A Notation for Lambda Terms: A Generalization of Environments*

Gopalan Nadathur
Department of Computer Science
Ryerson Hall, 1100 E 58th Street
University of Chicago
Chicago, IL 60637
gopalan@cs.uchicago.edu

Debra Sue Wilson
IBM Corporation
TCJA/B631 D106
4102 S. Miami Blvd.
Research Triangle Park, NC 27706
debwilson@vnet.ibm.com

Abstract

A notation for lambda terms is described that is useful in contexts where the intensions of these terms need to be manipulated. The scheme of de Bruijn is used for eliminating variable names, thus obviating α-conversion in comparing terms. A category of terms is provided that can encode other terms together with substitutions to be performed on them. The notion of an environment is used to realize this 'delaying' of substitutions. However, the precise environment mechanism employed here is more complex than the usual one because the ability to examine subterms embedded under abstractions has to be supported. The representation presented permits a β-contraction to be realized via an atomic step that generates a substitution and associated steps that percolate this substitution over the structure of a term. Operations on terms are provided that allow for the combination and hence the simultaneous performance of substitutions. Our notation eventually provides a basis for efficient realizations of β-reduction and also serves as a means for interleaving steps inherent in this operation with steps in other operations such as higher-order unification. Manipulations on our terms are described through a system of rewrite rules whose correspondence to the usual notion of β-reduction is exhibited and exploited in establishing confluence and other similar properties. Our notation is similar in spirit to recent proposals deriving from the Categorical Combinators of Curien, and the relationship to these is discussed. Refinements to our notation and their use in describing manipulations on lambda terms are considered in a companion paper.

Keywords: Lambda calculus, lambda terms as representation devices, explicit substitution notations, confluence and noetherianity properties.

---

* This paper is a revised version of Technical Report CS-1994-03, Department of Computer Science, Duke University and is to appear in *Theoretical Computer Science*.

1 Introduction

This paper concerns a notation for the terms in a lambda calculus that can serve as a basis for efficient implementations of operations on such terms. Traditionally, lambda terms have been used as a vehicle for performing computations, and the representation of these terms and the design of efficient evaluators for the lambda calculus in this context have received considerable attention. Our interest, however, is in a situation where lambda terms are used as a *representational* device. This interest is motivated primarily by implementation questions pertaining to λProlog, a logic programming language that employs the terms of a typed lambda calculus as its data structures [31]. We believe, however, that this issue is of wider concern, given the number of computer systems and programming languages in existence today that use some variety of the lambda calculus in representing and manipulating formal objects such as formulas, programs and proofs [5, 7, 8, 15, 18, 28, 35, 36].

Lambda terms have been found to be useful as data structures because of their ability to represent naturally the notion of binding that is part of the syntax of several kinds of objects [6, 23, 28, 34, 37]. Consider, for instance, the task of representing the quantified formula $\forall x((p\ x) \vee (q\ x))$ in which p and q are predicate names. Observing that a quantifier plays the dual role of determining a scope and of making a predication, this formula can be rendered fairly transparently into the lambda term $(all\ (\lambda x\,((p\ x)\ or\ (q\ x))))$; in this term, *all* is a constant that represents universal quantification and *or* is an (infix) constant representing disjunction. Using such a representation makes the implementation of several logical operations on formulas relatively straightforward. For example, consider the operation of instantiation. Under the chosen representation, instantiating a 'formula' of the form $(all\ P)$ by t is given simply by the term $(P\ t)$. The actual task of substitution is carried out with all the necessary renamings by the β-reduction operation on lambda terms. As another example, suppose that we wish to determine if a given formula has a certain structure; such an operation would be relevant, for instance, to the construction of a theorem prover. The notion of unifying lambda terms provides a powerful tool for performing such 'template matching'. Thus, consider the term $(all\ (\lambda x\,((P\ x)\ or\ (Q\ x))))$ in which P and Q are variables. This term matches with any formula whose top-level structure is that of a universal quantification over a disjunction and thus 'recognizes' such formulas. In contrast, the term $(all\ (\lambda x\,((P\ x)\ or\ Q)))$ requires also that the second disjunct not contain the quantified variable and thus serves as a sharper discriminator.[1]

Our interest in this paper is in a suitable representation for lambda terms, assuming that they are to be used in the manner outlined above. The intended application obviously places constraints on the kinds of representations that might be considered. For example, the applications of interest generally require the comparison of the structures of lambda terms. The chosen representation

[1]The notion of unification (used in an informal sense here) is intelligible only in the context of certain typed versions of the lambda calculus. We do not discuss the issue of typing explicitly here since the main concerns of this paper are orthogonal to it.

must therefore make this structure readily available. At a more detailed level, the comparison of lambda terms must ignore the particular names used for bound variables. To cater to this need, the representation that is used must permit equality up to α-convertibility to be determined easily. Finally, an operation of obvious importance is β-reduction, and any reasonable representation must enable this to be performed efficiently. For reasons that we discuss in Section 4, the representation that is used must support two requirements relative to this operation: first, it should be possible to perform the substitutions generated by β-contractions in a *lazy* manner and, second, it should be possible to perform β-contractions *under* abstractions as well as to percolate substitutions generated by it into such contexts.

We describe a notation for lambda terms in this paper that provides a basis for meeting these various requirements. The starting point for our notation is a scheme suggested by de Bruijn [3] for eliminating variable names from terms. To provide a means for delaying substitutions, we utilize the notion of an environment. However, a direct use of this device as developed in the context of implementations of functional programming languages is not possible; the complicating factor is the need for performing substitutions and β-contractions under abstractions. The notation we describe embellishes the notion of an environment in a manner designed to overcome this difficulty. At a level of detail, our proposal shares features with the data structures used in [2] in implementing a normalization procedure. However, in a manner akin to other recent proposals deriving from the Categorical Combinators of Curien [1, 10, 13], it has the characteristic of reflecting the idea of an environment into the notation itself. There are two advantages to adopting this course. First, the resulting notation is fine-grained enough to support a wide variety of reduction procedures on lambda terms, and the analysis undertaken here makes it easy to verify the correctness of these procedures. Second, using such a notation makes it possible to intermingle what are traditionally conceived of as steps within β-contraction with other operations such as those needed in higher-order unification [20]. There is, in fact, a concrete realization of the second idea: the notation developed here is actually being used in this fashion in an implementation of λProlog [30].

The remainder of this paper is organized as follows. The next section summarizes prior logical notions that are used in this paper. Section 3 reviews the de Bruijn notation for lambda terms. We describe our notation for lambda terms in Section 4 and also present the rewrite rules that are intended to mimic β-reduction in its context. We then study the properties of our notation. In Section 5 we describe a well founded partial ordering relation on our terms that is useful in establishing termination properties of subsets of our rules and in constructing inductive arguments. In the following section, we analyze a particular subset of our rewrite rules whose purpose is, roughly, that of reducing terms in our notation that encapsulate substitutions into ones in de Bruijn's notation. We show that every sequence of rewritings using these rules eventually produces the anticipated de Bruijn term from any given term in our notation. In Section 7, we examine the correspondence between the usual notion of β-reduction and our system of rewrite rules. We show here that every β-reduction sequence on de Bruijn terms can be mimicked within our notation and, conversely, any rewrite sequence on our terms can be projected onto a β-reduction sequence on the underlying de Bruijn terms. The advantage of our notation can then be appreciated as follows: it defines a β-contraction operation that is a truly atomic and it provides a fine-grained control over the substitution process. In Section 8, we utilize the projection onto de Bruijn terms to show the confluence of our rewrite system. The method of proof we use is similar in spirit to that referred to as the *interpretation method* in [17] and used in [17] and [39] in establishing confluence properties of a combinator calculus. In the concluding section of this paper, we discuss the relationship of our work to that of others, especially that in [1] and [13].

2 Logical preliminaries

We are concerned in this paper with systems for rewriting expressions. Each such rewrite system is specified by a set of rule schemata. A rule schema has the form $l \rightarrow r$ where l and r are expression schemata referred to as the lefthand side and the righthand side of the rule schema, respectively. For example, the system we describe in Section 4 contains the schema:

$$[\![(t_1\ t_2), ol, nl, e]\!] \rightarrow ([\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!]).$$

In these schema, t_1, t_2, ol, nl and e represent metalanguage variables ranging over appropriately defined categories of expressions. Particular rules may be obtained from this schema by suitably instantiating these variables. All our rule schemata satisfy the property that any syntactic variable appearing in the righthand side already appears in the lefthand side.

Given a notion of subexpressions within the relevant expression language, a rule schema defines a relation between expressions as follows: t_1 is related to t_2 by the rule schema if t_2 is the result of replacing some subexpression s_1 of t_1 by s_2 where $s_1 \rightarrow s_2$ is an instance of the schema. We refer to occurrences in expressions of instances of the lefthand side of a rule schema as *redex* occurrences of the schema. The qualification by the rule schema may be omitted if it is clear from the context. Alternatively, a special name may be used to signify the correspondence to the rule schema.

The relation corresponding to a rule schema is referred to as the one that is *generated* by it. The relation generated by a collection of rule schemata is the union of the relations generated by each schema in the collection. Let $\rhd$ denote such a relation. We will usually write $t \rhd r$ to signify that t is related to s by virtue of $\rhd$. The reflexive and transitive closure of $\rhd$ will be denoted by $\rhd^*$, a relation that will, once again, be written in infix form. Intuitively, $t \rhd^* s$ signifies that t can be rewritten to s by a (possibly empty) sequence of applications of the relevant rule schemata. In accordance with this viewpoint, we refer to the relation $\rhd$ as a *rewrite* or *reduction* relation and we say that t $\rhd$-reduces to s if $t \rhd^* s$.

A notion of concern with regard to a rewrite relation $\rhd$ is that of a $\rhd$-*normal form*. An expression t is in this form if there is no expression s such that $t \rhd s$. That is, t contains no redex occurrences of any of the rule schemata that generate $\rhd$. A $\rhd$-normal form *of* an expression r is an expression t such that r $\rhd$-reduces to t and t is in $\rhd$-normal form. The existence and uniqueness of normal forms for expressions are issues that are of interest for a variety of reasons. For example, rewrite rules are often used as a means for computing. Their use in this capacity is meaningful only if the result of performing the computation — the normal form, if it exists — is independent of the method of carrying out the computation. This will be the case if normal forms are unique. In a sense more pertinent to this paper, a collection of rewrite rule schemata is usually intended as a set of equality axioms in a given logical system. Using them to rewrite expressions is useful in this context only if this somehow helps in determining equality. This is indeed the case if a unique normal form exists for every expression: the equality of two expressions can then be determined by reducing them to their normal forms and comparing these.

A rewrite relation $\rhd$ is *noetherian* if and only if there is no infinite sequence of the form $t_1 \rhd t_2 \rhd \cdots \rhd t_n \rhd \cdots$, *i.e.*, if and only if every sequence of rewritings relative to $\rhd$ terminates. If $\rhd$ is noetherian, a $\rhd$-normal form must exist for every expression. In showing that such a form is unique, the notion of *confluence* is useful. The relation $\rhd$ is said to be confluent if, given any expressions t, s_1 and s_2 such that $t \rhd^* s_1$ and $t \rhd^* s_2$, there must be some expression r such that $s_1 \rhd^* r$ and $s_2 \rhd^* r$. Confluence is of interest because of the following proposition whose proof is straightforward.

Proposition 2.1 *If $\rhd$ is a confluent reduction relation, then if a $\rhd$-normal form exists for any expression, it must be unique.*

A rewrite relation $\rhd$ is said to be *locally confluent* if, whenever $t \rhd s_1$ and $t \rhd s_2$ for expressions t, s_1 and s_2, there must be some expression r such that $s_1 \rhd^* r$ and $s_2 \rhd^* r$. Local confluence is related to confluence by the following proposition, a proof for which may be found in [21].

Proposition 2.2 *A noetherian reduction relation is confluent if and only if it is locally confluent.*

In showing that a reduction relation is locally confluent, an observation in [25] that is generalized in [21] may be used. To describe this observation, we need the following definition.

Definition 2.3 An expression t constitutes a *nontrivial overlap* of rule schemata R_1 and R_2 at a subexpression s of t if (a) t is a redex occurrence of R_1, (b) s is a redex occurrence of R_2 and also does not occur within the instantiation of a schema variable when t is matched with R_1, and (c) either s is distinct from t or R_1 is distinct from R_2. Let r_1 be the expression that results from rewriting t using R_1 and let r_2 result from t by rewriting s using R_2. Then the pair $\langle r_1, r_2 \rangle$ is referred to as the *conflict pair* relative to the overlap in question. The conflict pairs of a collection of rule schemata $\mathcal{R}$ is the set of the conflict pairs obtained by considering all possible nontrivial overlaps between the elements of $\mathcal{R}$.

The conflict pairs as defined here constitute all the ground instances of the critical pairs of a rewrite system in the sense of [21]. We use the notion of critical pairs only at a metalanguage level to avoid a consideration of expressions containing variables.

The observation that is critical to showing local confluence is now the following:

Theorem 2.4 *Let $\rhd$ be a reduction relation generated by the collection $\mathcal{R}$ of rule schemata. Then $\rhd$ is locally confluent if and only if for every conflict pair $\langle r_1, r_2\rangle$ of $\mathcal{R}$ there is some expression s such that $r_1 \rhd^* s$ and $r_2 \rhd^* s$.*

Proof. (Adapted from [21].) Only the 'if' part in nontrivial and needs argument. Let t be any expression and let t_1 and t_2 be the result of rewriting, respectively, the subexpressions s_1 and s_2 in t using the members R_1 and R_2 of $\mathcal{R}$. To show that $\mathcal{R}$ is locally confluent, we need to show that there is some expression r such that $t_1 \rhd^* r$ and $t_2 \rhd^* r$. We consider the various possibilities for s_1 and s_2 and show that this must be the case. If s_1 and s_2 appear in disjoint parts of t, this is obvious: there is a 'residue' of s_2 in t_1 and similarly of s_1 in t_2 and a common expression is obtained by rewriting the first of these (in t_1) using R_2 and the second (in t_2) using R_1. So suppose that one of s_1 and s_2 is a subexpression of the other. Without loss of generality, let s_2 be a subexpression of s_1. Now, if s_1 is identical to s_2 and $R_1 = R_2$, then $t_1 = t_2$ and the desired conclusion is immediately reached. If s_2 is a subexpression of a part of s_1 that is matched with a schema variable in R_1, a little additional argument suffices. On the one hand, the rewriting step that produces t_1 will create a finite number of copies of s_2 in t_1 and on the other hand rewriting s_2 produces in t_2 a subexpression s_1' that is still a redex occurrence of R_1. It is easily seen that using R_2 repeatedly to rewrite the copies of s_2 in t_1 and R_1 to rewrite s_1' in t_2 produces a common expression. The only remaining situation is the one where s_2 is a subexpression of s_1 that matches with a part of R_1 distinct from a schema variable and where either s_1 is distinct from s_2 or R_1 is distinct from R_2. However, in this case s_1 constitutes a nontrivial overlap of R_1 and R_2 at s_2. Let r_1 result from rewriting s_1 using R_1 and let r_2 result from s_1 by rewriting the subexpression s_2 using R_2. Then $\langle r_1, r_2\rangle$ constitutes a conflict pair of $\mathcal{R}$ and, by assumption, there is an expression s such that $r_1 \rhd^* s$ and $r_2 \rhd^* s$. Let r be the expression obtained from t by replacing the subexpression s_1 by s. It must then be the case that $t_1 \rhd^* r$ and $t_2 \rhd^* r$.

□

3 The de Bruijn notation

Conventional presentations of the lambda calculus utilize a scheme that requires names for bound (and free) variables (*e.g.* see [19]). This choice is well-motivated from the perspective of human readability but is not well-suited to machine implementations for at least two reasons. First, it is difficult to systematize the care that must be exercised within this notation in preventing the inadvertent capture of free variables in the course of performing substitutions generated by β-reduction. Second, the determination of identity of two terms is complicated by the need to consider renamings for bound variables. The 'nameless' notation proposed by de Bruijn [3] provides an elegant way of dealing with the first problem and it eliminates the second by rendering lambda terms in the conventional notation that differ only in the names of bound variables into a common form. This notation is central to the discussions in this paper and we therefore outline it below.

We begin with the definition of lambda terms in the de Bruijn notation.

Definition 3.1 The collection of *de Bruijn terms*, denoted by the syntactic category $\langle DTerm\rangle$, is given by the rule

$$\langle DTerm\rangle ::= \langle Cons\rangle \mid \#\langle Index\rangle \mid (\langle DTerm\rangle\ \langle DTerm\rangle) \mid (\lambda\, \langle DTerm\rangle)$$

where $\langle Cons\rangle$ is a category corresponding to a predetermined set of constant symbols and $\langle Index\rangle$ is the category of positive numbers. A de Bruijn term of the form (i) $\#i$ is referred to as an *index* or a *variable reference*, (ii) $(\lambda\, t)$ is called an *abstraction* and (iii) $(t_1\ t_2)$ is referred to as an *application*. The subterm or subexpression relation on de Bruijn terms is given recursively as follows: Each term is a subterm of itself. If t is of the form $(\lambda\, t')$, then each subterm of t' is also a subterm of t. If t is of the form $(t_1\ t_2)$, then each subterm of t_1 and of t_2 is also a subterm of t.

A bound variable occurrence within the conventional scheme for writing lambda terms is represented in the de Bruijn notation by an index that counts the number of abstractions between the occurrence and the abstraction binding it. Thus, the term $(\lambda x\, ((\lambda y\, (y\ x))\ x))$ in conventional presentations is written in the de Bruijn notation as $(\lambda\, ((\lambda\, (\#1\ \#2))\ \#1))$. An alternative, more complete, exposition of the correspondence is the following. We think of the *level* of a subterm in a term as the number of abstractions in the term within which the subterm is embedded. We also assume a fixed listing of the free variables with respect to which we can talk of the n-th free variable. Then, a variable reference $\#i$ occurring at level j in a term corresponds to a bound variable if $j \geq i$. Further, in this case, it represents a variable that is bound by the abstraction at level $(j-i)$ within which the variable reference occurs. In the case that $i > j$, the index $\#i$ represents a free variable, and, in fact, the $(i-j)$-th free variable. It is easily seen that lambda terms that are α-convertible in the conventional notation correspond to the same term under this scheme.

An important operation on lambda terms is that of substitution. In the context of the de Bruijn notation, a generalized notion of substitution — that of substituting terms for *all* the free variables — is given by the following definition.

Definition 3.2 Let t be a de Bruijn term and let $s_1, s_2, s_3, \ldots$ represent an infinite sequence of de Bruijn terms. Then the result of simultaneously substituting s_i for the i-th free variable in t for $i \geq 1$ is denoted by $S(t; s_1, s_2, s_3, \ldots)$ and is defined recursively as follows:

(1) $S(c; s_1, s_2, s_3, \ldots) = c$, for any constant c,

(2) $S(\#i; s_1, s_2, s_3, \ldots) = s_i$ for any variable reference $\#i$,

(3) $S((t_1\ t_2); s_1, s_2, s_3, \ldots) = (S(t_1; s_1, s_2, s_3, \ldots)\ S(t_2; s_1, s_2, s_3, \ldots))$, and

(4) $S((\lambda\, t); s_1, s_2, s_3, \ldots) = (\lambda\, S(t; \#1, s'_1, s'_2, s'_3, \ldots))$ where, for $i \geq 1$, $s'_i = S(s_i; \#2, \#3, \#4, \ldots)$.

