-contractum pair is of the form (ζ) or (ζ'), the result follows immediately from the typing rule for alternation. If it is of the form (β') or (β''), the result follows from Lemma 9.

The only remaining case is (ψ) and we need to prove

$$\langle\!\langle (\mathcal{C}\phi)[M_1, \ldots, M_k] \rangle\!\rangle : \mathsf{term} \tag{9}$$

assuming

$$\begin{aligned} &M_i : \tau_i \quad \text{for} \quad i = 1, \ldots, k \\ &\phi = (\tau_1, \ldots, \tau_k)/(\sigma_1, \ldots, \sigma_k) \\ &\langle\!\langle \mathcal{C}[\hat{}\langle\!\langle M_1 \rangle\!\rangle \circ \sigma_1, \ldots, \hat{}\langle\!\langle M_k \rangle\!\rangle \circ \sigma_k] \rangle\!\rangle : \mathsf{term} \end{aligned}$$

and assuming also that the expression in the last line is well-formed.

The last assumption and the typing rule for quotations imply

$$\mathcal{C}[z_1^{\sigma_1}, \ldots, z_k^{\sigma_k}] : \sigma$$

for some σ, where the z_i are fresh variables. By Lemma 6,

$$(\mathcal{C}\phi)[z_1^{\tau_1}, \ldots, z_k^{\tau_k}] : \sigma\phi$$

and then by Lemma 5 and Remark 3,

$$(\mathcal{C}\phi)[M_1, \ldots, M_k] : \sigma\phi$$

Now the required judgment (9) follows by the typing rule for quotations (the pure quotation case). □

6 Local Confluence

In *reFLect* as in the standard λ-calculus, local confluence is largely a consequence of coherence of reduction and substitution as stated in Lemma 13. There are a few additional cases that are easily disposed of.

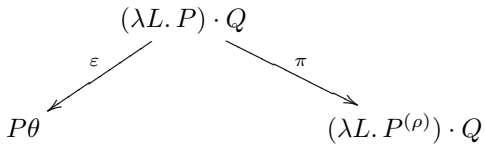
In this section we will write $E^{(\pi)}$ for the result of reducing E at the redex position π. Since (by simple inspection) every redex in the system $\to_R$ uniquely determines the corresponding contractum, the expression $E^{(\pi)}$ is uniquely determined by E and π. (The same is not true for $\to_S$ because of two possibilities for reducing alternations.)

Lemma 15. *Let π and π' be two redexes in a well-formed expression E. Then there exists E' such that $E^{(\pi)} \to_R^* E'$ and $E^{(\pi')} \to_R^* E'$.*

Proof. Confluence is immediate if π and π' are incomparable. In this case, π and π' are redex positions of $E^{(\pi')}$ and $E^{(\pi)}$ respectively, and $E^{(\pi)(\pi')} = E^{(\pi')(\pi)}$.

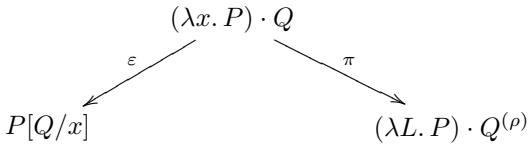
Thus, we can assume that π' is a prefix of π. Clearly, it is no loss of generality to assume also that $\pi' = \varepsilon$, i.e. that E is a redex. We split the proof into four cases, depending on what kind of redex E is. For convenience, we will shorten $\to_R$ to $\to$ in this proof.

Case 1: E is a β-redex. We have $E = (\lambda L.\, P) \cdot Q$ and π is associated with either a position in P, or a position in Q. In the first case, we have $\pi = \mathbf{03}\rho$ for some position ρ in P, and the peak to be resolved has this form:



where θ is the substitution that matches L with Q. The term at the bottom right is a β-redex that reduces to $P^{(\rho)}\theta$. By Corollary 2, we have $P\theta \to P^{(\rho)}\theta$, proving the confluence of our peak.

In the remaining case, we have $\pi = \mathbf{1}\rho$, where ρ is a redex position in Q. We must have now that L is a variable pattern, for otherwise Q would be a pure quotation, hence irreducible. Thus, our peak has the form



Again, the expression at the bottom right is a β-redex; it reduces to $P[Q^{(\rho)}/x]$. On the other hand, Lemma 13(b) implies $P[Q/x] \to^* P[Q^{(\rho)}/x]$, finishing the proof.

Case 2: E is a γ-redex. Similar to Case 1.