We shall use the expression $S(t; s_1, s_2, s_3, \ldots)$ as a meta-notation for the term it denotes.

Towards understanding the above definition, we note that within a term of the form $(\lambda\, t)$, the first free variable is actually denoted by the index $\#2$, the second by $\#3$ and so on. This requires, in (4) above, that the indices for free variables within the terms $s_1, s_2, s_3, \ldots$ being substituted into $(\lambda\, t)$ be "incremented" by 1 prior to substitution into t. Further, the index $\#1$ must remain unchanged within t and it is the indices $\#2$,$\#3$,... that must be substituted for.

We will need to consider the effect of cascading substitutions of the above kind. An observation made in [3] is useful in this context. The term denoted by $S(S(t; s_1, s_2, s_3, \ldots); s'_1, s'_2, s'_3, \ldots)$ is produced by first replacing *every* index in t with some term s_i, and then substituting the terms $s'_1, s'_2, s'_3, \ldots$ into the result. Thus the s'_i terms will only be substituted into occurrences of the s_j terms, and the effect of this substitution can be precomputed. This is formalized in the following proposition taken from [3].

Proposition 3.3 *Given de Bruijn terms* $t, s_1, t_1, s_2, t_2, s_3, t_3 \ldots$

$$S(S(t; s_1, s_2, s_3, \ldots); t_1, t_2, t_3, \ldots) = S(t; u_1, u_2, u_3, \ldots)$$

where, for $i \geq 1$, $u_i = S(s_i; t_1, t_2, t_3, \ldots)$.

The substitution operation is useful in defining the notion of $\rhd_\beta$-reduction, also referred to simply as β-reduction.

Definition 3.4 The β-contraction rule schema is the following

$$((\lambda\, t_1)\ t_2) \rightarrow S(t_1; t_2, \#1, \#2, \ldots)$$

where t_1 and t_2 are schema variables for de Bruijn terms. The relation (on de Bruijn terms) generated by this rule schema is denoted by $\rhd_\beta$ and is called β-contraction. An instance of the lefthand side of the rule schema is called a β-redex.

When a β-contraction is performed, the β-redex is replaced by the term which results from substituting t_2 for the first free variable in t_1 and adjusting the remaining indices. In the next section a notation will be introduced which decouples the generation and performance of the substitution by, in essence, moving the meta-notation $S(t_1; t_2, \#1, \#2, \ldots)$ into the term representation. The following theorem states a property of commutativity between β-reduction and the substitution operation that will be useful in analyzing this notation.

Theorem 3.5 *Let $t_0, t_1, t_2, \ldots$ be de Bruijn terms.*

(i) *If $t_0 \rhd_\beta^* t'_0$, then $S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta^* S(t'_0; t_1, t_2, t_3, \ldots)$.*

(ii) *If, for $i \geq 1$, $t_i \rhd_\beta^* t'_i$, then $S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta^* S(t_0; t'_1, t'_2, t'_3, \ldots)$*

Proof. (i) It suffices to show that if $t_0 \rhd_\beta t'_0$ then $S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta S(t'_0; t_1, t_2, t_3, \ldots)$. We do this by an induction on the structure of t_0. Note first that t_0 must be either an abstraction or an application. Suppose t_0 is an abstraction. In particular, let $t_0 = (\lambda\, s)$. Then the redex that is rewritten must be a subterm of s. The desired conclusion now follows from Definition 3.2 and the inductive hypothesis. If t_0 is an application, there are two possibilities. In the first case, t_0 is not the redex rewritten. In this case we again use Definition 3.2 and the inductive hypothesis to reach the desired conclusion. In the other case, t_0 is of the form $((\lambda\, s_1)\ s_2)$ and, correspondingly, t'_0 is the term $S(s_1; s_2, \#1, \#2, \ldots)$. Now, assuming that, for $i \geq 1$, $t'_i = S(t_i; \#2, \#3, \#4, \ldots)$,

$$S(t_0; t_1, t_2, t_3 \ldots) = ((\lambda\, S(s_1; \#1, t'_1, t'_2, \ldots))\ S(s_2; t_1, t_2, t_3, \ldots))$$

But then

$$S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta S(S(s_1; \#1, t'_1, t'_2, \ldots); S(s_2; t_1, t_2, t_3, \ldots), \#1, \#2, \ldots).$$

Using Proposition 3.3,

$$S(S(s_1; \#1, t'_1, t'_2, \ldots); S(s_2; t_1, t_2, t_3, \ldots), \#1, \#2, \ldots) = S(s_1; S(s_2; t_1, t_2, t_3, \ldots), t''_1, t''_2, \ldots),$$

where $t''_i = S(t'_i; S(s_2; t_1, t_2, t_3, \ldots), \#1, \#2, \ldots)$. Noting the definition of t'_i and using Proposition 3.3 again, it can be seen that $t''_i = t_i$. Thus, $S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta S(s_1; S(s_2; t_1, t_2, t_3, \ldots), t_1, t_2, \ldots)$. On the other hand, again using (Proposition 3.3),

$$S(t'_0; t_1, t_2, t_3, \ldots) = S(S(s_1; s_2, \#1, \#2, \ldots); t_1, t_2, t_3, \ldots) = S(s_1; S(s_2; t_1, t_2, t_3, \ldots), t_1, t_2, \ldots).$$

Thus, even in this case, $S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta S(t'_0; t_1, t_2, t_3, \ldots)$.

(ii) The proof is again by induction on the structure of t_0. The constant and index cases are immediate and the application case is handled by a straightforward recourse to Definition 3.2 and the inductive hypothesis. The only remaining case is that when t_0 is of the form $(\lambda\, s)$. In this case

$$S(t_0; t_1, t_2, t_3, \ldots) = (\lambda\, S(s; \#1, u_1, u_2, \ldots))$$

where $u_i = S(t_i; \#2, \#3, \#4, \ldots)$. By (i), for $i \geq 1$, $u_i \rhd_\beta^* S(t'_i; \#2, \#3, \#4, \ldots)$. Using the inductive hypothesis and Definition 3.2, it now follows easily that $S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta^* S(t_0; t'_1, t'_2, t'_3, \ldots)$

□

The following corollary is proved by using Theorem 3.5 twice.[2]

[2]This corollary generalizes a theorem in [3] that is used in proving the Church-Rosser Theorem for β-reduction.

Corollary 3.6 *Let $t_0, t_1, t_2, \ldots$ be de Bruijn terms and, for $i \geq 0$, let $t_i \rhd_\beta^* t_i'$. Then*

$$S(t_0; t_1, t_2, t_3, \ldots) \rhd_\beta^* S(t_0'; t_1', t_2', t_3', \ldots).$$

Finally, we observe the celebrated Church-Rosser Theorem for β-reduction. A proof of it in the context of the de Bruijn notation appears in [3].

Proposition 3.7 *The relation $\rhd_\beta$ is confluent.*

4 Incorporating environments into terms

The de Bruijn notation is useful in contexts where the intensions of lambda terms have to be examined because it makes it unnecessary to consider α-conversion. However, the operation of substitution necessitated by β-contraction is a fairly complex one even within this notation. From a practical perspective, it is useful to obtain some control over this operation and, in particular, to be able to perform it lazily. For instance, consider the task of determining whether the two terms

$$((\lambda\,(\lambda\,(\lambda\,((\#3\ \#2)\ s))))\ (\lambda\,\#1)) \text{ and } ((\lambda\,(\lambda\,(\lambda\,((\#3\ \#1)\ t))))\ (\lambda\,\#1))$$

are equal modulo the rules of λ-conversion; s and t denote arbitrary terms here. It might be concluded that they are not, by observing that these terms reduce to $(\lambda\,(\lambda\,(\#2\ s')))$ and $(\lambda\,(\lambda\,(\#1\ t')))$, where s' and t' result from s and t by appropriate substitutions. Notice that it is enough to determine that the *heads* of these terms are distinct *without* explicitly performing the potentially costly operation of substitution on the arguments. Along a different direction, we observe that the structures of terms have to be traversed while attempting to reduce them to normal forms as well as in performing the substitutions generated by each β-contraction. By delaying substitutions, it may be possible to combine these traversals, thereby leading to gains in efficiency. Thus, consider the term $((\lambda\,((\lambda\,t_1)\ t_2))\ t_3)$ where t_1, t_2 and t_3 represent arbitrary terms. Let t_2' be the result of substituting t_3 for the 'first' free variable in t_2 and decrementing the indices of all the other free variables by one. Now, in reducing the given term to a normal form, it is necessary to substitute t_2' and t_3 for the the first and second free variables in t_1 and to decrement the indices of all the other free variables by two. All these substitutions can be achieved in *one* traversal over the structure of t_1 provided we have a fine-grained control over the way each substitution is carried out. An observation of this kind is, in fact, exploited in the implementation of β-reduction in [2].

In contexts where the lambda calculus is employed as a vehicle for computation, the use of an environment that describes bindings for free variables suffices for delaying substitutions. In situations where the de Bruijn notation is utilized, this device is adequate only because the structure of terms embedded within abstractions need not be explored. Thus, if a term is produced in the course of β-reduction that has an abstraction at the outermost level, then the term may be combined with its environment and returned as a *closure*; this idea is used, for instance, in [9].

However, this assumption is *not* appropriate in contexts where lambda terms are used as a means for representation. As an example, consider again the task of determining whether the two terms

$$((\lambda\,(\lambda\,(\lambda\,((\#3\ \#2)\ s))))\ (\lambda\,\#1)) \text{ and } ((\lambda\,(\lambda\,(\lambda\,((\#3\ \#1)\ t))))\ (\lambda\,\#1))$$

are equal. In ascertaining that they are not, it is necessary to propagate a substitution generated by a β-contraction *under* an abstraction and also to contract β-redexes embedded *inside* abstractions. The idea of an environment cannot be adapted naively to yield a delaying mechanism relative to these requirements. For instance, if a term of the form $((\lambda\,t)\ s)$ is embedded within abstractions, it is to be expected that $(\lambda\,t)$ contains free variables. Hence, if the result of β-contracting this term is to be encoded by the term t and an 'environment', the environment must record not just the substitution of s for the first free variable but also the 'decrementing' of the indices corresponding to all the other free variables. Similar observations can be made about propagating substitutions under abstractions.

While the usual idea of an environment cannot be employed directly, a generalization of this notion suffices even in the context of interest. We describe a notation for lambda terms in this section that incorporates such a generalization into the de Bruijn representation for these terms. Our notation provides a means for capturing the generation of the substitution corresponding to a β-contraction in a truly atomic step. This operation is then combined with rules for 'reading' terms to realize the full effect of the complex substitution operation described in Section 3.

4.1 Informal description of an enhanced notation

Before presenting the details of our notation, we explain the main ideas that underlie it. Our objective is to include a new category of expressions within our terms that will encode 'suspended' forms of substitutions that are to be performed over de Bruijn terms. An encoding of the substitution operation described in Definition 3.2 in its full generality is difficult: this would require the representation in a finite structure of simultaneous substitutions for an *infinite* number of variable references. Fortunately, we need to deal only with the kinds of substitutions that arise through β-contractions and the subsequent propagation of these by virtue of Definition 3.2. Such substitutions exhibit a pattern that can be exploited in providing finite representations for them. In particular, they all have the form $S(t; s_1, s_2, s_3, \ldots)$ where, for some finite $i \geq 1$, it is the case that the sequence $s_i, s_{i+1}, s_{i+2}, \ldots$ is one of consecutive positive integers. The outcome of such a substitution is completely determined by the starting point of this sequence, the terms up to s_i that are *not* part of this sequence and, finally, the term t into which the substitutions are to be performed.

Let us look at the particular kind of situations that are to be treated to understand how exactly these items of information may be recorded. In the simplest case, the task is that of encoding the alterations that must be made to the variable references within a term t to account for the rewriting of a β-redex inside whose 'left' subterm t is embedded. Thus, suppose that the β-redex we wish to rewrite is

$$((\lambda \ldots (\lambda \ldots (\lambda \ldots t \ldots) \ldots) \ldots) \; s);$$

we have elided much of the term in this depiction, indicating only those aspects of its shape that are relevant to the present discussion. Rewriting this term produces a term of the form

$$(\ldots (\lambda \ldots (\lambda \ldots t' \ldots) \ldots).$$

Our goal is to provide a means for representing of the term t' that appears in this expression as the term t together with the substitutions that are to be performed on it.

The variable references within t can be factored into two groups: those that correspond to free variables relative to the given β-redex (but that may possibly be bound in a larger context), and those that correspond to variables bound by one of the abstractions contained within the β-redex. Given a term in a particular context, let us refer to the number of abstractions enclosing that term as its *embedding level*. For example, assuming that every abstraction within the displayed β-redex has been explicitly depicted, the embedding level of t relative to this β-redex is 3. Rewriting a β-redex eliminates an abstraction and thus changes the embedding level for t; in the particular case considered, this becomes 2. Let us refer to the embedding levels before and after the rewriting step as the old and new embedding levels and let us denote them by ol and nl respectively. Now, the variable references in t that are in the first group are precisely those of the form $\#i$ where $i > ol$.[3] Further, these references need to be rewritten to $\#j$ where $j = (i - ol) + nl$ to reflect the fact that they are now free variables relative to a new embedding level. Thus, recording the old and new embedding levels with t determines both the variable references in the first group and the substitutions that must be made for them.

The variable references in the second group are finite in number and substitutions for them can be recorded in an environment. To use a concrete syntax, the term t' in the situation considered might be represented by an expression of the form $[\![t, ol, nl, e]\!]$, where e encodes the appropriate environment. Note that the number of entries in this environment must be identical to the old embedding level. At a level of detail, the environment can be maintained as a list whose elements are in reverse order to the (old) embedding level of the abstractions they correspond to. The virtue of using this order is that the substitution pertaining to the variable reference $\#i$ is given by the i-th element of the list. The environment must, in general, contain information pertaining to two different kinds of abstractions: those that persist in the new term and those that disappear as a result of a β-contraction. The information present must suffice, in the first case, for computing a new value for a variable reference bound by the relevant abstraction and, in the second case, for determining the term to replace it with. One quantity that needs to be maintained in either

[3] This assumes, of course, that the variable reference is not embedded within further abstractions in t. This assumption is dispensed with by considering the old and new embedding levels at the variable reference *occurrence*.

situation is the new embedding level at the relevant abstraction. (For abstractions that persist, we intend this to be the new embedding level just within the scope of the abstraction.) We refer to this quantity as the *index* of the corresponding element of the environment and note that certain 'consistency' properties must hold over the list of indices of environment elements: they must form a non-increasing sequence and none of them should be greater than the new embedding level at the term into which the substitutions are being made. Now, for an abstraction that is not eliminated by a β-contraction, the index is the *only* information that needs to be retained in the environment: the new value of a variable reference corresponding to this abstraction can be calculated as one greater than the difference between this index and the new embedding level at the variable reference. At a concrete level, this information can be recorded through an entry of the form $@l$ where $l+1$ is the value of the index. For an abstraction that disappears due to a β-contraction, it suffices to maintain an entry of the form (s, l) where s is a term and l is the index. Such an entry signals that a variable reference that corresponds to it is to be replaced by s. However, the indices corresponding to some of the free variables in s may have to be renumbered. The particular interpretation is that s is a term that used to appear at an embedding level of l, but is now to be inserted at the (new) embedding level nl. The actual term to be substituted in is, therefore, given by the expression $[\![s, 0, (nl - l), nil]\!]$ in which nil represents the empty environment.

The enhanced syntax for terms that we have outlined up to this point can be used to realize β-contraction through a genuinely atomic step. For example, suppose we wish to rewrite the β-redex $((\lambda\, t)\ s)$. Such a rewriting might consist of producing the term $[\![t, 1, 0, (s, 0) :: nil]\!]$; an environment whose first element is et and whose remaining elements are given by e is denoted here by the expression $et :: e$. We refer to a term of this kind as a *suspension* to indicate that it encodes a substitution that has yet to be computed. In calculating the de Bruijn term that corresponds to this term, it is necessary to 'push' the suspended substitution over the structure of t. We have already indicated how this is to be done in the case that t is a variable reference. The case when t is a constant is also easily handled. If t is a term of the form $(t_1\ t_2)$, the substitution can be distributed over t_1 and t_2 by generating the term $([\![t_1, 1, 0, (s, 0) :: nil]\!]\ [\![t_2, 1, 0, (s, 0) :: nil]\!])$. Finally, in the case that t is of the form $(\lambda\, t_1)$, the suspended substitution can be lowered into the abstraction by generating the term $(\lambda\, [\![t_1, 2, 1, @0 :: (s, 0) :: nil]\!])$. It is interesting to contrast the treatment of abstraction here with that in Definition 3.2. We note specifically that our scheme does not renumber the variable references in the terms in the environment each time an abstraction is descended into but, rather, does this in one swoop when actual replacements are performed.

In the above discussion, we have implicitly assumed that t is a de Bruijn term in a term of the form $[\![t, ol, nl, e]\!]$. However, it is possible for t to itself be a suspension. One approach to dealing with this situation is that we first expose a top-level structure for t that is akin to that of a de Bruijn term and then attempt to propagate the outer substitution over this. While this approach suffices for simulating β-reduction, it does not allow for the combination of substitution walks.

To understand this, let us reconsider the reduction of the term $((\lambda\,((\lambda\,t_1)\ t_2))\ t_3)$ to normal form. Two β-redexes have been exhibited in this term, and the rewriting of the inner one of these can be carried out either before or after the substitution generated by rewriting the outer one has been propagated over it. Depending on the order chosen (and assuming only the minimal propagation of substitutions) we obtain one of the two terms

$[\![[\![t_1, 1, 0, (t_2, 0) :: nil]\!], 1, 0, (t_3, 0) :: nil]\!]$ or
$[\![[\![t_1, 2, 1, @0 :: (t_3, 0) :: nil]\!], 1, 0, ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: nil]\!]$.

Reducing either of these terms to a de Bruijn term based on the approach just suggested is tantamount to substituting t_3 and (a possibly modified version of) t_2 into t_1 in two separate walks.

In order to support the combination of substitution walks, it is necessary to provide a means for rewriting a term of the form $[\![[\![t, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!]$ into one of the form $[\![t, ol', nl', e']\!]$. Notice that e' here represents a 'merging' of the environments e_1 and e_2. In determining the exact shape of the new term, it is important to observe that e_1 and e_2 represent substitutions for overlapping sequences of abstractions within which t is embedded. The generation of the two suspensions can, in fact, be visualized as follows: first, a walk is made over ol_1 abstractions immediately enclosing t, recording substitutions for each of them and leaving behind nl_1 enclosing abstractions. Then a walk is made over ol_2 abstractions immediately enclosing the suspension $[\![t_1, ol_1, nl_1, e_1]\!]$ in the new term, recording substitutions for each of them in e_2 and leaving behind nl_2 abstractions. Notice that the ol_2 abstractions relevant to the second walk is coextensive with some final segment of the nl_1 abstractions left behind after the first walk and includes additional abstractions if $ol_2 > nl_1$.

Based on the image just evoked, it is not difficult to see what ol' in the term representing the combination of the two suspensions, should be: these suspensions together represent a walk over ol_1 enclosing abstractions in the case that $ol_2 \leq nl_1$ and $ol_1 + (ol_2 - nl_1)$ abstractions otherwise and, clearly, ol' should be the appropriate one of these values. In a similar fashion, it can be observed that the number of abstractions eventually left behind is nl_2 or $nl_2 + (nl_1 - ol_2)$ depending on whether or not $nl_1 \leq ol_2$, and this determines the value of nl'.

Thus, only the structure of the merged environment e' remains to be described. We denote this environment by the expression $\{\!\{e_1, nl_1, ol_2, e_2\}\!\}$ to indicate the components of the inner and outer suspensions that determine its value. Notice that, in this expression, the 'length' of e_2 is exactly ol_2 and the indices of the elements of e_1 are bounded by nl_1. Now, e' has a length at least that of e_1 and its length is greater than this only if $ol_2 > nl_1$. In the case that its length is greater than ol_1, its elements beyond the ol_1-th one are exactly the last $(ol_2 - nl_1)$ elements of e_2. As for the first ol_1 elements of e', these must be the ones in e_1 modified to take into account the substitutions encoded in e_2. To understand the precise shape of these elements, suppose that e_1 has the form $et :: e_1'$. The first element of the merged environment will then be a modified form of et that we will write as $\langle\!\langle et, nl_1, ol_2, e_2\rangle\!\rangle$ to indicate, once again, the components determining its value. By the *abstraction height* of et let us mean the difference between nl_1 and the index of et. Let this quantity be h in the present context. A little thought reveals the following: et represents a substitution in e_1 for an abstraction that lies within the scope of those scanned in generating the substitutions in e_2 only if h is less than ol_2. Thus, only when this condition is satisfied must et be changed before inclusion in the merged environment. The nature of the change depends on the kind of element et is. If it is of the form $@l$, then it corresponds to an abstraction that persists after the walk that generates the suspension $[\![t, ol_1, nl_1, e_1]\!]$ and the substitution for this abstraction in the merged environment must be the one contained for it in the environment e_2. However, the index of this entry from e_2 will have to be 'normalized' if the merged environment represents substitutions for a longer sequence of abstractions than does the outer abstraction. This is true exactly when nl_1 is greater than ol_2 and, in this case, the index of the entry must be increased by $nl_1 - ol_2$. If et is an element of the form (t, l), then it represents a component of e_1 that is obtained from rewriting a β-redex that is within the scope of the outermost $ol_2 - h$ abstractions considered in generating e_2. Removing the first h elements from e_2 produces an environment that encodes substitutions for these abstractions in the outer suspension. Let us denote this truncated part of e_2 by e_h and let the index of the first entry in it be l'. The 'term' component of the relevant entry in the merged environment must obviously be t modified by the substitutions in e_h and is, in fact, given precisely by the expression $[\![t, ol_2 - h, l', e_h]\!]$. Finally, it is easily observed that the index of this entry should be l', normalized as before in the case that nl_1 is greater than ol_2.

We provide a concrete illustration of the combination of suspensions by considering the term

$$[\![[\![t_1, 1, 0, (t_2, 0) :: nil]\!], 1, 0, (t_3, 0) :: nil]\!]$$

that results through β-contraction from the term $((\lambda\,((\lambda\,t_1)\;t_2))\;t_3)$. Based on the above discussions, this term might be denoted by the expression $[\![t_1, 2, 0, \{\!\{(t_2, 0) :: nil, 0, 1, (t_3, 0) :: nil\}\!\}]\!]$ in which the precise shape of the merged environment has to be spelled out. The length of this environment is obviously 2 and its second element must be identical to $(t_3, 0)$, the first element of the outer environment. The first element is, on the other hand, given by the value of $\langle\!\langle (t_2, 0), 0, 1, (t_3, 0) :: nil\rangle\!\rangle$. Now, the abstraction height of $(t_2, 0)$ is 0 and so the term component of the value of $\langle\!\langle (t_2, 0), 0, 1, (t_3, 0) :: nil\rangle\!\rangle$ should be $[\![t_2, 1, 0, (t_3, 0) :: nil]\!]$; intuitively, the effect of the entire outer environment must be reflected on t_2 in computing the relevant term in the merged environment. The index of this environment element must be identical to that of $(t_3, 0)$. Thus, the merged suspension may be written out in detail as

$$[\![t_1, 2, 0, ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: (t_3, 0) :: nil]\!].$$

We had observed earlier that the term $((\lambda\,((\lambda\,t_1)\;t_2))\;t_3)$ could also have been rewritten to

$$[\![[\![t_1, 2, 1, @0 :: (t_3, 0) :: nil]\!], 1, 0, ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: nil]\!].$$

Merging the two environments in this term produces the same term as that obtained through the reduction sequence considered earlier, as the reader is invited to verify.

In our discussions of the combination of suspensions, we have acted as though the objective is to calculate the final merged form in one step. Adopting this viewpoint is useful in presenting the intuition governing the computation but, because of the complexity of the merging process, runs counter to our overarching goal of providing a fine-grained control over β-reduction and substitution. The actual notation that we describe corrects this situation by permitting the merging computation to be broken up into a sequence of atomic steps that can be intermingled with other computations on the term. It may be useful to "compile" a sequence of such steps into a larger step that is easy to carry out and that has practical benefits such as providing for the combination of substitution walks. A compilation of this kind can be achieved through the identification of derived or admissible rules for our notation. This matter is discussed in [29].

4.2 A modified syntax for terms

At a formal level, the main addition to the syntax of de Bruijn terms that yields our notation is that of a suspension. In presenting this category of terms, it is necessary to also explain the structure of environments and environment terms. The syntax of these various expressions is given as follows:

Definition 4.1 The categories of suspension terms, environments and environment terms, denoted by $\langle STerm\rangle$, $\langle Env\rangle$ and $\langle ETerm\rangle$, are defined by the following syntax rules:

$$\begin{array}{lcl}
\langle STerm\rangle & ::= & \langle Cons\rangle \mid \#\langle Index\rangle \mid (\langle STerm\rangle\ \langle STerm\rangle) \mid \\
 & & (\lambda\,\langle STerm\rangle) \mid [\![\langle STerm\rangle, \langle Nat\rangle, \langle Nat\rangle, \langle Env\rangle]\!] \\
\langle Env\rangle & ::= & nil \mid \langle ETerm\rangle :: \langle Env\rangle \mid \{\!\{\langle Env\rangle, \langle Nat\rangle, \langle Nat\rangle, \langle Env\rangle\}\!\} \\
\langle ETerm\rangle & ::= & @\langle Nat\rangle \mid (\langle STerm\rangle, \langle Nat\rangle) \mid \langle\!\langle \langle ETerm\rangle, \langle Nat\rangle, \langle Nat\rangle, \langle Env\rangle\rangle\!\rangle.
\end{array}$$

We assume that $\langle Cons\rangle$ and $\langle Index\rangle$ are as in Definition 3.1 and that $\langle Nat\rangle$ is the category of natural numbers. We refer to the expressions described by these rules collectively as *suspension expressions*.

The class of suspension terms obviously includes all the de Bruijn terms. By an extension of terminology, we shall refer to suspension terms of the form $\#i$, $(\lambda\, t)$ and $(t_1\ t_2)$ as indices or variable references, abstractions and applications, respectively. The qualification 'suspension' applied to our terms and expressions is intended to distinguish them from similar notions in the context of the de Bruijn notation. We shall henceforth drop this qualification assuming that we are talking about terms and expressions in the new notation unless otherwise stated.

Definition 4.2 The *immediate subexpression(s)* of an expression x are given as follows:

(1) If x is a term, then if (a) x is $(t_1\ t_2)$, these are t_1 and t_2, (b) if x is $(\lambda\, t)$, this is t, and (c) if x is $[\![t, ol, nl, e]\!]$, these are t and e.

(2) If x an environment, then (a) if x is $et :: e$, these are et and e, and (b) if x is $\{\!\{e_1, i, j, e_2\}\!\}$, these are e_1 and e_2.

(3) If x is an environment term, then (a) if x is (t, l), then this is t, and (b) if x is $\langle\!\langle et, i, j, e\rangle\!\rangle$, then these are et and e.

The *subexpressions* of an expression are the expression itself and the subexpressions of its immediate subexpressions. We sometimes use the term *subterm* when the subexpression in question is a term. A *proper* subexpression of an expression x is any subexpression distinct from x.

The syntax of environments and environment terms includes forms of expressions that are useful in capturing the merging of suspensions. In analyzing the properties of our notation it will often be convenient to exclude such expressions and consider only those environments that correspond transparently to a *list* of bindings. This class of expressions is identified by the following definition.

Definition 4.3 A *simple expression* is an expression that does not have subexpressions of the form $\langle\!\langle et, j, k, e\rangle\!\rangle$ or $\{\!\{e_1, j, k, e_2\}\!\}$. If the expression in question is a term, an environment or an environment term, it may be referred to as a simple term, a simple environment or a simple environment term, respectively. Note that a simple environment e is either nil or of the form $et_1 :: et_2 :: \ldots :: et_n :: nil$. In the latter case, for $1 \leq i \leq n$, we write $e[i]$ to denote et_i; observe that $e[i]$ must itself be of the form $@l$ or (t, l). Further, for $1 \leq j \leq n$, we write $e\{j\}$ to denote the environment $et_j :: \ldots :: et_n :: nil$.

An expression of the form $\{\!\{e_1, i, j, e_2\}\!\}$ encodes the merging of the environments e_1 and e_2. This environment has at least as many elements as e_1 has and may have more if the number of abstractions considered in generating e_2 is greater than i, the count of the abstractions left behind after the generation of e_1. The following definition is, thus, an obvious formalization of a familiar notion. The symbol $\dot{-}$ used in it denotes the subtraction operation on natural numbers.

Definition 4.4 The length of an environment e, denoted by $len(e)$, is given as follows: (a) if e is nil then $len(e) = 0$; (b) if e is $et :: e'$ then $len(e) = len(e') + 1$; and (c) if e is $\{\!\{e_1, i, j, e_2\}\!\}$ then $len(e) = len(e_1) + (len(e_2) \dot{-} i)$.

By the l-th index of an environment we intend to denote the index of the l-th element of the environment if it has such an element and the quantity 0 otherwise. We make this notion as well as that of the index of an environment term precise below. The details of this definition as they relate to expressions of the form $\{\!\{e_1, i, j, e_2\}\!\}$ and $\langle\!\langle et, i, j, e\rangle\!\rangle$ are a reflection of the simple environments and environment terms that they are intended to correspond to.

Definition 4.5 The index of an environment term et, denoted by $ind(et)$, and, for each natural number l, the l-th index of an environment e, denoted by $ind_l(e)$, are defined simultaneously by structural induction on expressions as follows[4]:

(i) If et is $@m$ then $ind(et) = m + 1$.

(ii) If et is (t', m) then $ind(et) = m$.

(iii) If et is $\langle\langle et', j, k, e\rangle\rangle$, let $m = (j \dot{-} ind(et'))$. Then

$$ind(et) = \begin{cases} ind_m(e) + (j \dot{-} k) & \text{if } len(e) > m \\ ind(et') & \text{otherwise.} \end{cases}$$

(iv) If e is nil then $ind_l(e) = 0$.

(v) If e is $et :: e'$ then $ind_0(e) = ind(et)$ and $ind_{l+1}(e) = ind_l(e')$.

(vi) If e is $\{\{e_1, j, k, e_2\}\}$, let $m = (j \dot{-} ind_l(e_1))$ and $l_1 = len(e_1)$. Then

$$ind_l(e) = \begin{cases} ind_m(e_2) + (j \dot{-} k) & \text{if } l < l_1 \text{ and } len(e_2) > m \\ ind_l(e_1) & \text{if } l < l_1 \text{ and } len(e_2) \leq m \\ ind_{(l-l_1+j)}(e_2) & \text{if } l \geq l_1. \end{cases}$$

The index of an environment, denoted by $ind(e)$, is $ind_0(e)$.

In our informal discussions, we had noted certain constraints that are satisfied by suspension expressions when these are used in the intended fashion. These constraints will be useful in later analysis and we therefore formulate them as wellformedness conditions on our expressions.

Definition 4.6 An expression is well formed if the following conditions hold of every subexpression s of the expression:

(i) If s is of the form $[\![t, ol, nl, e]\!]$ then $len(e) = ol$ and $ind(e) \leq nl$.

(ii) If s is of the form $et :: e$ then $ind(e) \leq ind(et)$.

(iii) If s is of the form $\langle\langle et, j, k, e\rangle\rangle$ then $len(e) = k$ and $ind(et) \leq j$.

(iv) If s is of the form $\{\{e_1, j, k, e_2\}\}$ then $len(e_2) = k$ and $ind(e_1) \leq j$.

The following additional constraint on environments is a consequence of the ones in Definition 4.6.

[4]For environment terms and environments that are well formed in the sense of Definition 4.6, the $\dot{-}$ operation in the definitions of m that appear in items (iii) and (vi) in this definition may be replaced by simple subtraction.

$$(\beta_s) \qquad ((\lambda\, t_1)\; t_2) \rightarrow [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$$

Figure 1: The β_s-contraction rule schema

Lemma 4.7 *Let e be a well formed environment. Then $ind_i(e) \geq 0$. Further, for $i \geq len(e)$, $ind_i(e) = 0$. Finally, for any natural numbers i, j such that $i < j$, it is the case that $ind_i(e) \geq ind_j(e)$.*

Proof. By an induction on the structure of e, using Definition 4.5. The details are straightforward and hence omitted.

□

We henceforth consider only well formed expressions and this qualification is assumed implicitly whenever we speak of terms, environments, environment terms or expressions.

4.3 Rules for rewriting expressions

Suspensions, as we have explained informally, are intended to provide for a laziness in the substitution operation needed in β-contraction. This understanding is now formalized through the presentation of a suitable collection of rewrite rules. We divide these rules into three categories in this presentation: the *β_s-contraction* rules that generate suspensions, the *reading* rules that propagate suspended substitutions over terms and the *merging* rules that enable the combination of suspensions. Rules in each of these categories are obtained from the schemata that appear in Figures 1–3, respectively. The following tokens, used in these schemata perhaps with subscripts or superscripts, are to be interpreted as schema variables for the indicated syntactic categories: c for constants, t for terms, et for environment terms, e for environments, i and j for positive numbers and ol, nl, l, m and n for natural numbers. The applicability of several of the rule schemata are dependent on 'side' conditions that are presented together with them. Further, in determining the relevant instance of the righthand side of some of the rule schemata, simple arithmetic operations may have to be performed on components of the expression matching the lefthand side. In the discussions that follow, we shall often include these arithmetic operations within the expression being written. Using this convention, rule (m1) in Figure 3 may also be written as

$$[\![[\![t_1, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \rightarrow [\![t_1, ol_1 + (ol_2 \dot{-} nl_1), nl_2 + (nl_1 \dot{-} ol_2), \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!].$$

Given the syntax of expressions, this convention is really an abuse of notation. However, this abuse is harmless and unambiguous and is, in addition, extremely convenient.

Definition 4.8 The reduction relations generated by the rule schemata in Figure 1, 2 and 3 are denoted by $\rhd_{\beta_s}$, $\rhd_r$ and $\rhd_m$ respectively. The union of the relations $\rhd_r$ and $\rhd_m$ is denoted by $\rhd_{rm}$, the union of $\rhd_r$ and $\rhd_{\beta_s}$ by $\rhd_{r\beta_s}$ and the union of $\rhd_r$, $\rhd_m$ and $\rhd_{\beta_s}$ by $\rhd_{rm\beta_s}$.

(r1) $[\![c, ol, nl, e]\!] \rightarrow c$,
provided c is a constant.

(r2) $[\![\#i, 0, nl, nil]\!] \rightarrow \#j$,
where $j = i + nl$.

(r3) $[\![\#1, ol, nl, @l :: e]\!] \rightarrow \#j$,
where $j = nl - l$.

(r4) $[\![\#1, ol, nl, (t, l) :: e]\!] \rightarrow [\![t, 0, nl', nil]\!]$,
where $nl' = nl - l$.

(r5) $[\![\#i, ol, nl, et :: e]\!] \rightarrow [\![\#i', ol', nl, e]\!]$,
where $i' = i - 1$ and $ol' = ol - 1$, provided $i > 1$.

(r6) $[\![(t_1\ t_2), ol, nl, e]\!] \rightarrow ([\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!])$.

(r7) $[\![(\lambda\, t), ol, nl, e]\!] \rightarrow (\lambda\, [\![t, ol', nl', @nl :: e]\!])$,
where $ol' = ol + 1$ and $nl' = nl + 1$.

Figure 2: Rule schemata for reading suspensions

(m1) $[\![[\![t, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \rightarrow [\![t, ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]$,
where $ol' = ol_1 + (ol_2 \dot{-} nl_1)$ and $nl' = nl_2 + (nl_1 \dot{-} ol_2)$.

(m2) $\{\!\{nil, nl, 0, nil\}\!\} \rightarrow nil$.

(m3) $\{\!\{nil, nl, ol, et :: e\}\!\} \rightarrow \{\!\{nil, nl', ol', e\}\!\}$,
where $nl, ol \geq 1$, $nl' = nl - 1$ and $ol' = ol - 1$.

(m4) $\{\!\{nil, 0, ol, e\}\!\} \rightarrow e$.

(m5) $\{\!\{et :: e_1, nl, ol, e_2\}\!\} \rightarrow \langle\!\langle et, nl, ol, e_2\rangle\!\rangle :: \{\!\{e_1, nl, ol, e_2\}\!\}$.

(m6) $\langle\!\langle et, nl, 0, nil\rangle\!\rangle \rightarrow et$.

(m7) $\langle\!\langle @n, nl, ol, @l :: e\rangle\!\rangle \rightarrow @m$,
where $m = l + (nl \dot{-} ol)$, provided $nl = n + 1$.

(m8) $\langle\!\langle @n, nl, ol, (t, l) :: e\rangle\!\rangle \rightarrow (t, m)$,
where $m = l + (nl \dot{-} ol)$, provided $nl = n + 1$.

(m9) $\langle\!\langle (t, nl), nl, ol, et :: e\rangle\!\rangle \rightarrow ([\![t, ol, l', et :: e]\!], m)$
where $l' = ind(et)$ and $m = l' + (nl \dot{-} ol)$.

(m10) $\langle\!\langle et, nl, ol, et' :: e\rangle\!\rangle \rightarrow \langle\!\langle et, nl', ol', e\rangle\!\rangle$,
where $nl' = nl - 1$ and $ol' = ol - 1$, provided $nl \neq ind(et)$.

Figure 3: Rule schemata for merging suspensions

The legitimacy of the above definition is dependent on our rewrite rules producing well formed expressions from well formed expressions. The following sequence of observations culminating in Theorem 4.12 establishes this fact.

Lemma 4.9 *If e_1 is an environment and $e_1 \rhd_{rm\beta_s} e_2$ then $len(e_1) = len(e_2)$.*

Proof. Let e_1 be an environment. Then the following fact is easily established by induction on the structure of e_1: if x_1 is a subexpression of e_1 and x_2 is an expression of the same type as x_1 such that $len(x_1) = len(x_2)$ in the case that x_1 is an environment, and if e_2 is obtained from e_1 by replacing x_1 by x_2, then $len(e_1) = len(e_2)$. The desired conclusion would then follow if whenever x_1 is an environment and $x_1 \rightarrow x_2$ is an instance of one of the rule schemata in Figures 1–3, then $len(x_1) = len(x_2)$. This can be seen to be the case by inspecting the relevant schemata, namely (m2), (m3), (m4) and (m5).