Case 3: E is a ξ-redex. We have $E = (\lambda L.\, M + L') \cdot N$, and it ξ-reduces to $L' \cdot N$. Since N must now be a pure quotation, reducing E at π produces an expression E' of the form $(\lambda L.\, M' + L') \cdot N$ or $(\lambda L.\, M + L'') \cdot N$, where $M \to M'$ and $L' \to L''$ respectively. In the first case, E' ξ-reduces to $L' \cdot N$, and in the second case both E' and $L' \cdot N$ reduce to $L'' \cdot N$.

Case 4: E is a ψ-redex. We have $E = \langle\!\langle \mathcal{C}[\hat{}\langle\!\langle M_1 \rangle\!\rangle \circ \sigma_1, \ldots, \hat{}\langle\!\langle M_k \rangle\!\rangle \circ \sigma_k] \rangle\!\rangle$. Since $\mathcal{C}$ is a level consistent context and all the $\langle\!\langle M_i \rangle\!\rangle$ are pure quotations, the only active subexpressions of E are the $\langle\!\langle M_i \rangle\!\rangle$. None of them is a redex, so this case does not arise. $\square$

7 Termination and Confluence

Recall that termination (or strong normalization) of a reduction system means non-existence of infinite reduction sequences. In this section we prove the main result of the paper:

Theorem 2. *The reduction relation $\to_R$ on the set of well-formed reFLect expressions is confluent (satisfies the Church-Rosser property).*

Since we have established local confluence (Lemma 15), by Newman's Lemma (see [3]) it would suffice for the proof of Theorem 2 to prove that $\to_R$ is terminating. This goal, by Lemma 14, will follow from the following.

Theorem 3. *The reduction relation $\to_S$ is terminating.*

We will write $\to_S$ as a union of two simpler rewriting systems and deduce Theorem 3 from the following lemma of Bachmair and Dershowitz.

Lemma 16. *[4] If $\to_A$ and $\to_B$ are terminating reduction systems such that $\to_A \circ \to_B \subseteq \to_B \circ (\to_A \cup \to_B)^*$, then $A \cup B$ is terminating.* □

The relationship between $\to_A$ and $\to_B$ used in Lemma 16 is usually called quasi-commutation. We will use this lemma with $\to_A = \to_{\beta'} \cup \to_\zeta \cup \to_{\zeta'}$ and $\to_B = \to_{\beta''} \cup \to_\psi$. To complete the proof of Theorem 2, we prove:

(1) *Each of the reduction systems $\to_{\beta'}$, $\to_\zeta$, $\to_{\zeta'}$ quasi-commutes with each of $\to_{\beta''}$, $\to_\psi$.*
(2) *$\to_A$ and $\to_B$ are terminating.*

The first property will be proved in the following subsection 7.1. Termination of $\to_B$ will be shown in 7.2. The remaining subsections are devoted to the proof of termination of $\to_A$, which is the most difficult part.

7.1 Quasi-commutation proofs

Suppose $E_1 \to_\omega E_2 \to_{\omega'} E_3$, where ω is either β'' or ψ and ω' is either $\to_{\beta'}$, $\to_\zeta$, or $\to_{\zeta'}$. Suppose also that these two reductions occur at positions π and π' respectively. If none of π and π' is the prefix of the other, then we have simple commutation $E_1 \to_\beta E_2' \to_\omega E_3$, where the first reduction occurs at π and the second at π'. The remaining case splits into two.

Case 1: π is a prefix of π'. Without loss of generality, $\pi = \varepsilon$. We must have $\omega = \beta''$ because if $E_1 \to_\psi E_2$, then E_2 is a pure quotation and so is ω'-irreducible. Thus, we have $E_1 = (\lambda L.\, M) \cdot N$ and $E_2 = M\theta$, where θ substitutes some variables with expressions that are pure quotations. Since pure quotations are irreducible, the ω'-redex position π' in $M\theta$ must also be an ω'-redex position in M. Suppose $\omega' = \beta'$; the remaining two cases when ω' is ζ or ζ' are similar and easier, so will be omitted. If $M|_{\pi'} = (\lambda x.\, P) \cdot Q$, then $(M\theta)|_{\pi'} = (M|_{\pi'})\theta =$

$(\lambda x.\,(P\theta)) \cdot (Q\theta)$ (see Lemmas 10 and 12), and so $E_3 = (M\theta)[\pi' \mapsto (P\theta)[Q\theta/x]]$. Thus, $E_3 = (M[\pi' \mapsto P[Q/x]])\theta$, and therefore $E_1 \to_{\beta'} (\lambda L.\,M[\pi' \mapsto P[Q/x]]) \cdot N \to_{\beta''} E_3$.