□

Lemma 4.10 *Let $x_1 \rightarrow x_2$ be an instance of some schema in Figures 1–3. If x_1 is an environment term then $ind(x_1) = ind(x_2)$. If x_1 is an environment, then, for every natural number l, $ind_l(x_1) = ind_l(x_2)$.*

Proof. By a routine inspection of the relevant rule schemata, namely (m2)–(m10).

□

Lemma 4.11 *If x_1 is an environment term or an environment and $x_1 \rhd_{rm\beta_s} x_2$, then $ind(x_1) = ind(x_2)$.*

Proof. Let x_1 and x_2 both be environment terms or environments with the following property: if x_1 is an environment term then $ind(x_1) = ind(x_2)$ and if x_1 is an environment then, for every natural number l, $ind_l(x_1) = ind_l(x_2)$. The following facts are easily established by a simultaneous induction on the structure of expressions: If y_1 is an environment term with x_1 as a subexpression and y_2 results from y_1 by replacing x_1 by x_2, then $ind(y_1) = ind(y_2)$. If y_1 is an environment instead and y_2 results from it by a similar replacement, then, for every natural number l, $ind_l(y_1) = ind_l(y_2)$. The desired conclusion now follows easily from Lemma 4.10.

□

Theorem 4.12 *Let x be a well formed expression and let y be such that $x \rhd_r y$, $x \rhd_m y$, $x \rhd_{\beta_s} y$, $x \rhd_{rm} y$, $x \rhd_{r\beta_s} y$ or $x \rhd_{rm\beta_s} y$. Then y is a well formed expression.*

Proof. It is sufficient to show that this property holds if $x \rhd_{rm\beta_s} y$. Given Lemmas 4.9 and 4.11, this would be true if whenever x_1 is a well formed expression and $x_1 \rightarrow x_2$ is an instance of some schema in Figures 1–3, then x_2 is well formed. This is verified by an inspection of the relevant schemata. The argument is routine in all cases except those of schemata (m1) and (m5). In the case of (m1), *i.e.*, when the rule is

$$[\![[\![t_1, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!] \rightarrow [\![t, ol_1 + (ol_2 \dot{-} nl_1), nl_2 + (nl_1 \dot{-} ol_2), \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!],$$

some care is needed in verifying that $ind(\{\!\{e_1, nl_1, ol_2, e_2\}\!\}) \leq nl_2 + (nl_1 \dot{-} ol_2)$. In the case when $len(e_1) = 0$ or $len(e_1) > 0$ and $len(e_2) > (nl_1 - ind_0(e_1))$, this follows from the fact that $ind_l(e_2) \leq nl_2$. In the only remaining case, $ind_0(e_1) \leq (nl_1 - len(e_2))$. Noting that in this case $ind(\{\!\{e_1, nl_1, ol_2, e_2\}\!\}) = ind_0(e_1)$ and that $len(e_2) = ol_2$, the desired conclusion is obtained. In the case of (m5), *i.e.*, when the rule is

$$\{\!\{et :: e_1, j, k, e_2\}\!\} \rightarrow \langle\!\langle et, j, k, e_2\rangle\!\rangle :: \{\!\{e_1, j, k, e_2\}\!\}.$$

we need to verify that $ind(\langle\!\langle et, j, k, e_2\rangle\!\rangle) \geq ind(\{\!\{e_1, j, k, e_2\}\!\})$. However, this is done easily using Lemmas 4.10 and 4.7.

□

We illustrate the rewrite rules presented in this section by considering their use on the term $((\lambda\,((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2))\ t_3)$, assuming that t_2 and t_3 are arbitrary de Bruijn terms. The following constitutes a $\rhd_{rm\beta_s}$-reduction sequence for this term:

$$\begin{array}{l}
((\lambda\,((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2))\ t_3) \\
\quad \rhd_{\beta_s} [\![((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2), 1, 0, (t_3, 0) :: nil]\!] \\
\quad \rhd_{\beta_s} [\![[\![(\lambda\,((\#1\ \#2)\ \#3)), 1, 0, (t_2, 0) :: nil]\!], 1, 0, (t_3, 0) :: nil]\!] \\
\quad \rhd_{m} [\![(\lambda\,((\#1\ \#2)\ \#3)), 2, 0, \{\!\{(t_2, 0) :: nil, 0, 1, (t_3, 0) :: nil\}\!\}]\!] \\
\quad \rhd_{m} [\![(\lambda\,((\#1\ \#2)\ \#3)), 2, 0, \langle\!\langle (t_2, 0), 0, 1, (t_3, 0) :: nil\rangle\!\rangle :: \{\!\{nil, 0, 1, (t_3, 0) :: nil\}\!\}]\!] \\
\quad \rhd_{m} [\![(\lambda\,((\#1\ \#2)\ \#3)), 2, 0, ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: \{\!\{nil, 0, 1, (t_3, 0) :: nil\}\!\}]\!] \\
\quad \rhd_{m} [\![(\lambda\,((\#1\ \#2)\ \#3)), 2, 0, ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: (t_3, 0) :: nil]\!].
\end{array}$$

Notice that, in producing this term, the merging of suspensions has been realized through a sequence of genuinely atomic steps. The combined environment can now be moved inside the remaining abstraction by using a reading rule to yield the term

$$(\lambda\,[\![((\#1\ \#2)\ \#3), 3, 1, @0 :: ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: (t_3, 0) :: nil]\!]).$$

A repeated application of reading rules transforms the last term into

$$(\lambda\,((\#1\ [\![[\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0, 1, nil]\!])\ [\![t_3, 0, 1, nil]\!])).$$

The application of merging rules to this term yields

$$(\lambda\,((\#1\ [\![t_2, 1, 1, (t_3, 0) :: nil]\!])\ [\![t_3, 0, 1, nil]\!])).$$

Depending on the particular structures of t_2 and t_3, the reading rules can be applied repeatedly to this term to finally produce a de Bruijn term that results from the original term by contracting the two outermost β-redexes.

4.4 Some properties of our notation

We observe some properties of $\rhd_{rm}$ that relate our notation to the earlier informal discussion of it.

Lemma 4.13 *Let e be a simple environment. Then*

$$[\![\#i, ol, nl, e]\!] \quad \rhd^*_{rm} \begin{cases} \#(i + (nl - ol)) & \text{if } i > ol \\ \#(nl - m) & \text{if } i \leq ol \text{ and } e[i] = @m \\ [\![t, 0, nl - m, nil]\!] & \text{if } i \leq ol \text{ and } e[i] = (t, m). \end{cases}$$

Proof. By an induction on ol if $i > ol$ and on i if $i \leq ol$, using the rule schemata (r2)–(r5).

□

Lemma 4.14 *Let e be a simple environment. If $(nl - l) \geq ol$, then $\langle\!\langle (t, l), nl, ol, e \rangle\!\rangle \rhd^*_{rm} (t, l)$. If $(nl - l) < ol$, then $\langle\!\langle (t, l), nl, ol, e \rangle\!\rangle$ $\rhd_{rm}$-reduces to*

$$([\![t, ol - (nl - l), ind(e[nl - l + 1]), e\{nl - l + 1\}]\!], ind(e[nl - l + 1]) + (nl \mathbin{\dot{-}} ol)).$$

Proof. If $(nl - l) < ol$, we use an induction on $(nl - l)$ and if $(nl - l) \geq ol$, we use an induction on ol. Rule schemata (m10), (m9) and (m6) are used in this proof.

□

Lemma 4.15 *Let e be a simple environment. Then*

$$\langle\!\langle @l, nl, ol, e \rangle\!\rangle \quad \rhd^*_{rm} \begin{cases} @l & \text{if } (nl - l) > ol \\ @(m + (nl \mathbin{\dot{-}} ol)) & \text{if } (nl - l) \leq ol \text{ and } e[nl - l] = @m \\ (t, m + (nl \mathbin{\dot{-}} ol)) & \text{if } (nl - l) \leq ol \text{ and } e[nl - l] = (t, m). \end{cases}$$

Proof. Analogous to that of Lemma 4.14, using rule schemata (m6)–(m8) and (m10).

□

Lemma 4.16 *Let e_2 be a simple environment. Then $\{\!\{nil, nl, ol, e_2\}\!\}$ $\rhd_{rm}$-reduces to nil if $nl \geq ol$ and to $e_2\{nl + 1\}$ otherwise.*

Proof. By an induction on ol if $nl \geq ol$ and on nl if $nl < ol$ using rule schemata (m2)–(m4).

□

Suppose that e_1 and e_2 are simple environments and, further, that e_1 is $et_1 :: \ldots et_n :: nil$. By a repeated use of rule schema (m5), the term $\{\!\{e_1, nl, ol, e_2\}\!\}$ can be reduced to

$$\langle\!\langle et_1, nl, ol, e_2 \rangle\!\rangle :: \ldots :: \langle\!\langle et_n, nl, ol, e_2 \rangle\!\rangle :: \{\!\{nil, nl, ol, e_2\}\!\}.$$

Lemmas 4.14–4.16 show the correspondence of this environment to the desired merged environment described in Section 4.1.

The above observations are relativized to simple expressions. They extend in a natural way to arbitrary expressions once the existence of $\rhd_{rm}$-normal forms has been demonstrated.

5 A well founded partial order on suspension expressions

We define in this section a well founded partial ordering relation on suspension expressions that will be used primarily in showing the finiteness of all $\rhd_{rm}$-reduction sequences. Not surprisingly, a determining factor in this relation is a measure of the work remaining in calculating a suspended substitution. To understand the construction of a possible measure, consider a term of the form $[\![t, ol, nl, e]\!]$. The substitutions encoded in this term need to be propagated over the structure of t and so it is relevant to count the complexity of this structure. Further, terms from e are embedded in a suspension before they are substituted in — this is apparent from rule schema (r4) — and the complexity of their structure should also be counted. A complication in this basic pattern is that the propagation of substitutions may create multiple copies of an environment — this happens, for instance, when rule schema (r6) is used to rewrite $[\![(t_1\ t_2), ol, nl, e]\!]$ — and yet the resulting expression should have a lower complexity. A solution to this problem is to use the maximum 'height' in a term over which substitutions have to be propagated as opposed to the complexity of the structure of the term.

The ideas described above underlie the measure η that we now define. The auxiliary measure μ used in defining η counts, roughly, the heights of terms. The function max on pairs of integers picks the larger of its arguments.

Definition 5.1 The measures η on expressions and μ on terms are given as follows:

Category of exp	*exp*	$\eta(exp)$	$\mu(exp)$
term	constant	0	1
	$\#i$	0	1
	$(t_1\ t_2)$	$\max(\eta(t_1), \eta(t_2))$	$\max(\mu(t_1), \mu(t_2)) + 1$
	$(\lambda\, t)$	$\eta(t)$	$\mu(t) + 1$
	$[\![t, ol, nl, e]\!]$	$\mu(t) + \eta(e)$	$\mu(t) + \eta(e) + 1$
environment	nil	0	—
	$et :: e$	$\max(\eta(et), \eta(e))$	—
	$\{\!\{e_1, nl, ol, e_2\}\!\}$	$\eta(e_1) + \eta(e_2) + 1$	—
environment term	$@l$	0	—
	(t, l)	$\mu(t)$	—
	$\langle\!\langle et, nl, ol, e\rangle\!\rangle$	$\eta(et) + \eta(e) + 1$	—

The following properties of the measures η and μ are easily observed.

Lemma 5.2 *For any expression x, $\eta(x) \geq 0$. Further, for any term t, $\mu(t) > \eta(t)$.*

Lemma 5.3 *Let x_1 and x_2 be expressions of the same syntactic category and such that $\eta(x_1) \geq \eta(x_2)$ and, if x_1 and x_2 are terms, $\mu(x_1) \geq \mu(x_2)$. If x results from y by the replacement of subexpression x_1 by x_2, then $\eta(y) \geq \eta(x)$ and, if x and y are terms, $\mu(y) \geq \mu(x)$.*

The measure η does not yield by itself the ordering relation we desire. The reason for this is twofold. First, there are certain rewrite rules — in particular, those obtained from the schemata (r5), (m1), (m3), (m5) and (m10) — for which the lefthand and righthand sides have the same η value. Second, replacing a subexpression by one with a lower η value does not necessarily decrease the η value of the overall expression. We deal with these problems by extending the ordering on expressions imposed by η in a way that specifically overcomes them. This is the content of Definition 5.5.

Definition 5.4 Two expressions are said to have the same top-level structure if they are both constants, variable references, abstractions, applications, or suspensions or if they are both of the forms nil, $et :: e$, $\{\!\{e_1, i, j, e_2\}\!\}$, $@l$, (t, l), or $\langle\!\langle et, i, j, e\rangle\!\rangle$. If two suspension expressions that have the same top-level structure have any immediate subexpressions, then there is an obvious correspondence between these subexpressions. This correspondence will be utilized below.

Definition 5.5 Given two expressions x_1 and x_2, we say $x_1 \sqsupset x_2$ if either $\eta(x_1) > \eta(x_2)$ or $\eta(x_1) = \eta(x_2)$ and one of the following conditions hold:

(1) x_1 is $\#i$ and x_2 is $\#j$ where $i > j$.

(2) x_1 is $[\![t_1, ol_1, nl_1, e_1]\!]$, x_2 is $[\![t_2, ol_2, nl_2, e_2]\!]$ and $\eta(t_1) > \eta(t_2)$.

(3) x_1 is $\{\!\{e_1, nl, ol, e_2\}\!\}$, x_2 is $et :: e$ and $x_1 \sqsupset e$.

(4) x_1 and x_2 have the same top-level structure and also have immediate subexpressions such that each immediate subexpression of x_1 is identical to the corresponding immediate subexpression of x_2 except for one pair of immediate subexpressions x_1' of x_1 and x_2' of x_2 for which $x_1' \sqsupset x_2'$.

(5) x_2 is an immediate subexpression of x_1.

We shall write $x_1 \sqsupseteq x_2$ to signify that $x_1 = x_2$ or $x_1 \sqsupset x_2$.

Note that $\sqsupset$ is not transitive and hence it is not a partial ordering relation.[5] However, its transitive closure provides a well founded partial ordering relation. The following lemma will be useful in showing that this is the case.

Lemma 5.6 *There is no infinite sequences of expressions $x_1, x_2, \ldots, x_n, \ldots$ such that*

$$x_1 \sqsupset x_2 \sqsupset \cdots \sqsupset x_n \sqsupset \cdots.$$

Proof. Let us use the phrase "infinite descending sequence" to denote an infinite sequence of expressions $x_1, x_2, \ldots, x_n, \ldots$ such that $x_1 \sqsupset x_2 \sqsupset \cdots \sqsupset x_n \sqsupset \cdots$. We prove the following

[5] A partial ordering relation has been described in the literature to be one that is irreflexive and transitive [27] as well as to be one that is reflexive, transitive and antisymmetric [16]. It is the former definition that we use here.

by induction on $\eta(x_1)$: (a) there is no infinite descending sequence of expressions $x_1, x_2, \ldots, x_n, \ldots$ such that, for $i, j \geq 1$, $\eta(x_i) = \eta(x_j)$, and (b) there is no infinite descending sequence of expressions. Note that if $x \sqsupset y$, then $\eta(x) \geq \eta(y)$. Thus, (b) is an consequence of (a) and the hypothesis. It is therefore only necessary to show (a). We do this by an induction on the structure of x. The argument proceeds by considering the various possibilities for this structure.

If x is a term: x is minimal with respect to $\sqsupset$ if it is a constant. If x is $\#k$, the descending chain is of length at most $(k-1)$. Suppose x is the application $(s_1\ t_1)$. Any infinite descending sequence of expressions starting at x and preserving η values must be of one of two forms:

$$(s_1\ t_1), (s_2\ t_2), \ldots, (s_n\ t_n), \ldots,$$

where, for $i \geq 1$, either $s_i = s_{i+1}$ and $t_i \sqsupset t_{i+1}$ or $s_i \sqsupset s_{i+1}$ and $t_i = t_{i+1}$, or

$$(s_1\ t_1), (s_2\ t_2), \ldots, (s_n\ t_n), r_{n+1}, \ldots,$$

where, for $1 \leq i < n$, either $s_i = s_{i+1}$ and $t_i \sqsupset t_{i+1}$ or $s_i \sqsupset s_{i+1}$ and $t_i = t_{i+1}$ and r_{n+1} is either s_n or t_n. In either case, there will be an infinite descending sequence starting at either s_1 or t_1. This contradicts the hypothesis since $\eta(s_1), \eta(t_1) \leq \eta(x)$ and s_1 and t_1 are subexpressions of x.

An argument similar to that for an application can be provided when x is an abstraction. This leave only the case of a suspension. Let $x = [\![s_1, ol_1, nl_1, e_1]\!]$. We use now an additional induction on $\eta(s_1)$. By Lemma 5.2, $\eta(x) > \eta(s_1)$ and $\eta(x) > \eta(e_1)$. There are, therefore, no infinite descending sequences from s_1 or e_1. From this, by an argument similar to that used in the case of an application, we see that a purportedly infinite descending sequence starting from t must have an initial segment of the form

$$[\![s_1, ol_1, nl_1, e_1]\!], [\![s_2, ol_2, nl_2, e_2]\!], \ldots, [\![s_n, ol_n, nl_n, e_n]\!], [\![s_{n+1}, ol_{n+1}, nl_{n+1}, e_{n+1}]\!],$$

where, for $1 \leq i < n$, $s_i \sqsupseteq s_{i+1}$ and $e_i \sqsupseteq e_{i+1}$, and $\eta(s_n) > \eta(s_{n+1})$. Clearly, $\eta(s_1) > \eta(s_{n+1})$. Thus, such an initial segment cannot exist if $\eta(s_1) = 0$. Furthermore, even if $\eta(s_1) > 0$, the segment cannot be extended into an infinite descending sequence: that would entail the existence of an infinite descending sequence from $[\![s_{n+1}, ol_{n+1}, nl_{n+1}, e_{n+1}]\!]$, in contradiction to the hypothesis. The claim must, therefore, be true in this case as well.

If x is an environment term: x is minimal with respect to $\sqsupset$ if it is of the form $@l$. If x is (t', l), then there is an infinite descending sequence from it only if there is one from t'. However, $\eta(x) > \eta(t')$ by Lemma 5.2 and so this is impossible by hypothesis. Finally, suppose that x is $\langle\!\langle et, nl, ol, e\rangle\!\rangle$. There can be an infinite descending sequence from x only if there is one from either et or e. This is, again, impossible because $\eta(x) > \eta(et)$ and $\eta(x) > \eta(e)$.

If x is an environment: x is minimal with respect to $\sqsupset$ if it is nil. Let $x = et :: e$. By an argument similar to that for an application, there is an infinite descending sequence starting at x only if there is also one starting at et or e. However, this is impossible by hypothesis, because $\eta(x) \geq \eta(et)$, $\eta(x) \geq \eta(e)$, and et and e are subexpressions of x.

The remaining case, where x is of the form $\{\!\{e_1, nl, ol, e_1'\}\!\}$, requires a non-constructive proof. Let us assume that there are infinite descending sequences starting at x. We pick from these a sequence $x = y_1, y_2, y_3, \ldots$ that is minimal in the following sense: for each $i \geq 1$, there is no infinite descending sequence of the form $y_1, y_2, \ldots, y_i, y_{i+1}', \ldots$ where y_{i+1}' is a subexpression of y_{i+1}. We focus now on the sequence picked. Since $\eta(x) > \eta(e_1)$ and $\eta(x) > \eta(e_1')$, there are, by hypothesis, no infinite descending sequences starting at either e_1 or e_1'. From this, it is easily seen that our sequence must be of the form

$$\{\!\{e_1, nl, ol, e_1'\}\!\}, \{\!\{e_2, nl, ol, e_2'\}\!\}, \ldots, \{\!\{e_n, nl, ol, e_n'\}\!\}, et_{n+1} :: e_{n+1}, \ldots,$$

where, for $1 \leq i < n$, $e_i \sqsupseteq e_{i+1}$, $e_i' \sqsupseteq e_{i+1}'$ and $\{\!\{e_n, nl, ol, e_n'\}\!\} \sqsupset e_{n+1}$. Now, this infinite descending sequence entails that there is a similar sequence starting from $et_{n+1} :: e_{n+1}$. By a familiar argument, this can be the case only if there is an infinite descending sequence $z_1, z_2, z_3, \ldots$, where z_1 is either et_{n+1} or e_{n+1}. We note that $\eta(et_{n+1}) \leq \eta(x)$ and that et_{n+1} is an environment term. Thus, we have already shown that the former situation is impossible. In the latter case, we can construct the infinite descending sequence

$$\{\!\{e_1, nl, ol, e_1'\}\!\}, \{\!\{e_2, nl, ol, e_2'\}\!\}, \ldots, \{\!\{e_n, nl, ol, e_n'\}\!\}, z_1, z_2, z_3, \ldots,$$

contradicting our assumption of minimality for the sequence picked initially. We conclude, therefore, that no infinite sequence could have existed to begin with.

All the cases having been considered, our claim stands verified and so the lemma must be true.

□

We now deliver the promised ordering relation on expressions.

Definition 5.7 The relation $\succ$ on expressions is the transitive closure of the relation $\sqsupset$.

Theorem 5.8 *The relation $\succ$ is a well founded partial ordering relation on expressions.*

Proof. We need to show that $\succ$ is irreflexive and, assuming that it is a partial order, is also well founded. Both requirements follow from the observation that there can be no infinite descending chains relative to $\succ$, a fact that is an obvious consequence of Lemma 5.6.

□

We have provided a direct proof for the fact that $\succ$ is well founded so as to give specific insight into the nature of this relation. However, an alternative proof can be provided by invoking Kruskal's tree theorem [14, 26], thereby exhibiting relationships between $\succ$ and the notions of simplification orderings [12] and Kamin and Lévy's extended recursive path orderings (described, for example, in [22]). Towards this end, we note that expressions of the form $[\![t, ol, nl, e]\!]$, $\{\!\{e_1, nl, ol, e_2\}\!\}$ and

$\langle\langle et, nl, ol, e\rangle\rangle$ can be thought of as functions of two arguments by incorporating nl and ol into the name of the function symbol; thus, the first expression may be rendered into the expression $f_{ol,nl}(t, e)$, the second into $g_{nl,ol}(e_1, e_2)$ and the third into $h_{nl,ol}(et, e)$. In a similar fashion, an environment term of the form (t, l) could be rendered into the expression $k_l(t)$, *i.e.*, a function of one argument. Finally, expressions of the form $(t_1\ t_2)$ and $(\lambda\, t)$ can be translated into $app(t_1, t_2)$ and $lam(t)$, respectively, :: can be interpreted as a binary function symbol and expressions of the form $\#k$, $@l$ and nil can be thought of as constants. Given such a translation, $\succ$ can be seen to be a simplification ordering. This alone does not allow us to conclude that $\succ$ is well founded, since the alphabet over which our terms are constructed is infinite. However, let $\approx$ be the relation over this alphabet that includes the identity relation and is such that (i) $f_{ol,nl} \approx f_{ol',nl'}$, $g_{nl,ol} \approx g_{nl',ol'}$ and $h_{nl,ol} \approx h_{nl',ol'}$ for all ol, ol', nl, nl', (ii) $k_l \approx k_{l'}$ and $@l \approx @l'$ for all l, l', (iii) $\#i \approx \#j$ if $i \leq j$, and (iv) $c \approx c'$ for all constants c and c' of the original vocabulary. It is easily seen that $\approx$ is a well quasi ordering relation on the alphabet. Now, let $\preceq$ be the homeomorphic embedding of $\approx$. By Kruskal's theorem, $\preceq$ is a well quasi order on expressions. We observe at this point that if $x \succ y$, then it cannot be the case that $x \preceq y$. From this it follows that $\succ$ is well founded.

6 Correctness of the reading and merging rules

The reading and merging rules propagate substitutions embodied in suspension expressions. The correctness of these rules is dependent on their ability to eventually transform any given expression in our notation into ones that are 'substitution-free'. Further, the expression that is so produced should be independent of the order of application of the rules. In the terminology of rewrite systems, these two requirements amount to the existence of a unique $\rhd_{rm}$-normal form for every expression. A final requirement is that the effect of using these rules should correspond to our informal understanding of the meaning of a suspension term. We show in this section that all these properties hold of the reading and merging rules.

6.1 Existence of Normal Forms

A stronger property than the existence of a normal form for every expression holds of the $\rhd_{rm}$ relation: *every* $\rhd_{rm}$-reduction sequence terminates. The proof of this property uses the well founded partial ordering relation defined in Section 5 in an obvious fashion.

Lemma 6.1 *If $l \rightarrow r$ is an instance of one of the rule schemata in Figures 2 or 3, then $\eta(l) \geq \eta(r)$ and, if l and r are terms, $\mu(l) \geq \mu(r)$.*

Proof. By a routine inspection of the rules in question. We omit the details, but note that in all cases except when the rule is an instance of (r5), (m1), (m3), (m5) or (m10), $\eta(l) > \eta(r)$.

□

Lemma 6.2 *If x_1 and x_2 are expressions such that $x_1 \rhd_{rm} x_2$, then $\eta(x_1) \geq \eta(x_2)$.*

Proof. This follows immediately from Lemmas 5.3 and 6.1.

$\square$

Lemma 6.3 *If $l \to r$ is an instance of one of the rule schemata in Figures 2 or 3, then $l \succ r$.*

Proof. Immediate from the fact noted in the proof of Lemma 6.1 in all cases except when the rule is an instance of (r5), (m1), (m3), (m5) or (m10). In the cases left, the lemma is easily shown to be true using Lemma 6.1 and inspecting Definitions 5.5 and 5.7. It is necessary only to note, for (m1), that $\eta([\![t, ol, nl, e]\!]) \geq \mu(t)$ and, by Lemma 5.2, $\mu(t) > \eta(t)$.

$\square$

Lemma 6.4 *If $x_1 \rhd_{rm} x_2$, then $x_1 \succ x_2$.*

Proof. By induction on the structure of x_1. If $x_1 \to x_2$ is an instance of one of the rule schemata in Figures 2 or 3, this follows from Lemma 6.3. Otherwise x_1 and x_2 have the same top-level structure and, by Lemma 6.2, $\eta(x_1) \geq \eta(x_2)$. If $\eta(x_1) > \eta(x_2)$, the desired conclusion follows. If $\eta(x_1) = \eta(x_2)$, by the definition of $\rhd_{rm}$ and by the hypothesis, there is an immediate subexpression x_1' of x_1 and a corresponding immediate subexpression x_2' of x_2 such that $x_1' \succ x_2'$ and every other immediate subexpressions of x_1 is identical to the corresponding immediate subexpression of x_2. Using the definition of $\succ$, it follows easily that $x_1 \succ x_2$.

$\square$

Theorem 6.5 *The relation $\rhd_{rm}$ is noetherian.*

Proof. An obvious consequence of Lemma 6.4 and Theorem 5.8.

$\square$

Thus, a $\rhd_{rm}$-normal form exists for every expression. We note that our rules eventually transform suspension terms into de Bruijn terms and that they produce simple expressions in general.

Theorem 6.6 *An expression x is in $\rhd_{rm}$-normal form if and only if one of the following holds: (a) x is a de Bruijn term; (b) x is an environment term of the form $@l$ or (t, l) where t is a term in $\rhd_{rm}$-normal form; or (c) x is an environment of the form nil or $et :: e$ where et and e are, respectively, an environment term and an environment in $\rhd_{rm}$-normal form.*

Proof. An inspection of Figures 2 and 3 shows that a well formed expression that has a subexpression of the form $[\![t, i, j, e]\!]$, $\{\!\{e_1, i, j, e_2\}\!\}$ or $\langle\!\langle et, i, j, e\rangle\!\rangle$ can be rewritten by using one of the rule schemata appearing in these figures. Such an expression can therefore not be in $\rhd_{rm}$-normal form.

$\square$

6.2 An associativity property for environment merging

More than two environments might be merged in the production of a $\rhd_{rm}$-normal form. For example, given the term $[\![[\![[\![t, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!], ol_3, nl_3, e_3]\!]$, the three environments e_1, e_2 and e_3 might be merged before the substitutions they represent are propagated over the structure of t. Now, such a merging can be accomplished in two different ways: we may merge e_1 and e_2 first and then merge the result with e_3, or we may merge e_1 with the outcome of merging e_2 and e_3. The environments that are produced by these different processes are given by

$$\{\!\{\{\!\{e_1, nl_1, ol_2, e_2\}\!\}, nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\}\!\} \text{ and } \{\!\{e_1, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \{\!\{e_2, nl_2, ol_3, e_3\}\!\}\}\!\},$$

respectively. A question that is pertinent to the uniqueness of $\rhd_{rm}$-normal forms is whether the *same* environment results from either merging process.

We answer this question affirmatively below. In particular, we show that the reading and merging rules can be used to rewrite the two displayed environment expressions to a common form. At a conceptual level, our argument utilizes a partitioning into two kinds of the elements of the environments corresponding to these expressions: those obtained from transforming the elements of e_1 to account for the substitutions encoded in e_2 and e_3 and those obtained from merging (relevant segments of) e_2 and e_3. For each kind of element, we show that the "calculations" encoded in the two different expressions can be made to converge. A detailed consideration of cases is involved of necessity in this process. The trusting reader may wish only to note the statement of Theorem 6.12.

The following observation is needed in arguing the identity of indices of environment terms.

Lemma 6.7 $((i + (j \dot{-} k)) \dot{-} l) = (i \dot{-} l) + (j \dot{-} (k + (l \dot{-} i)))$.

Proof. $((i + (j \dot{-} k)) \dot{-} l) = (i \dot{-} l) + ((j \dot{-} k) \dot{-} (l \dot{-} i)) = (i \dot{-} l) + (j \dot{-} (k + (l \dot{-} i)))$.

□

Certain reduction properties for environments and environment term will also be useful.

Lemma 6.8 *Let et be an environment term such that $ind(et) \leq nl$. Then, for $j \geq 1$,*

$$\langle\!\langle et, nl + j, ol + j, et_1 :: \ldots :: et_j :: e\rangle\!\rangle \rhd_{rm}^{*} \langle\!\langle et, nl, ol, e\rangle\!\rangle.$$

Proof. By an induction on j, using rule schema (m10).

□

Lemma 6.9 *Let e_1 be a simple environment. Further, let nl and ol be natural numbers such that $(nl - ind(e_1)) \geq ol$. Then $\{\!\{e_1, nl, ol, e_2\}\!\} \rhd_{rm}^{*} e_1$.*

Proof. We assume that e_2 is also a simple environment; if not, it can be $\rhd_{rm}$-reduced to one. We now use an induction on $len(e_1)$. If this is 0, then $e_1 = nil$. Since $ind(nil) = 0$, $nl \geq ol$ and so, by Lemma 4.16, $\{\!\{e_1, nl, ol, e_2\}\!\}\rhd^*_{rm} nil$. If $len(e_1) > 0$, e_1 is of the form $et_1 :: e'_1$. Using rule schema (m5), $\{\!\{e_1, nl, ol, e_2\}\!\}\rhd^*_{rm}\langle\!\langle et_1, nl, ol, e_2\rangle\!\rangle :: \{\!\{e'_1, nl, ol, e_2\}\!\}$. We note that $ind(et_1) = ind(e_1)$ and, by the definition of wellformedness, $ind(e'_1) \leq ind(e_1)$. The lemma then follows from Lemma 6.8, rule schema (m6) and the inductive hypothesis.

$\square$

Lemma 6.10 *If e_1 is an environment such that $ind(e_1) \leq nl$ then, for $j \geq 1$, the expressions*

$$\{\!\{e_1, nl + j, ol + j, et_1 :: \ldots :: et_j :: e_2\}\!\} \quad \text{and} \quad \{\!\{e_1, nl, ol, e_2\}\!\}$$

$\rhd_{rm}$-reduce to a common expression for any environment e_2.

Proof. We assume that e_1 and e_2 are simple expressions: if they are not, then they can be $\rhd_{rm}$-reduced to such expressions and, since, by Lemma 4.11, $ind(e_1)$ is preserved by such a reduction, we can then invoke the argument provided here. We now use an induction on $len(e_1)$. If this is 0, using Lemma 4.16 we see that both expressions $\rhd_{rm}$-reduce to nil if $nl \geq ol$ and to $e_2\{nl + 1\}$ otherwise. If $len(e_1) > 0$, then e_1 is of the form $et' :: e'_1$. Using rule schema (m5),

$$\begin{aligned}&\{\!\{e_1, nl + j, ol + j, et_1 :: \ldots :: et_j :: e_2\}\!\}\rhd_{rm}\\ &\qquad\langle\!\langle et', nl + j, ol + j, et_1 :: \ldots :: et_j :: e_2\rangle\!\rangle :: \{\!\{e'_1, nl + j, ol + j, et_1 :: \ldots :: et_j :: e_2\}\!\},\end{aligned}$$

and, similarly, $\{\!\{e_1, nl, ol, e_2\}\!\}\rhd_{rm}\langle\!\langle et', nl, ol, e_2\rangle\!\rangle :: \{\!\{e'_1, nl, ol, e_2\}\!\}$. Noting that $ind(et') = ind(e_1)$ and $ind(e'_1) \leq ind(e_1)$, Lemma 6.8 and the inductive hypothesis yield the desired conclusion.

$\square$

Suppose that e_1 is an environment of the form $et_1 :: e'_1$. The first element of the merger of e_1, e_2 and e_3 can then be calculated in two ways: by accounting for the effect of e_2 on et_1 and, subsequently, for the effect of e_3 on the result or by accounting for the effect on et_1 of the merger of e_2 and e_3. We show below that an identical value can be produced using either method of calculation.

Lemma 6.11 *Let a and b be environment terms of the form*

$$\langle\!\langle\langle\!\langle et_1, nl_1, ol_2, e_2\rangle\!\rangle, nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\rangle\!\rangle \text{ and } \langle\!\langle et_1, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \{\!\{e_2, nl_2, ol_3, e_3\}\!\}\rangle\!\rangle,$$

respectively. Then there is an environment term r such that $a\rhd^*_{rm} r$ and $b\rhd^*_{rm} r$.

Proof. We assume, without loss of generality, that et_1, e_2 and e_3 are simple expressions and we prove the lemma by an induction on $len(e_2)$.

Base Case: $len(e_2) = 0$. In this case, a is $\langle\!\langle\langle\!\langle et_1, nl_1, 0, nil\rangle\!\rangle, nl_2 + nl_1, ol_3, e_3\rangle\!\rangle$ and, similarly, b is $\langle\!\langle et_1, nl_1, ol_3 \dot{-} nl_2, \{\!\{nil, nl_2, ol_3, e_3\}\!\}\rangle\!\rangle$. Using rule schema (m6), $a\rhd^*_{rm}\langle\!\langle et_1, nl_2 + nl_1, ol_3, e_3\rangle\!\rangle$.

Our analysis now splits into two subcases, depending on whether or not $nl_2 \geq ol_3$. Suppose that $nl_2 \geq ol_3$. Noting that $ind(et_1) \leq nl_1$, by either Lemma 4.14 or Lemma 4.15, $a \rhd^*_{rm} et_1$.Using Lemma 4.16 and rule schema (m6) it is easily seen that b also $\rhd_{rm}$-reduces to et_1. Suppose instead that $nl_2 < ol_3$. Using Lemmas 6.8 and 4.16 it can be seen that both a and b $\rhd_{rm}$-reduce to $\langle\langle et_1, nl_1, ol_3 \dot{-} nl_2, e_3\{nl_2 + 1\}\rangle\rangle$.

Inductive Step: $len(e_2) > 0$. Let e_2 be of the form $et_2 :: e'_2$. We now use a further induction on $nl_1 - ind(et_1)$.

Base Case for Second Induction: $nl_1 - ind(et_1) = 0$. We consider the cases for the structure of et_1:

(a) et_1 is of the form $@l$. We note first that $l = nl_1 - 1$. Now, our analysis splits into two further subcases, depending on whether et_2 is of the form $@m$ or of the form (t, m).

Suppose et_2 is of the form $@m$. Using Lemma 4.15 on the one hand and rule schema (m5) on the other, it can be seen that

$$a \rhd^*_{rm} \langle\langle @(m + (nl_1 \dot{-} ol_2)), nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\rangle\rangle \quad \text{and}$$
$$b \rhd^*_{rm} \langle\langle @l, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \langle\langle @m, nl_2, ol_3, e_3\rangle\rangle :: \{\!\{e'_2, nl_2, ol_3, e_3\}\!\}\rangle\rangle.$$

Now, if $(nl_2 - m) > ol_3$, by using Lemma 4.15 repeatedly and noting that $(ol_3 \dot{-} nl_2) = 0$, it can be seen that a and b both $\rhd_{rm}$-reduce to $@(m + (nl_1 \dot{-} ol_2))$. If, on the other hand, $(nl_2 - m) \leq ol_3$, we need to consider the form of $e_3[nl_2 - m]$. In the case that this is (t, p), then, using Lemmas 4.15 and 6.7, it can be seen that both a and b $\rhd_{rm}$-reduce to $(t, p + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3))$. If $e_3[nl_2 - m]$ is $@p$, both a and b can be shown to $\rhd_{rm}$-reduce to $@(p + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3))$ by a similar argument.

Suppose instead that et_2 is of the form (t, m). Using Lemma 4.15 and rule schema (m5) again, we see that

$$a \rhd^*_{rm} \langle\langle (t, m + (nl_1 \dot{-} ol_2)), nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\rangle\rangle \quad \text{and}$$
$$b \rhd^*_{rm} \langle\langle @l, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \langle\langle (t, m), nl_2, ol_3, e_3\rangle\rangle :: \{\!\{e'_2, nl_2, ol_3, e_3\}\!\}\rangle\rangle.$$

Now, if $(nl_2 - m) \geq ol_3$, it can be seen that both a and b $\rhd_{rm}$-reduce to $(t, m + (nl_1 \dot{-} ol_2))$. On the other hand, if $(nl_2 - m) < ol_3$, it follows from Lemmas 4.14, 4.15 and 6.7 that a and b both $\rhd_{rm}$-reduce to

$$([\![t, ol_3 - (nl_2 - m), ind(e_3[nl_2 - m + 1]), e_3\{nl_2 - m + 1\}]\!],$$
$$ind(e_3[nl_2 - m + 1]) + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3)).$$

(b) et_1 is of the form (t, l). Clearly, $l = nl_1$. Let $ind(et_2) = m$. Using Lemma 4.14,

$$a \rhd^*_{rm} \langle\langle ([\![t, ol_2, m, e_2]\!], m + (nl_1 \dot{-} ol_2)), nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\rangle\rangle \qquad (1)$$

Our analysis again splits into two subcases, depending on whether or not $(nl_2 - m) \geq ol_3$.