Case 2: π' is a prefix of π. Without loss of generality, $\pi' = \varepsilon$. Suppose $\omega' = \beta'$; again, the cases when ω' is ζ or ζ' are similar and will be omitted. We have $E_1 = (\lambda x.\,P) \cdot Q$. Also, $E_2 = (\lambda x.\,P') \cdot Q$ or $E_2 = (\lambda x.\,P) \cdot Q'$, where in the first case $P \to_\omega P'$, and in the second case $Q \to_\omega Q'$. Thus, $E_3 = P'[Q/x]$ or $E_3 = P[Q'/x]$. By Lemma 13, we have $E_1 \to_\beta P[Q/x] \to_\omega^* E_3$ in both cases.

7.2 Termination of $\to_B$

We measure the complexity of an expression E by the pair (m_1, m_2), where m_1 is the number of active application positions in E, and m_2 is the size (total number of positions) of E. Every application of $\to_{\beta''}$ decreases m_1 because the substitution of some variables with pure quotations does not create any new active positions. On the other hand, any application of $\to_\psi$ preserves m_1, but reduces the total expression size.

7.3 System $\to_A$: A Nondeterministic Lambda Calculus

Suppose we restrict the system $\to_A$ to the subset of expressions that do not use any quotations and abstractions with non-variable patterns. The resulting system, call it $\to_{A'}$, can be seen as the extension of the simply typed lambda calculus with the choice operator + constrained by its typing rule as in Box 1 and reduction rules ζ, ζ'. If we could prove $\to_{A'}$ is terminating, we could with little effort derive termination of $\to_A$. (Quotations and abstractions with non-variable patterns would be treated as constant operators whose arguments are maximal active subexpressions.)

There are several versions of nondeterministic lambda calculi in the literature (cf. [5, 6, 9] and the references there), but it seems that the termination problem for them is being considered only in [5]. Unfortunately, the calculus described in [5] does not exactly match $\to_{A'}$, and it is not clear that termination for $\to_{A'}$ can be derived from the results of [5]. Therefore, we decide to give a direct proof of termination of $\to_A$.

7.4 Standard derivations in $\to_A$

Termination of $\to_A$ will be proved by generalizing the standardization technique for the classical λ-calculus, as described in Section 2.2 of [2], which we try to follow closely. We have to deal with the increased complexity caused by the addition of the alternation rules. The requisite generalizations are in most cases straightforward to state and prove.

From now on, we will simply write $\to$ for $\to_A$. For better readibility, we will also write variables without their superscripts. The application operator will (as usual) be assumed to associate to the left.

We will write $E \xrightarrow{\pi} E'$ to indicate that the reduction occurs at the position π. The reduction chains will be called DERIVATIONS, and if D is the derivation $E_0 \xrightarrow{\pi_1} E_1 \xrightarrow{\pi_2} \cdots \xrightarrow{\pi_k} E_k$, we will write $E_0 \xrightarrow{D} E_k$.

Suppose $E \xrightarrow{D} E'$ is a single reduction occuring at the position π. (Note that, because of alternation, D is not uniquely determined by π.) For any redex position μ of E, the set μ/D of RESIDUALS of μ after reduction at π is the set of redexes of E' defined as follows.

$$\mu/D = \begin{cases} \emptyset & \text{if } \mu = \pi \\ \{\mu\} & \text{if } \mu < \pi, \text{ or } \mu \text{ and } \pi \text{ are incomparable} \\ \{\pi\nu\mu' \mid \pi\mathbf{02} \text{ binds } \pi\mathbf{03}\nu \text{ in } E\} & \text{if } \mu = \pi\mathbf{1}\mu' \text{ and } D \text{ is a } \beta'\text{-reduction} \\ \{\pi\mu'\} & \text{if } \mu = \pi\mathbf{4}\mu' \text{ and } D \text{ is a } \zeta\text{-reduction} \\ \{\pi\mu'\} & \text{if } \mu = \pi\mathbf{5}\mu' \text{ and } d \text{ is a } \zeta'\text{-reduction} \end{cases}$$

This definition readily extends to U/D, where U is any set of redexes of E and D is any derivation starting from D:

$$U/D = \bigcup\{\mu/D \mid \mu \in U\} \qquad U/(DD') = (U/D)/D',$$

where the derivation D has length one, and D' is arbitrary.