Suppose that $(nl_2 - m) \geq ol_3$. Using Lemma 4.14 and the observation that

$$(nl_2 + (nl_1 \dot{-} ol_2) - (m + (nl_1 \dot{-} ol_2))) = (nl_2 - m) \geq ol_3,$$

it follows from (1) that $a \rhd^*_{rm} (\llbracket t, ol_2, m, e_2 \rrbracket, m + (nl_1 \dot{-} ol_2))$. Since $(nl_2 - m) \geq ol_3$, $(ol_3 \dot{-} nl_2) = 0$. Hence, using Lemma 6.9 and the fact that $ind(e_2) = ind(et_2)$, $\{\!\{e_2, nl_2, ol_3, e_3\}\!\} \rhd^*_{rm} e_2$. Invoking Lemma 4.14 we can now conclude that

$$b = \langle\!\langle (t, nl_1), nl_1, ol_2, \{\!\{e_2, nl_2, ol_3, e_3\}\!\} \rangle\!\rangle \rhd^*_{rm} (\llbracket t, ol_2, m, e_2 \rrbracket, m + (nl_1 \dot{-} ol_2)).$$

Thus, a and b both $\rhd_{rm}$-reduce to the same expression in this case.

In the remaining subcase, $(nl_2 - m) < ol_3$. Lemma 4.14 used in conjunction with (1) yields

$$\begin{aligned} a \rhd^*_{rm} (\llbracket \llbracket t, ol_2, m, e_2 \rrbracket, ol_3 - (nl_2 - m), ind(e_3[nl_2 - m + 1]), e_3\{nl_2 - m + 1\} \rrbracket, \\ ind(e_3[nl_2 - m + 1]) + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3)). \end{aligned}$$

Using rule schema (m1), we get from this that

$$\begin{aligned} a \rhd^*_{rm} (\llbracket t, ol_2 + ((ol_3 - (nl_2 - m)) \dot{-} m), \\ ind(e_3[nl_2 - m + 1]) + (m \dot{-} (ol_3 - (nl_2 - m))), \\ \{\!\{e_2, m, ol_3 - (nl_2 - m), e_3\{nl_2 - m + 1\}\}\!\} \rrbracket, \\ ind(e_3[nl_2 - m + 1]) + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3)). \end{aligned}$$

Now, since $(nl_2 - m) < ol_3$, it must be the case that

$$((ol_3 - (nl_2 - m)) \dot{-} m) = (ol_3 \dot{-} nl_2) \text{ and } (m \dot{-} (ol_3 - (nl_2 - m))) = (nl_2 \dot{-} ol_3).$$

These identities can be used to simplify the expression a is shown to $\rhd_{rm}$-reduce to. In particular,

$$\begin{aligned} a \rhd^*_{rm} (\llbracket t, ol_2 + (ol_3 \dot{-} nl_2), ind(e_3[nl_2 - m + 1]) + (nl_2 \dot{-} ol_3), \\ \{\!\{e_2, m, ol_3 - (nl_2 - m), e_3\{nl_2 - m + 1\}\}\!\} \rrbracket, \\ ind(e_3[nl_2 - m + 1]) + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3)). \end{aligned} \tag{2}$$

With regard to b, using rule schema (m5) we first observe that it $\rhd_{rm}$-reduces to

$$\langle\!\langle (t, nl_1), nl_1, ol_2 + (ol_3 \dot{-} nl_2), \langle\!\langle et_2, nl_2, ol_3, e_3 \rangle\!\rangle :: \{\!\{e_2', nl_2, ol_3, e_3\}\!\} \rangle\!\rangle.$$

Since $ind(et_2) = m$, it follows from either Lemma 4.14 or Lemma 4.15 that $\langle\!\langle et_2, nl_2, ol_3, e_3 \rangle\!\rangle$ $\rhd_{rm}$-reduces to an environment term whose index is $ind(e_3[nl_2 - m + 1]) + (nl_2 \dot{-} ol_3)$. By Lemma 4.11, indices are preserved under reduction. Hence, using Lemma 4.14,

$$\begin{aligned} b \rhd^*_{rm} (\llbracket t, ol_2 + (ol_3 \dot{-} nl_2), ind(e_3[nl_2 - m + 1]) + (nl_2 \dot{-} ol_3), \\ \langle\!\langle et_2, nl_2, ol_3, e_3 \rangle\!\rangle :: \{\!\{e_2', nl_2, ol_3, e_3\}\!\} \rrbracket, \\ ind(e_3[nl_2 - m + 1]) + (nl_2 \dot{-} ol_3) + (nl_1 \dot{-} (ol_2 + (ol_3 \dot{-} nl_2)))). \end{aligned} \tag{3}$$

Lemma 6.7 can be used to show that the indices of the environment terms in (2) and (3) are identical. Further inspecting these expressions, we see that they would $\triangleright_{rm}$-reduce to a common expression if $\{\!\{e_2, m, ol_3 - (nl_2 - m), e_3\{nl_2 - m + 1\}\}\!\}$ and $\langle\!\langle et_2, nl_2, ol_3, e_3\rangle\!\rangle :: \{\!\{e_2', nl_2, ol_3, e_3\}\!\}$ $\triangleright_{rm}$-reduce to one. Using the rule schema (m5) and invoking Lemmas 6.8 and 6.10 after recalling that $ind(e_2') \leq ind(et_2) = m$ and $(nl_2 - m) < ol_3$, this can be seen to be the case.

Inductive Step for the Second Induction: $nl_1 - ind(et_1) > 0$. In this case, by using rule schema (m10) on a and rule schemata (m5) and (m10) on b, we observe that

$$a \triangleright_{rm}^* \langle\!\langle \langle\!\langle et_1, nl_1 - 1, ol_2 - 1, e_2'\rangle\!\rangle, nl_2 + ((nl_1 - 1) \dot{-} (ol_2 - 1)), ol_3, e_3\rangle\!\rangle, \text{ and}$$
$$b \triangleright_{rm}^* \langle\!\langle et_1, nl_1 - 1, (ol_2 - 1) + (ol_3 \dot{-} nl_2), \{\!\{e_2', nl_2, ol_3, e_3\}\!\}\rangle\!\rangle.$$

Obviously, $len(e_2') < len(e_2)$. The inductive hypothesis can now be invoked to conclude that a and b $\triangleright_{rm}$-reduce to a common expression.

□

Theorem 6.12 *Let a and b be environments of the form*

$$\{\!\{\{\!\{e_1, nl_1, ol_2, e_2\}\!\}, nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\}\!\} \text{ and } \{\!\{e_1, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \{\!\{e_2, nl_2, ol_3, e_3\}\!\}\}\!\},$$

respectively. Then there is an environment r such that $a \triangleright_{rm}^* r$ and $b \triangleright_{rm}^* r$.

Proof. By induction on $len(e_1)$, assuming that e_1, e_2 and e_3 are simple expressions.

Base Case: $len(e_1) = 0$, *i.e.*, $e_1 = nil$. Our analysis splits into two subcases.

(a) $nl_1 < ol_2$. Using Lemma 4.16 and noting that, in this subcase, $(nl_1 \dot{-} ol_2) = 0$, we see that

$$a \triangleright_{rm}^* \{\!\{e_2\{nl_1 + 1\}, nl_2, ol_3, e_3\}\!\}.$$

Using rule schema (m5) repeatedly, it follows that a $\triangleright_{rm}$-reduces to

$$\langle\!\langle e_2[nl_1 + 1], nl_2, ol_3, e_3\rangle\!\rangle :: \ldots :: \langle\!\langle e_2[ol_2], nl_2, ol_3, e_3\rangle\!\rangle :: \{\!\{nil, nl_2, ol_3, e_3\}\!\}. \quad (4)$$

Now, if $nl_1 < ol_2$, then $nl_1 < (ol_2 + (ol_3 \dot{-} nl_2))$. Using this fact together with rule schema (m5) and Lemma 4.16, it can be seen that b also $\triangleright_{rm}$-reduces to the expression shown in (4).

(b) $nl_1 \geq ol_2$. By adopting arguments similar to those in subcase (a), it can be seen that a and b both $\triangleright_{rm}$-reduce to $e_3\{nl_2 + (nl_1 \dot{-} ol_2) + 1\}$ if $ol_3 > (nl_2 + (nl_1 \dot{-} ol_2))$ and to nil if $ol_3 \leq (nl_2 + (nl_1 \dot{-} ol_2))$.

Inductive Step: $len(e_1) > 0$. Let $e_1 = et_1 :: e_1'$. Using rule schema (m5), we see that a and b $\triangleright_{rm}$-reduce to

$$\langle\!\langle \langle\!\langle et_1, nl_1, ol_2, e_2\rangle\!\rangle, nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\rangle\!\rangle :: \{\!\{\{\!\{e_1', nl_1, ol_2, e_2\}\!\}, nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\}\!\}, \text{ and}$$
$$\langle\!\langle et_1, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \{\!\{e_2, nl_2, ol_3, e_3\}\!\}\rangle\!\rangle :: \{\!\{e_1', nl_1, ol_2 + (ol_3 \dot{-} nl_2), \{\!\{e_1, nl_2, ol_3, e_3\}\!\}\}\!\},$$

respectively. Lemma 6.11 and the hypothesis can be used to show that the latter two expressions $\triangleright_{rm}$-reduce to a common expression.

□

6.3 Uniqueness of normal forms

We now show the uniqueness of $\rhd_{rm}$-normal forms. By virtue of Proposition 2.1, this property would hold if $\rhd_{rm}$ is a confluent reduction relation. Further, in light of Proposition 2.2 and Theorem 6.5 it actually suffices to show that $\rhd_{rm}$ is locally confluent.

Theorem 6.13 *The relation $\rhd_{rm}$ is locally confluent.*

Proof. By Theorem 2.4, it is enough to show that, for each conflict pair $\langle r_1, r_2\rangle$ of the rule schemata in Figures 2 and 3, there is some expression s such that $r_1 \rhd_{rm}^* s$ and $r_2 \rhd_{rm}^* s$. To do this, we need to consider the various nontrivial overlaps between the rule schemata in question. Examining these schemata, we see that such overlaps occur only between (m1) and each rule schema in Figure 2, (m1) and (m1) and (m2) and (m4). The last case is dealt with easily: the overlap occurs over the expression $\{\!\{nil, 0, 0, nil\}\!\}$ and the two expressions in the corresponding conflict pair are identical, both being nil. We consider the conflict pairs relative to the remaining overlaps in turn below to complete our argument. In each case, we refer to the expression that constitutes the nontrivial overlap as t and to the terms in the conflict pair as r_1 and r_2 respectively. We will assume that subexpressions that are common to t, r_1 and r_2 are simple ones for, if not, they can always be reduced to such a form at the outset.

Overlap between (m1) and (r1). Let t be the term $[\![[\![c, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!]$. It is easily seen that r_1 and r_2 both $\rhd_{rm}$-reduce to c.

Overlap between (m1) and (r2). Let t be the term $[\![[\![\#i, 0, nl_1, nil]\!], ol_2, nl_2, e_2]\!]$. Then r_1 is the term $[\![\#i, ol_2 \dot{-} nl_1, nl_2 + (nl_1 \dot{-} ol_2), \{\!\{nil, nl_1, ol_2, e_2\}\!\}]\!]$ and r_2 is the term $[\![\#(i + nl_1), ol_2, nl_2, e_2]\!]$. We distinguish three cases:

$nl_1 \geq ol_2$. From Lemmas 4.16 and 4.13 and noting that $(nl_1 \dot{-} ol_2) = (nl_1 - ol_2)$ and $(ol_2 \dot{-} nl_1) = 0$, we conclude that r_1 and r_2 both $\rhd_{rm}$-reduce to $\#(i + nl_2 + (nl_1 - ol_2))$.

$nl_1 < ol_2$ and $i > (ol_2 - nl_1)$. A similar argument to that above can be provided to show that r_1 and r_2 both $\rhd_{rm}$-reduce to $\#(i + nl_2 - (ol_2 - nl_1))$.

$nl_1 < ol_2$ and $i \leq (ol_2 - nl_1)$. The common expression in this case depends on the form of $e_2[i + nl_1]$. If this is $@m$, then r_1 and r_2 are both $\rhd_{rm}$-reduce to $\#(nl_2 - m)$ and if this is (t, m), then r_1 and r_2 both similarly reduce to $[\![t, 0, nl_2 - m, nil]\!]$.

Overlap between (m1) and (r3). Let t be the term $[\![[\![\#1, ol_1, nl_1, @l :: e_1]\!], ol_2, nl_2, e_2]\!]$. Then r_1 and r_2 are the terms

$$[\![\#1, ol_1 + (ol_2 \dot{-} nl_1), nl_2 + (nl_1 \dot{-} ol_2), \{\!\{@l :: e_1, nl_1, ol_2, e_2\}\!\}]\!] \text{ and } [\![\#(nl_1 - l), ol_2, nl_2, e_2]\!],$$

respectively. We distinguish two cases:

$(nl_1 - l) > ol_2$. Note first that $(nl_1 \dot{-} ol_2) = (nl_1 - ol_2)$. Using rule schema (m5), Lemmas 4.13 and 4.15, it then follows that r_1 and r_2 both $\rhd_{rm}$-reduce to $\#(nl_1 - l + nl_2 - ol_2)$.

$(nl_1 - l) \leq ol_2$. A similar argument to that above shows that r_1 and r_2 $\rhd_{rm}$-reduce to $\#(nl_2 - m)$ if $e_2[nl_1 - l] = @m$ and to $[\![t, 0, nl_2 - m, nil]\!]$ if $e_2[nl_1 - l] = (t, m)$.

Overlap between (m1) and (r4). Let t be the term $[\![[\![\#1, ol_1, nl_1, (t, l) :: e_1]\!], ol_2, nl_2, e_2]\!]$. Then r_1 is the term

$$[\![\#1, ol_1 + (ol_2 \dot{-} nl_1), nl_2 + (nl_1 \dot{-} ol_2), \{\!\{(t, l) :: e_1, nl_1, ol_2, e_2\}\!\}]\!]$$

and r_2 is the term $[\![[\![t, 0, nl_1 - l, nil]\!], ol_2, nl_2, e_2]\!]$. Using rule schema (m1), r_2 may be rewritten to

$$[\![t, ol_2 \dot{-} (nl_1 - l), nl_2 + ((nl_1 - l) \dot{-} ol_2), \{\!\{nil, nl_1 - l, ol_2, e_2\}\!\}]\!].$$

From this and from using Lemmas 4.13 and 4.16, it is easily seen that in the case that $(nl_1 - l) \geq ol_2$, r_1 and r_2 both $\rhd_{rm}$-reduce to $[\![t, 0, nl_2 + (nl_1 - l) - ol_2, nil]\!]$. In the case that $(nl_1 - l) < ol_2$, using Lemma 4.16 we see first that r_2 $\rhd_{rm}$-reduces to $[\![t, ol_2 - (nl_1 - l), nl_2, e_2\{nl_1 - l + 1\}]\!]$. We wish to show that r_1 also $\rhd_{rm}$-reduces to this term. Towards this end, letting $ind(e_2\{nl_1 - l + 1\}) = m$ and using rule schema (m5) and Lemma 4.14, we observe that

$$\begin{aligned}&\{\!\{(t, l) :: e_1, nl_1, ol_2, e_2\}\!\} \rhd^*_{rm}\\ &\qquad ([\![t, ol_2 - (nl_1 - l), m, e_2\{nl_1 - l + 1\}]\!], m + (nl_1 \dot{-} ol_2)) :: \{\!\{e_1, nl_1, ol_2, e_2\}\!\}.\end{aligned}$$

But then, by rule schema (r4), $r_1 \rhd^*_{rm} [\![[\![t, ol_2 - (nl_1 - l), m, e_2\{nl_1 - l + 1\}]\!], 0, nl_2 - m, nil]\!]$. Using rule schema (m1) and invoking Lemma 6.9, it follows from this that r_1 $\rhd_{rm}$-reduces to the term $[\![t, ol_2 - (nl_1 - l), nl_2, e_2\{nl_1 - l + 1\}]\!]$ as desired.

Overlap between (m1) and (r5). Let t be the term $[\![[\![\#k, ol_1, nl_1, et :: e_1]\!], ol_2, nl_2, e_2]\!]$ where $k > 1$. Then r_1 is the term

$$[\![\#k, ol_1 + (ol_2 \dot{-} nl_1), nl_2 + (nl_1 \dot{-} ol_2), \{\!\{et :: e_1, nl_1, ol_2, e_2\}\!\}]\!]$$

and r_2 is the term $[\![[\![\#(k - 1), ol_1 - 1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!]$. It is easily seen that both r_1 and r_2 $\rhd_{rm}$-reduce to $[\![\#(k - 1), ol_1 + (ol_2 \dot{-} nl_1) - 1, nl_2 + (nl_1 \dot{-} ol_2), \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]$.

Overlap between (m1) and (r6). Let t be the term $[\![[\![(t_1\ t_2), ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!]$. Then r_1 is the term $[\![(t_1\ t_2), ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]$ where $ol' = (ol_1 + (ol_2 \dot{-} nl_1))$ and $nl' = (nl_2 + (nl_1 \dot{-} ol_2))$, and r_2 is of the term $[\![([\![t_1, ol_1, nl_1, e_1]\!]\ [\![t_2, ol_1, nl_1, e_1]\!]), ol_2, nl_2, e_2]\!]$. It is easily seen that r_1 and r_2 both $\rhd_{rm}$-reduce to $([\![t_1, ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]\ [\![t_2, ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!])$.

Overlap between (m1) and (r7). Let t be the term $[\![[\![(\lambda\, t'), ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!]$. Then r_1 is the term $[\![(\lambda\, t'), ol', nl', \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!]$ where $ol' = (ol_1 + (ol_2 \dot{-} nl_1))$ and $nl' = (nl_2 + (nl_1 \dot{-} ol_2))$, and r_2 is the term $[\![(\lambda\, [\![t', ol_1 + 1, nl_1 + 1, @nl_1 :: e_1]\!]), ol_2, nl_2, e_2]\!]$. Now, using rule schema (r7), we see that $r_1 \rhd^*_{rm} (\lambda\, [\![t', ol' + 1, nl' + 1, @nl' :: \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!])$. Similarly, using rule schemata (r7), (m1) and (m5) and invoking Lemma 4.15, we observe that

$$r_2 \rhd^*_{rm} (\lambda\, [\![t', ol' + 1, nl' + 1, @nl' :: \{\!\{e_1, nl_1 + 1, ol_2 + 1, @nl_2 :: e_2\}\!\}]\!]).$$

Noting that $ind(e_1) \leq nl_1$ and using Lemma 6.10, we conclude that $\{\!\{e_1, nl_1 + 1, ol_2 + 1, @nl_2 :: e_2\}\!\}$ and $\{\!\{e_1, nl_1, ol_2, e_2\}\!\}$ $\rhd_{rm}$-reduce to a common expression. But then so too do r_1 and r_2.