We will say that a position π occurs TO THE LEFT of another position π', when either $\pi < \pi'$ or there exists a position μ such that $\mu\mathbf{0} < \pi$ and $\mu\mathbf{1} < \pi'$. A LEFTMOST REDEX in E is one for which there are no other redexes that are to the left of it. It is easy to see that every expression that is not irreducible has exactly one leftmost redex. More precisely, the leftmost redex of $L + M$ is ε, and the leftmost redex of $\lambda x.\, M$ is the leftmost redex of M prefixed with $\mathbf{3}$. If L is not irreducible, then its leftmost redex prefixed with $\mathbf{0}$ is the leftmost redex of $L \cdot M$; otherwise, the leftmost redex of $L \cdot M$ is the leftmost redex of M prefixed with $\mathbf{1}$. Variables, quotations, and abstractions with non-variable patterns are, of course, irreducible (in $\to_A$).

By definition, a derivation $D\colon E_0 \xrightarrow{\pi_1} E_1 \xrightarrow{\pi_2} \cdots \xrightarrow{\pi_k} E_k$ is

- NORMAL if every π_i is the leftmost redex in E_{i-1};
- STANDARD if for every i, j, μ such that $i < j$ and μ is to the left of π_i in E_{i-1}, the redex π_j is not a residual of μ (i.e. $\pi_j \notin \mu/D_{ij}$, where D_{ij} is the subderivation of D going from E_{i-1} to E_{j-1}).

We will write $E \xrightarrow{\text{norm}} E'$ and $E \xrightarrow{\text{std}} E'$ when there exists a normal (respectively standard) derivation from E to E'. Note the following easy facts:

- Every normal derivation is also standard.
- If $D = D_1 D_2$ is standard, then D_1 and D_2 are standard.
- If D_1 is normal and D_2 is standard, then $D_1 D_2$ is standard.

Lemma 17. *Every standard derivation $M \xrightarrow{D} \lambda x.\, N$ is of the form $M \xrightarrow{D_1} \lambda x.\, N' \xrightarrow{D_2} \lambda x.\, N$, where D_1 is normal and D_2 is standard.*

Proof. We argue by induction on the length of D. All that needs to be proved is that if M is not an abstraction, then the first reduction in D occurs at the leftmost redex position in M.

Case 1: M is an alternation. The root of M is a redex position, and it eventually gets reduced. Since D is standard, the reduction at the root must happen first.

Case 2: M is an application. Let π be the longest position of M of the form $\mathbf{11}\cdots\mathbf{1}$, and let p be its length. Thus, we have $M = M_0 \cdot M_1 \cdot \ldots \cdot M_n$, where M_0 is not an application. Note that $n > 0$, so we can write $\pi = \pi'$.

Consider first the case when π' is a redex in M. Then it is the leftmost redex, and will remain as residual in all expressions derived from M unless it is reduced in the first step. Thus, the reduction sequence D must begin with π'.

Suppose now π' is not a redex in M. Note that M_0 cannot be a variable or a quotation, because in these cases, every expression that can be derived from M would have to be of the form $M_0 \cdot M_1' \cdot \ldots \cdot M_n'$. The only possibility is that $M_0 = P + Q$ for some P, Q. Now π is the leftmost redex position of M. The derivation D must start at π, because otherwise π will remain residual in all expressions derived from M. □

7.5 Standardization Lemma

Let SN denote the set of all well-formed expressions that are strongly normalizing with respect to our rewriting system $\rightarrow = \rightarrow_A$; i.e. $M \in SN$ when there are no infinite $\rightarrow$-derivations starting from M. Termination means all well-formed expressions are strongly normalizable.

It is well-known that for any $M \in SN$ the derivations starting at M cannot be arbitrarily long; we will write $\delta(M)$ for the maximal possible length of such a derivation. We will also write $|M|$ for the size of M.

Lemma 18. *If $M \in SN$ and $M \rightarrow^* N$, then $M \xrightarrow{\text{std}} N$.*

Proof. Induction on $(\delta(M), |N|)$.

Case 1: $M = \lambda x.\, M'$. We must have $N = \lambda x.\, N'$ and $M' \rightarrow^* N'$, for some N'. Since $\delta(M') = \delta(M)$, but $|M'| < |M|$, we have $M' \xrightarrow{\text{std}} N'$ by induction hypothesis, and so $M \xrightarrow{\text{std}} N$.