Overlap between (m1) and (m1). Let t be the term $[\![[\![[\![t_1, ol_1, nl_1, e_1]\!], ol_2, nl_2, e_2]\!], ol_3, nl_3, e_3]\!]$. Then r_1 and r_2 are the terms

$$[\![[\![t_1, ol_1, nl_1, e_1]\!], ol_2 + (ol_3 \dot{-} nl_2), nl_3 + (nl_2 \dot{-} ol_3), \{\!\{e_2, nl_2, ol_3, e_3\}\!\}]\!] \quad \text{and}$$
$$[\![[\![t_1, ol_1 + (ol_2 \dot{-} nl_1), nl_2 + (nl_1 \dot{-} ol_2), \{\!\{e_1, nl_1, ol_2, e_2\}\!\}]\!], ol_3, nl_3, e_3]\!].$$

Using rule schema (m1), we see that

$$r_1 \rhd_{rm}^* [\![t_1, ol', nl', \{\!\{e_1, nl_1, ol_2 + (ol_3 \dot{-} nl_2), \{\!\{e_2, nl_2, ol_3, e_3\}\!\}\}\!\}]\!] \tag{5}$$

where $ol' = ol_1 + ((ol_2 + (ol_3 \dot{-} nl_2)) \dot{-} nl_1)$ and $nl' = nl_3 + (nl_2 \dot{-} ol_3) + (nl_1 \dot{-} (ol_2 + (ol_3 \dot{-} nl_2)))$. Similarly,

$$r_2 \rhd_{rm}^* [\![t_1, ol'', nl'', \{\!\{\{\!\{e_1, nl_1, ol_2, e_2\}\!\}, nl_2 + (nl_1 \dot{-} ol_2), ol_3, e_3\}\!\}]\!] \tag{6}$$

where $ol'' = ol_1 + (ol_2 \dot{-} nl_1) + (ol_3 \dot{-} (nl_2 + (nl_1 \dot{-} ol_2)))$ and $nl'' = nl_3 + ((nl_2 + (nl_1 \dot{-} ol_2)) \dot{-} ol_3)$. Using Theorem 6.12 in conjunction with (5) and (6), we see that r_1 and r_2 would $\rhd_{rm}$-reduce to a common expression if $ol' = ol''$ and $nl' = nl''$. But this can be seen to be the case using Lemma 6.7.

All the necessary cases having been considered, the proof of the theorem is complete.

□

As noted already, the following theorem is an immediate consequence:

Theorem 6.14 *The reduction relation $\rhd_{rm}$ is confluent.*

By virtue of Theorem 6.5, Proposition 2.1 and Theorem 6.14, every suspension expression has a unique $\rhd_{rm}$-normal form. It will be convenient to have a special notation for such forms.

Definition 6.15 The $\rhd_{rm}$-normal form of an expression t is denoted by $|t|$.

6.4 Correspondence to de Bruijn terms

A suspension term is intended to encapsulate a de Bruijn term with a 'pending' substitution. We use $\rhd_{rm}$-normal forms and the meta-notation for substitution described in Section 3 to show that this encapsulation is as expected.

Theorem 6.16 *Let $t = [\![t', ol, nl, e]\!]$ be a term and let $e' = |e|$. Then $|t| = S(|t'|; s_1, s_2, s_3, \ldots)$ where*

$$s_i = \begin{cases} \#(i - ol + nl) & \text{if } i > ol \\ \#(nl - m) & \text{if } i \leq ol \text{ and } e'[i] = @m \\ |[\![t_i, 0, nl - m, nil]\!]| & \text{if } i \leq ol \text{ and } e'[i] = (t_i, m). \end{cases}$$

Proof. By induction on t with respect to the well founded ordering relation $\succ$. The argument is based on a consideration of the structure of the term t'.

If t' is a constant: In this case $|t|$ and $S(|t'|; s_1, s_2, s_3, \ldots)$ are both identical to t'.

If t' is a variable reference: Noting the confluence of $\rhd_{rm}$, the desired conclusion in this case follows easily from Lemma 4.13.

If t' is an application: Let $t' = (r_1\ r_2)$. Now $t \rhd_{rm} ([\![r_1, ol, nl, e]\!]\ [\![r_2, ol, nl, e]\!])$ by virtue of rule schema (r6) and, therefore, by the confluence of $\rhd_{rm}$,

$$|t| = (|[\![r_1, ol, nl, e]\!]|\ |[\![r_2, ol, nl, e]\!]|). \tag{7}$$

Additionally, using Lemma 6.4, $t \succ ([\![r_1, ol, nl, e]\!]\ [\![r_2, ol, nl, e]\!])$. Now, for $i = 1$ and $i = 2$,

$$([\![r_1, ol, nl, e]\!]\ [\![r_2, ol, nl, e]\!]) \succ [\![r_i, ol, nl, e]\!]$$

and, by transitivity, $t \succ [\![r_i, ol, nl, e]\!]$. Invoking the hypothesis of the induction,

$$|[\![r_i, ol, nl, e]\!]| = S(|r_i|; s_1, s_2, s_3, \ldots).$$

From this fact used in conjunction with (7) and Definition 3.2, it follows that

$$|t| = S((|r_1|\ |r_2|); s_1, s_2, s_3, \ldots).$$

Noting finally that $|t'| = (|r_1|\ |r_2|)$, the theorem is seen to hold in this case.

If t' is an abstraction: Let $t' = (\lambda\, r)$. Then $|t| = (\lambda\, |[\![r, ol+1, nl+1, @nl :: e]\!]|)$ by virtue of rule schema (r7) and the confluence of $\rhd_{rm}$. By an argument similar to that employed in the case when t' is an application, we also see that $t \succ [\![r, ol+1, nl+1, @nl :: e]\!]$. Using the inductive hypothesis, $|[\![r, ol+1, nl+1, @nl :: e]\!]| = S(|r|; s'_1, s'_2, s'_3, \ldots)$ where

$$s'_i = \begin{cases} \#1 & \text{if } i = 1 \\ \#(i - ol + nl) & \text{if } i > ol + 1 \\ \#(nl + 1 - m) & \text{if } 1 < i \leq (ol+1) \text{ and } e'[i-1] = @m \\ |[\![s, 0, nl + 1 - m, nil]\!]| & \text{if } 1 < i \leq (ol+1) \text{ and } e'[i-1] = (s, m) \end{cases} \tag{8}$$

Noting now that $|t'| = (\lambda\, |r|)$ and using Definition 3.2, we see that

$$S(|t'|; s_1, s_2, s_3, \ldots) = (\lambda\, S(|r|; \#1, S(s_1; \#2, \#3, \#4, \ldots), S(s_2; \#2, \#3, \#4, \ldots), \ldots)). \tag{9}$$

From inspecting (8) and (9), it follows that the theorem would hold in this case if, for $i \geq 1$, $s'_{i+1} = S(s_i; \#2, \#3, \#4, \ldots)$. We show that this must be true by considering several subcases.

(a) $i > ol$. In this case both terms are $\#(i + 1 - ol + nl)$ and hence are identical.

(b) $1 < i \leq ol$ and $e'[i]$ is of the form $@m$. Now both terms are identical to $\#(nl + 1 - m)$.

(c) $1 < i \leq ol$ and $e'[i]$ is of the form (s, m). Here we need to show that

$$|[\![s, 0, nl+1-m, nil]\!]| = S(|[\![s, 0, nl-m, nil]\!]|; \#2, \#3, \#4, \ldots). \tag{10}$$

By virtue of rule schemata (m1) and (m2), $[\![[\![s, 0, nl-m, nil]\!], 0, 1, nil]\!] \rhd^*_{rm} [\![s, 0, nl+1-m, nil]\!]$ and, thus,

$$|[\![s, 0, nl+1-m, nil]\!]| = |[\![[\![s, 0, nl-m, nil]\!], 0, 1, nil]\!]|. \tag{11}$$

Referring to Definition 5.1, we claim that $\eta(t) > \eta([\![[\![s, 0, nl-m, nil]\!], 0, 1, nil]\!])$. This is seen by noting the following: $\eta([\![[\![s, 0, nl-m, nil]\!], 0, 1, nil]\!]) = \mu(s) + 1$, $\eta(t) \geq \mu(s) + \mu((\lambda\, r))$, and $\mu((\lambda\, r)) \geq 2$. It thus follows that $t \succ [\![[\![s, 0, nl-m, nil]\!], 0, 1, nil]\!]$. The inductive hypothesis can therefore be applied to the term on the right of (11). Doing so easily yields (10).

If t' is a suspension: Using Lemma 6.4 and noting that $t' \neq |t'|$, $t \succ [\![|t'|, ol, nl, e]\!]$. Invoking the inductive hypothesis with respect to the latter term and noting that $||t'|| = |t'|$, the theorem follows in this case.

□

7 Correspondence to beta reduction on de Bruijn terms

The β_s-contraction rule schema is intended to be a counterpart in the context of suspension terms of the β-contraction rule schema for de Bruijn terms. Towards stating the correspondence precisely, we note first that the reading and merging rules partition the collection of suspension terms into equivalence classes based on the notion of "having the same $\rhd_{rm}$-normal form." The intention, then, is that the β_s-contraction rule schema have the same effect relative to the *equivalence* classes of suspension terms as does the β-contraction rule schema relative to de Bruijn terms.

We show in this section that the desired correspondence does, in fact, hold. In one direction, this amounts to a relative completeness result for the β_s-contraction rule schema.

Theorem 7.1 *Let t be a de Bruijn term and let $t \rhd_\beta s$. Then there is a suspension term r such that $t \rhd_{\beta_s} r$ and $|r| = s$.*

Proof. By an induction on the structure of t.

Base Case: t is the β-redex rewritten by a β-contraction rule. Let $t = ((\lambda\, t_1)\; t_2)$. By definition,

$$s = S(t_1; t_2, \#1, \#2, \ldots). \tag{1}$$

Now let $r = [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$. Obviously $t \rhd_{\beta_s} r$ and, using Theorem 6.16,

$$|r| = S(|t_1|; |[\![t_2, 0, 0, nil]\!]|, \#1, \#2, \ldots). \tag{2}$$

Noting that t_1 is a de Bruijn term, it follows that $|t_1| = t_1$. Using Theorem 6.16 and noting that t_2 is a de Bruijn term, we similarly see that $|[\![t_2, 0, 0, nil]\!]| = t_2$. Thus, the terms on the righthand sides of (1) and (2) are identical, *i.e.*, $|r| = s$.

Inductive Step: t is an abstraction or an application. The argument in both cases is similar so we consider only the first case. Let $t = (\lambda\, t_1)$. Then $s = (\lambda\, s_1)$ where s_1 is such that $t_1 \rhd_\beta s_1$. By hypothesis, there is a suspension term r_1 such that $t_1 \rhd_{\beta_s} r_1$ and $|r_1| = s_1$. Letting $r = (\lambda\, r_1)$, we see that the requirements of the theorem are satisfied: $|r| = (\lambda\, |r_1|) = (\lambda\, s_1) = s$ and obviously $t \rhd_{\beta_s} r$.

□

In showing the correspondence in the converse direction, it will be necessary to consider the use of the β-contraction rule schema on suspension expressions.

Definition 7.2 The relation on suspension expressions generated by the β-contraction rule schema is denoted by $\rhd_{\beta'}$.

Note that the restriction of $\rhd_{\beta'}$ to suspension terms in $\rhd_{rm}$-normal form is identical to $\rhd_\beta$. The following lemmas, whose proofs are obvious, ensure that $\rhd_{\beta'}$ preserves the lengths of environments and the indices of environments and environment terms. Thus, $\rhd_{\beta'}$ is well defined in that it relates only well formed suspension expressions.

Lemma 7.3 *Let et_1 be an environment term and let et_2 be such that $et_1 \rhd^*_{\beta'} et_2$. Then the following holds: if et_1 is $@m$, then et_2 is $@m$; if et_1 is of the form (t_1, m), then et_2 is of the form (t_2, m). Further, if et_1 is in $\rhd_{rm}$-normal form, then, in the latter case, $t_1 \rhd^*_\beta t_2$.*

Lemma 7.4 *Let e_1 be an environment and let e_2 be such that $e_1 \rhd^*_{\beta'} e_2$. Then $len(e_1) = len(e_2)$. Further, if $len(e_1) > 0$, then the following holds for $1 \leq i \leq len(e_1)$: if $e_1[i]$ is $@m$, then $e_2[i]$ is $@m$; if $e_1[i]$ is of the form (t_1, m), then $e_2[i]$ is of the form (t_2, m). Finally, if e_1 is in $\rhd_{rm}$-normal form, then, in the latter case, $t_1 \rhd^*_\beta t_2$.*

A strengthened form of Theorem 7.1 can be obtained from it by an easy structural induction.

Lemma 7.5 *Let x and y be suspension expressions such that $x \rhd_{\beta'} y$. Then there is a suspension expression z such that $x \rhd_{\beta_s} z$ and $|z| = |y|$.*

Theorem 7.1 shows that each application of the β-contraction rule schema on de Bruijn terms can be mimicked by a *single* use of the β_s-contraction rule schema and some reading and merging steps. Mimicking an application of the β_s-contraction rule schema may, on the other hand, require *several* or *no* uses of the β-contraction rule schema on the underlying de Bruijn term. This reflects the fact that the use of environments may foster a sharing of β-redexes or, alternatively, may result in temporarily maintaining β-redexes that would not appear in the term if the substitution were carried out completely. The important point to note, however, is that a β_s-contraction *can* be simulated by a sequence of β-contractions, *i.e.*, the β_s-contraction schema is relatively sound. This follows from Theorem 7.9 whose proof uses the intervening lemmas.

Lemma 7.6 *Let t_1 be a term in $\rhd_{rm}$-normal form and let t_2 be such that $t_1 \rhd^*_{\beta'} t_2$. Further, let e_1 be an environment in $\rhd_{rm}$-normal form and let e_2 be such that $e_1 \rhd^*_{\beta'} e_2$. Then*

$$|[\![t_1, ol, nl, e_1]\!]| \rhd^*_{\beta} |[\![t_2, ol, nl, e_2]\!]|.$$

Proof. We note, using Theorem 6.16 and Corollary 3.6, that if s_1 and s_2 are de Bruijn terms such that $s_1 \rhd^*_{\beta} s_2$, then $|[\![s_1, 0, n, nil]\!]| \rhd^*_{\beta} |[\![s_2, 0, n, nil]\!]|$. Using Theorem 6.16 and Lemma 7.4 in conjunction with the assumptions of the lemma, it follows easily that

$$|[\![t_1, ol, nl, e_1]\!]| = S(u_0; u_1, u_2, u_3, \ldots) \quad \text{and} \quad |[\![t_2, ol, nl, e_2]\!]| = S(v_0; v_1, v_2, v_3, \ldots),$$

where, for $i \geq 0$, $u_i \rhd^*_{\beta} v_i$. But then, by Corollary 3.6, $|[\![t_1, ol, nl, e_1]\!]| \rhd^*_{\beta} |[\![t_2, ol, nl, e_2]\!]|$.

□

Lemma 7.7 *Let et_1 be an environment term in $\rhd_{rm}$-normal form and let et_2 be an expression such that $et_1 \rhd^*_{\beta'} et_2$. Further, let e_1 be an environment in $\rhd_{rm}$-normal form and let e_2 be such that $e_1 \rhd^*_{\beta'} e_2$. Then $|\langle\!\langle et_1, nl, ol, e_1 \rangle\!\rangle| \rhd^*_{\beta'} |\langle\!\langle et_2, nl, ol, e_2 \rangle\!\rangle|$.*

Proof. An easy consequence of Lemmas 7.3, 7.4, 4.14, 4.15 and 7.6.

□

Lemma 7.8 *Let e_1 and e_2 be environments in $\rhd_{rm}$-normal form and let e_1' and e_2' be such that $e_1 \rhd^*_{\beta'} e_1'$ and $e_2 \rhd^*_{\beta'} e_2'$. Then $|\{\!\{e_1, nl, ol, e_2\}\!\}| \rhd^*_{\beta'} |\{\!\{e_1', nl, ol, e_2'\}\!\}|$.*

Proof. By induction on $len(e_1)$. If $len(e_1) = 0$, then e_1 and e_1' are both nil. Then, by Lemma 4.16, either $|\{\!\{e_1, nl, ol, e_2\}\!\}|$ and $|\{\!\{e_1', nl, ol, e_2'\}\!\}|$ are both nil, or, for some k, $|\{\!\{e_1, nl, ol, e_2\}\!\}| = e_2\{k\}$ and $|\{\!\{e_1', nl, ol, e_2'\}\!\}| = e_2'\{k\}$. The desired conclusion follows easily in either case. If $len(e_1) > 0$, let $e_1 = et_1 :: t_1$, noting that et_1 and t_1 must be in $\rhd_{rm}$-normal form. Then e_1' must be of the form $et_1' :: t_1'$ where $et_1 \rhd^*_{\beta'} et_1'$ and $t_1 \rhd^*_{\beta'} t_1'$. Using rule schema (m5),

$$|\{\!\{e_1, nl, ol, e_2\}\!\}| = |\langle\!\langle et_1, nl, ol, e_2 \rangle\!\rangle| :: |\{\!\{t_1, nl, ol, e_2\}\!\}|, \quad \text{and}$$
$$|\{\!\{e_1', nl, ol, e_2'\}\!\}| = |\langle\!\langle et_1', nl, ol, e_2' \rangle\!\rangle| :: |\{\!\{t_1', nl, ol, e_2'\}\!\}|.$$

The lemma now follows from Lemma 7.7 and the inductive hypothesis.

□

Theorem 7.9 *Let t and s be suspension expressions such that $t \rhd_{\beta_s} s$. Then $|t| \rhd^*_{\beta'} |s|$.*

Proof. By induction on t with respect to $\succ$. Note that t cannot be a constant, a variable reference, nil or of the form $@m$. The remaining cases for the structure of t are considered below.

If t is an application: There are two possibilities: t is the redex rewritten by a β_s-contraction rule or some proper subterm of t is rewritten. We analyze each possibility separately.

In the first subcase, t has the form $((\lambda\, t_1)\; t_2)$. We note first that $|t| = ((\lambda\, |t_1|)\; |t_2|)$. Further,

$s = [\![t_1, 1, 0, (t_2, 0) :: nil]\!]$.

Using Theorem 6.16, it can be seen that $|s| = S(|t_1|; |t_2|, \#1, \#2, \ldots)$, *i.e.*, that $|t| \rhd_{\beta'} |s|$.

In the second subcase, t is of the form $(t_1\ t_2)$. We assume, without loss of generality, that the redex rewritten is a subterm of t_1. Then $s = (s_1\ t_2)$, where $t_1 \rhd_{\beta_s} s_1$. Since t_1 is a proper subterm of t, $t \succ t_1$. Thus, by hypothesis, $|t_1| \rhd^*_{\beta'} |s_1|$. The theorem now follows from noting that $|t| = (|t_1|\ |t_2|)$ and $|s| = (|s_1|\ |t_2|)$.

If t is an abstraction or has the form (t', m) or $et :: e$: An inductive argument similar to that in the second subcase of an application can be used in each of these cases.

If t is a suspension: Let $t = [\![r, ol, nl, e]\!]$. Then $s = [\![r', ol, nl, e']\!]$ where $r \rhd_{\beta_s} r'$ and $e = e'$ or $r = r'$ and $e \rhd_{\beta_s} e'$. In either case, using the fact that r and e are proper subexpressions of t and hence $t \succ r$ and $t \succ e$, $|r| \rhd^*_{\beta'} |r'|$ and $|e| \rhd^*_{\beta'} |e'|$. Now, by confluence of $\rhd^*_{rm}$,

$$|[\![r, ol, nl, e]\!]| = |[\![|r|, ol, nl, |e|]\!]| \quad \text{and} \quad |[\![r', ol, nl, e']\!]| = |[\![|r'|, ol, nl, |e'|]\!]|.$$

Using Lemma 7.6, it follows from this that $|[\![r, ol, nl, e]\!]| \rhd^*_{\beta} |[\![r', ol, nl, e']\!]|$. Recalling that $\rhd_{\beta'}$ and $\rhd_{\beta}$ are identical on de Bruijn terms, the theorem is seen to be true.