Case 2: $M = M_1 + M_2$. If the root never gets reduced during the derivation $M \rightarrow^* N$, then $N = N_1 + N_2$ with $M_1 \rightarrow^* N_1$ and $M_2 \rightarrow^* N_2$. The induction hypthesis applies since $\delta(M_i) \leq \delta(M)$ and $|M_i| < |M|$, so we have $M_1 \xrightarrow{\text{std}} N_1$ and $M_2 \xrightarrow{\text{std}} N_2$. These two derivations combine into a standard derivation $M_1 + M_2 \rightarrow^* N_1 + M_2 \rightarrow^* N_1 + N_2$. (It is easy to check that every inerleaving of standard reduction sequences of M_1 and M_2 results in a standard reduction sequence of $M_1 + M_2$.)

If the root becomes reduced in $M \to^* N$, then $M_1 \to^* N$ or $M_2 \to^* N$ must hold. The induction hypothesis gives $M_i \xrightarrow{\text{std}} N$ (for some i), and so $M \xrightarrow{\text{norm}} M_i \xrightarrow{\text{std}} N$ is a standard derivation.

Case 3: $M = M_1 \cdot M_2$. If the root never gets reduced in our derivation $D\colon M \to^* N$, we are done as in the similar situation in Case 2. Thus, we can assume that D looks like

$$M_1 M_2 \to^* (\lambda x.\, N_1) \cdot N_2 \to N_1[N_2/x] \to^* N,$$

where $M_1 \to^* \lambda x.\, N_1$ and $M_2 \to^* N_2$. By Lemma 17, there exists P such that

$$M_1 \xrightarrow{\text{norm}} \lambda x.\, P \quad \text{and} \quad P \xrightarrow{\text{std}} N_1$$

Thus, we obtain a modified derivation from M to N:

$$M = M_1 \cdot M_2 \xrightarrow{D_1} (\lambda x.\, P) \cdot M_2 \xrightarrow{D_2} P[M_2/x] \xrightarrow{D_3} N_1[N_2/x] \xrightarrow{D_4} N,$$

where D_1 is normal, D_2 is a single step derivation at the root position, so $D_1 D_2$ is normal too. D_3 exists because of $P \xrightarrow{\text{std}} N_1$ and Lemma 13, and D_4 was part of D.

We have $\delta(P[M_2/x]) < \delta(M)$ because $M \xrightarrow{D_1 D_2} P[M_2/x]$ and this derivation has non-zero length. Thus, by induction hypothesis, we have $P[M_2/x] \xrightarrow{\text{std}} N$. Since a standard derivation remains standard when prefixed with a normal one, and since $D_1 D_2$ is normal, we obtain finally the desired $M \xrightarrow{\text{std}} N$. □

Lemma 19. *If $M[N/x] \xrightarrow{\text{std}} \lambda y.\, P$, then one of the following holds:*

- *$M \to^* \lambda y.\, Q$ and $Q[N/x] \to^* P$, for some Q;*
- *$M \to^* M' = x \cdot P_1 \cdot P_2 \cdot \ldots \cdot P_n$ and $M'[N/x] \to^* \lambda y.\, P$*

Proof. In view of Lemma 17, it suffices to prove our lemma assuming that the given standard derivation $M[N/x] \xrightarrow{D} \lambda y.\, P$ is actually normal. For this, we argue by induction on the length of D.

The proof splits according to whether the following property is true or not:

$$\text{The first reduction of } D \text{ occurs at a position of } M \tag{10}$$

If (10) holds, then D begins with $M[N/x] \to M'[N/x]$, where $M \to M'$, and induction works.

Now assume (10) does not hold. This eliminates the possibility that M is an alternation. If M is a variable or an abstraction, we are done. Thus, M is an application, and we can write $M = M' \cdot P_1 \cdot \ldots \cdot P_n$, where $n \geq 1$ and M' is not an application. It is not possible that M' is an alternation, because that alternation would be the leftmost redex in $M[N/x]$, violating the assumption that (10) does not hold. The same assumption would be violated if M' were an abstraction. Thus, M' must be a variable. □

7.6 Termination of $\rightarrow_A$

We argue by induction on the size $|M|$ that $M \in SN$ for every well-formed expression M.

There is only one non-trivial case, where $M = M_1 \cdot M_2$, both M_1 and M_2 are in SN, and we need to prove that $M \in SN$.

Consider a derivation that starts at M. If the root never becomes reduced, then the path is finite by induction hypothesis. So assume $M_1 \rightarrow^* \lambda x.\, N_1$, $M_2 \rightarrow^* N_2$, so that the reduction goes as follows

$$M = M_1 M_2 \rightarrow^* (\lambda x.\, N_1) \cdot N_2 \rightarrow N_1[N_2/x] \rightarrow \cdots$$

We know here that $N_1, N_2 \in SN$, so to complete the proof, it remains to prove the following lemma.

Lemma 20. *If $M, N \in SN$, then $M[N/x] \in SN$.*

Proof. As in [2], the proof will go by induction on $(|type(N)|, \delta(M), |M|)$, where by the length of a type we mean the number of nodes of its tree representation.

The cases when M is a variable, abstraction, alternation, or quotation are all trivial. In the only remaining case we have $M = M_1 \cdot M_2$, and so $M[N/x] = M_1[N/x] \cdot M_2[N/x]$.

Consider a reduction path starting from $M[N/x]$. If the root never becomes reduced, this reduction path is an interleaving of reduction paths of $M_1[N/x]$ and $M_2[N/x]$, and we are done by induction because $\delta(M_i) \leq \delta(M)$, $|M_i| < |M|$, and the same N is being used.

Thus, we can assume $M_1[N/x] \rightarrow^* \lambda y.\, P$ and $M_2[N/x] \rightarrow^* N_2$, and we want to prove $P[N_2/y] \in SN$.

Since $M_1, N \in SN$, it follows that $M_1[N/x] \in SN$. (This can be proved by induction on $|M_1|$.) Invoking Standardization Lemma 18, we have $M_1[N/x] \xrightarrow{\text{std}} \lambda y.\, P$. Lemma 19 presents us with two cases.

Case 1: $M \rightarrow^* \lambda y.\, Q$ and $Q[N/x] \rightarrow^* P$. By Lemma 13, we have $Q[N/x][N_2/y] \rightarrow^* P[N_2/y]$. On the other hand, by Lemma 12 we have

$$Q[N/x][N_2/y] = Q[N/x][M_2[N/x]/y] = Q[M_2/y][N/x] = M'[N/y],$$

where the last equality just introduces the notation M'. Now it suffices to prove that $M'[N/y] \in SN$. This follows from the induction hypothesis, since $M = M_1 \cdot M_2 \rightarrow^* (\lambda y.\, Q) \cdot M_2 \rightarrow M'$, and so $\delta(M') < \delta(M)$.

Case 2: $M \rightarrow M' = x \cdot P_1 \cdot \ldots \cdot P_n$ and $M'[N/x] \rightarrow^* \lambda y.\, P$. Here we prove $|type(N_2)| < |type(N)|$, and $P[N_2/y] \in SN$ will follow from the induction hypothesis. Indeed,

$$|type(N_2)| = |type(M_2)| < |type(M_1)| = |type(M')| < |type(x)| = |type(N)|,$$

where the three equalities should be clear from the context, and the two inequalities follow from $M = M_1 M_2$, $M' = x \cdot P_1 \cdot \ldots \cdot P_n$, well-formedness of all the occurring expressions, and typing rules. □

References

1. M.D. Aagaard, R.B. Jones, and C-J. H. Seger. *Lifted-FL*: A pragmatic implementation of combined model checking and theorem proving. In Y. Bertot, G. Dowek, A. Hirschowitz, and L. Théry, editors, *Theorem Proving in Higher-Order Logics (TPHOLS)*, volume 1690 of *LNCS*, pages 323–340. Springer, 1999.
2. R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*. Cambridge University Press, 1998.
3. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
4. L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In J. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction (CADE)*, volume 230 of *LNCS*, pages 5–20. Springer, 1986.
5. Ph. de Groote. Strong normalization in a non-deterministic typed lambda-calculus. In *Logical Foundations of Computer Science*, pages 142–152, 1994.
6. M. Dezani-Ciancaglini, U. de'Liguoro, and A. Piperno. A filter model for concurrent lambda-calculus. *SIAM J. Comput.*, 27(5):1376–1419, 1998.
7. J. Grundy, J. O'Leary, and T. Melham. A reflective functional language for hardware design and theorem proving. Technical Report PRG-RR-03-16, Oxford University Computing Laboratory, 2003.
8. J. Harrison. Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053, SRI Cambridge, 1995. Available on the Web as `http://www.cl.cam.ac.uk/users/jrh/papers/reflect.dvi.gz`.
9. C.-H. Luke Ong. Non-determinism in a functional setting. In *8th Symposium on Logic in Computer Science (LICS)*, pages 275–286. IEEE Computer Society Press, 1993.