If t has the form $\langle\!\langle et, nl, ol, e\rangle\!\rangle$: By an argument similar to that used for a suspension, s must be of the form $\langle\!\langle et', ol, nl, e'\rangle\!\rangle$ where $|et| \rhd^*_{\beta'} |et'|$ and $|e| \rhd^*_{\beta'} |e'|$. Noting that

$$|\langle\!\langle et, nl, ol, e\rangle\!\rangle| = |\langle\!\langle |et|, nl, ol, |e|\rangle\!\rangle| \quad \text{and} \quad |\langle\!\langle et', nl, ol, e'\rangle\!\rangle| = |\langle\!\langle |et'|, nl, ol, |e'|\rangle\!\rangle|,$$

and using Lemma 7.7, the theorem follows in this case.

If t is of the form $\{\!\{e_1, nl, ol, e_2\}\!\}$: Once again, s must be of the form $\{\!\{e'_1, nl, ol, e'_2\}\!\}$ where, $|e_1| \rhd^*_{\beta'} |e'_1|$ and $|e_2| \rhd^*_{\beta'} |e'_2|$. We note further that

$$|\{\!\{e_1, nl, ol, e_2\}\!\}| = |\{\!\{|e_1|, nl, ol, |e_2|\}\!\}| \quad \text{and} \quad |\{\!\{e'_1, nl, ol, e'_2\}\!\}| = |\{\!\{|e'_1|, nl, ol, |e'_2|\}\!\}|.$$

The theorem now follows from Lemma 7.8.

All possibilities for the structure of t having been considered, the proof of the theorem is complete.

□

The results of this section can be used to conclude that the rule schemata in Figures 1–3 correctly implement β-reduction. The following theorem is a generalization of this observation.

Theorem 7.10

(a) If x and y are suspension expressions such that $x \rhd^*_{rm\beta_s} y$, then $|x| \rhd^*_{\beta'} |y|$.

(b) If x and y are suspension expressions in $\rhd_{rm}$-normal form such that $x \rhd^*_{\beta'} y$ then $x \rhd^*_{rm\beta_s} y$.

Proof. *(a)* By an induction on the length of the reduction sequence by which $x \rhd^*_{rm\beta_s} y$. If the first rule used is an instance of the β_s-contraction rule schema, we use Theorem 7.9. Otherwise we use Theorem 6.14 to note that the $\rhd_{rm}$-normal form is preserved.

(b) An induction on the length of the reduction sequence by which $x \rhd^*_{\beta'} y$. It is only necessary to show that if $x \rhd_{\beta'} y$, then $x \rhd^*_{rm\beta_s} y$. This follows from Lemma 7.5 by noting that y must be in $\rhd_{rm}$-normal form and using the fact that $\rhd_{rm}$ is confluent and noetherian.

□

8 Some reduction properties of the overall system

The results of the previous two sections can be used to observe some properties of reduction within our system of rewrite rules. The most important of these properties is that of confluence.

Theorem 8.1 *The reduction relation $\rhd_{rm\beta_s}$ is confluent.*

Proof. This is evident from the diagram below.

$$\begin{tikzcd}
t \arrow[rr, "\rhd^*_{rm\beta_s}"] \arrow[dr, dashed, "\rhd^*_{rm}"] \arrow[dd, "\rhd^*_{rm\beta_s}"'] \arrow[rr, phantom, shift right=6ex, "(1)"] & & s \arrow[dr, dashed, "\rhd^*_{rm}"] & & \\
& {|t|} \arrow[rr, dashed, "\rhd^*_{\beta'}"] \arrow[dd, dashed, "\rhd^*_{\beta'}"'] \arrow[l, phantom, shift right=4ex, "(2)"] \arrow[ddrr, phantom, "(3)"] & & {|s|} \arrow[dd, dashed, "\rhd^*_{\beta'}"'] \arrow[dr, dashed] \arrow[ddr, phantom, "(5)"] & \\
r \arrow[dr, dashed, "\rhd^*_{rm}"'] & & & & {} \arrow[dl, dashed, "\rhd^*_{rm\beta_s}"] \\
& {|r|} \arrow[rr, dashed, "\rhd^*_{\beta'}"] \arrow[dr, dashed] \arrow[drr, phantom, "(4)"] & & u & \\
& & {} \arrow[ur, dashed, "\rhd^*_{rm\beta_s}"'] & &
\end{tikzcd}$$

In diagrams of this kind, dashed arrows signify the existence of reductions given by the labels on the arrows, depending on the reductions depicted by the solid arrows. The dashed arrows in the faces (1) and (2) are justified by Theorem 7.10, the remaining dashed arrows in face (3) are justified by a straightforward extension of Proposition 3.7 to $\rhd^*_{\beta'}$ and the last two dashed arrows in faces (4) and (5) are justified by Theorem 7.10.

□

Another observation concerns the redundancy in certain contexts of the merging rules. These rules have efficiency advantages in that they support the combination of substitution walks over terms. However, they are not essential to the implementation of β-reduction.

Lemma 8.2 *Let t be a de Bruijn term and let $t \rhd_\beta s$. Then $t \rhd^*_{r\beta_s} s$.*

Proof. By Theorem 7.1, $t \rhd_{\beta_s} r$ where $|r| = s$. We observe now that t, being a de Bruijn term, is a simple expression. From this it follows that r is also a simple expression. It is also easily seen that (a) a reading rule must be applicable to any simple expression that is not in $\rhd_{rm}$-normal form, and (b) applying such a rule produces another simple expression. Thus $r \rhd^*_r |r|$, *i.e.*, $r \rhd^*_r s$. This implies that $t \rhd^*_{r\beta_s} s$.

□

Theorem 8.3 *Let t and s be de Bruijn terms such that $t \rhd^*_\beta s$. Then $t \rhd^*_{r\beta_s} s$.*

Proof. By induction on the length of the $\rhd_\beta$-reduction sequence, using Lemma 8.2.

□

It is not possible to eliminate uses of merging rules from all $\rhd_{rm\beta_s}$-reduction sequences. However, when starting from a simple expression, merging rules are redundant if the objective is to produce an expression in the same $\rhd^*_{rm}$ equivalence class as the final expression that was originally produced.

Theorem 8.4 *Let t be a simple expression and let s be such that $t \rhd^*_{rm\beta_s} s$. Then there is an expression u such that $t \rhd^*_{r\beta_s} u$ and $s \rhd^*_{rm} u$.*

Proof. Letting u be the expression $|s|$, the lemma is evident from the diagram below.

$$\begin{tikzcd}
t \arrow[rr, "\rhd^*_{rm\beta_s}"] \arrow[d, dashed, "\rhd^*_r"'] \arrow[drr, phantom, "(1)"] & & s \arrow[d, dashed, "\rhd^*_{rm}"] \\
{|t|} \arrow[rr, dashed, "\rhd^*_{\beta'}"] \arrow[dr, dashed, no head] \arrow[rr, phantom, "(2)", shift right=6] & & {|s|} \\
& {} \arrow[ur, dashed, "\rhd^*_{r\beta_s}"'] &
\end{tikzcd}$$

The dashed arrows in the face labelled (1) in this figure are justified by Theorem 7.10; the label $\rhd^*_r$ on the arrow from t to $|t|$ is warranted by the observation (made in the proof of Lemma 8.2) that a simple expression can be reduced to its $\rhd_{rm}$-normal form by using only reading rules. The remaining dashed arrow in face (2) is justified by Theorem 8.3.

□

The arguments in this section use the 'projection' of suspension terms onto de Bruijn terms that follows from the results of Sections 6 and 7 in showing properties of our system. This method of argument is similar in spirit to the one referred to as the *interpretation method* in [17] and used in [17] and [39] in proving confluence properties of a combinator calculus. We use this method again in [29].

9 Conclusion

We have described in this paper a notation for the terms in a lambda calculus and a system for rewriting expressions in this notation. Our notation is based on the de Bruijn representation of lambda terms but embellishes this so as to allow for the representation of a term with a pending substitution. We have shown that the rewrite rules in our system can simulate the operation of β-reduction on terms in the usual representation and can, in a sense, be simulated by this operation. We have used this observation in establishing the confluence of our overall system. The notation developed here has several useful features. It is closely related to the usual representation of lambda terms and can in fact replace the latter notation even in contexts where intensions of terms have to be manipulated. The use of de Bruijn's scheme for representing variables obviates α-conversion in comparing terms. Our rewrite system provides a fine-grained control over the substitution process involved in β-contraction, and thus can be used as the basis for a wide variety of reduction procedures. Furthermore, the ability our notation provides to suspend substitutions leads to efficiency advantages in the implementation of β-reduction: substitution and reduction walks over the structures of terms can be combined and substitutions can be delayed in some cases till such a point that it becomes unnecessary to perform them. Finally, our notation permits components of a β-contraction step to be intermingled with other operations such as those involved in unifying lambda terms. This ability is of practical relevance and is, in fact, being used to advantage in an implementation of the language λProlog.

While the specific notation presented here is new, the ideas embedded in it have received previous and parallel developments. A central idea in our notation is the use of environments in representing suspended substitutions. This idea is an old one within the implementation of β-reduction to the extent that it is difficult to pinpoint a source for it. The category of terms that we have referred to as suspensions in this paper are what are usually called *closures*. However, most of these proposals have differed from that presented in this paper in two important respects. First, the idea of closures has been used largely as an implementation device and an attempt has not been made to reflect it into the notation or to describe a calculus that takes the resulting notation seriously. Second, in most cases the focus has been on generating *weak head normal forms*, *i.e.*, the percolation of substitutions or the rewriting of β-redexes under abstractions is not considered. The latter assumption has the effect of greatly simplifying the kind of notation required, as the reader may well verify. Moreover, as discussed already, this is not an assumption that is valid in all contexts.

In our knowledge, the first serious consideration of a notation and a calculus that incorporate a fine-grained control over substitutions appears in the work of P-L. Curien [10, 11]. In this work, a categorical combinatory logic called **CCL** is described. The language underlying this logic is not the lambda calculus, but bears a close relationship to it: there is a translation from the (pure) lambda calculus augmented with the pairing function to **CCL** and vice versa that preserves the intended equality relation in the two calculi. Unfortunately, the rewrite rules that constitute **CCL** are not confluent [17]; this result might be anticipated from the fact that the lambda calculus with the pairing function is not confluent [24]. However, a subset of **CCL** terms can be exhibited on which the rewrite rules are confluent [17, 39]. Moreover, a subclass of this class of terms is isomorphic to the class of lambda calculus terms and this isomorphism can be extended to one between a subset of **CCL** rules and β-reduction [17]. An interesting characteristic of this subsystem is that it permits 'β-contraction' to be factored into the generation of a substitution and the subsequent percolation of this substitution in much the spirit of the system described in this paper.

While the **CCL** system has several desirable features, its relationship to the lambda calculus is a somewhat complex one. More recently, the general ideas embedded in **CCL** have been used in conjunction with notations that are more directly based on the lambda calculus in [1] and [13]. The resulting systems are very similar to the one described here and our work, in fact, represents a concurrent and independent development of these general ideas.[6] At a level of detail, the notations in [1] and [13] are practically indistinguishable. However, they differ from our notation in two respects. The first of these is in the manner in which variables are represented. In our notation, these are represented directly by de Bruijn numbers. In contrast, in the other notations, variables are represented essentially as environment transforming operators that strip off parts of environments. The latter representation has the virtue of parsimony: a smaller vocabulary suffices and the rules that serve to combine environments can also be used to determine the bindings for variables. However, there are also advantages to our representation. As one example, the comparison of terms containing variables becomes somewhat easier. At a different level, there is a differentiation of rules in our system based on purpose, and this makes it easier to identify simpler, but yet complete, subsystems. Thus, as observed in Theorem 8.3, the rules for merging environments can be omitted from our system without losing the ability to simulate β-reduction. A similar observation cannot be made about the other systems being discussed.[7]

The second respect in which our notation differs from the ones in [1] and [13] is the manner

[6]The ideas described here are an outgrowth of those contained in [32]. The present exposition of these ideas has, however, been influenced by [1].

[7]We note in this context that the remark in [1] to the effect that the rule for merging environments (labelled (Clos)) can be eliminated is incorrect. However, as pointed out to us by P.-L. Curien, restricted versions of this rule and of other environment manipulating rules suffice from the perspective of simulating β-reduction in the notation presented there.

in which it encodes the adjustment that must be made to indices of terms in an environment. In our notation, this is not maintained explicitly but is obtained from the difference between the embedding level of the term that has to be substituted into and an embedding level recorded with the term in the environment. Thus, consider a suspension term of the form $[\![t_1, 1, nl, (t_2, nl') :: nil]\!]$. This represents a term that is to be obtained by substituting t_2 for the first free variable in t_1 (and modifying the indices for the other free variables). However, the indices for the free variables in t_2 must be 'bumped up' by $(nl - nl')$ before this substitution is made. In the other systems, the needed increment to the indices of free variables is maintained explicitly with the term in the environment. Thus, the suspension term shown above would be represented, as it were, as $[\![t_1, 1, nl, (t_2, (nl - nl')) :: nil]\!]$; actually, the old and new embedding levels are needed in this term only for determining the adjustment to the free variables in t_1 with indices greater than the old embedding level, and devices for representing environments encapsulating such an adjustment simplify the actual notation used. The representation used in [1] and [13] have the benefit of parsimony: no special syntax is required for environment terms and rules that are used for manipulating terms can also be used for manipulating terms in the environment. Notice, however, that the rule for moving substitutions under abstractions becomes more complex in that *every* term in the environment is now affected. Thus, from a term of the form $[\![(\lambda\, t_1), 1, nl, (t_2, (nl - nl')) :: nil]\!]$, this rule must produce a term that looks something like $(\lambda\, [\![t_1, 2, nl + 1, @1 :: (t_2, nl - nl' + 1) :: nil]\!])$. In contrast, using our representation, this rule is required only to add a 'dummy' element to the environment and to make a *local* change to the embedding levels of the overall term. On a balance, the trade-offs in the two approaches appear to be even in the context of the overall rewriting systems. However, our representation seems to have an advantage if a simpler rewriting system, such as that obtained by eliminating the merging rules, is used.

In a different direction, the general idea of delaying substitutions appears to have been anticipated by de Bruijn in [3] and [4]. In the latter paper, de Bruijn actually presents a notation for lambda terms that includes mappings for transforming variable indices within terms. The specific notation presented in [4] is quite cumbersome and, in addition, does not include any mechanisms for encoding the substitution operation needed for β-contraction. However, a special form of the general substitution operation that suffices for β-contraction has been described in the literature, and using laziness in its implementation results in a notation close to the one presented here. In particular, β-contraction is described in [17] by means of a binary function σ_n and a unary function τ_i^n on terms. These functions perform the following tasks: $\sigma_n(t_1, t_2)$ produces a term from t_1 by decreasing the indices for the $(n+1)$-st and later free variables by 1 and replacing the n-th free variable by t_2 after the indices for the free variables in t_2 have been 'bumped up' by n; $\tau_i^n(t)$ produces the term that results from t by raising the indices for the i-th and later free variables in it by n. A similar set of functions is described by Staples in [38]. Our notion of a suspension collapses these two functions into a common form and captures the effect of evaluating them in a delayed fashion. It is interesting to note that two indices *ol* and *nl are* needed in a term of the form $[\![t, ol, nl, e]\!]$ to achieve this objective; an attempt to use only one index was made in [33] but could not be carried out to completion. We also observe that our notation actually generalizes the mentioned functions by allowing for environments that represent *multiple* non-dummy substitutions that are to be performed simultaneously.

The notation studied in this paper is intended to have practical utility. Our particular desire is that this notation serve as a substrate upon which coarser-grained representations for lambda terms may be developed that are eventually used in actual implementations. We explore this issue in a companion paper [29]. One particular refinement we consider is that of eliminating the merging rules. These rules have a practical advantage in that it is only through them that substitution walks over the structure of a term can be combined. However, implementing these rules in their full generality can be cumbersome. Our approach to this is to capture some of their effects through auxiliary rules. The resulting rewrite system permits us to restrict our attention to only simple expressions. Another refinement consists of adding annotations to terms that determine whether or not they can be affected by substitutions generated by external β-contractions. We then use the refined notation to describe manipulations to lambda terms and to prove properties of such manipulations. It is this work that directly underlies the implementation that is being developed for λProlog [30].

Acknowledgements

We are grateful to P.-L. Curien for comments on an earlier version of this paper. Suggestions from reviewers have lead to significant improvements in presentation. John Hannan and an reviewer of an early presentation of the ideas here (in [32]) made us aware of related research. Stimulus was provided to the first author by Mike O'Donnell and his students by their participation in an exposition of these ideas in Spring 1991. Work on this paper has been supported by NSF grants CCR-89-05825 and CCR-92-08465.

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] L. Aiello and G. Prini. An efficient interpreter for the lambda-calculus. *The Journal of Computer and System Sciences*, 23:383–425, 1981.

[3] N. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Math.*, 34(5):381–392, 1972.

[4] N. de Bruijn. Lambda-calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Math.*, 40:348–356, 1978.

[5] N. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, 1980.

[6] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[7] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[8] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[9] G. Cousineau, P-L. Curien, and M. Mauny. The categorical abstract machine. *The Science of Programming*, 8(2):173–202, 1987.

[10] P-L. Curien. Categorical combinators. *Information and Control*, 69:188–254, 1986.

[11] P-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986.

[12] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[13] John Field. On laziness and optimality in lambda interpreters: Tools for specification and analysis. In *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 1–15. ACM Press, January 1990.

[14] Jean H. Gallier. What's so special about Kruskal's theorem and the ordinal Γ_0? A survey of some results in proof theory. *Annals of Pure and Applied Logic*, 53:199–260, 1991.

[15] Michael J. Gordon, Arthur J. Milner, and Christopher P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[16] Paul R. Halmos. *Naive Set Theory.* D. Van Nostrand Company, Inc., 1960.

[17] Thérèse Hardin. Confluence results for the pure strong categorical logic CCL. λ-calculi as subsystems of CCL. *Theoretical Computer Science*, 65:291–342, 1989.

[18] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[19] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinatory Logic and Lambda Calculus*. Cambridge University Press, 1986.

[20] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[21] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.

[22] Gérard Huet. Formal structures for computation and deduction. Unpublished course notes, Carnegie Mellon University, 1986.

[23] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[24] J.W. Klop. *Combinatory Reduction Systems*. Mathematisch Centrum, Amsterdam, 1980.

[25] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[26] J.B. Kruskal. Well-quasi-ordering, the tree theorem and Vázsonyi's conjecture. *Trans. Amer. Math. Soc.*, 95:210–225, 1960.

[27] Azriel Levy. *Basic Set Theory*. Springer-Verlag, 1979.

[28] Dale Miller and Gopalan Nadathur. A logic programming approach to manipulating formulas and programs. In Seif Haridi, editor, *IEEE Symposium on Logic Programming*, pages 379–388, San Francisco, September 1987.

[29] Gopalan Nadathur. A fine-grained notation for lambda terms and its use in intensional operations. Technical Report TR-96-13, Department of Computer Science, University of Chicago, May 1996. To appear in *Journal of Functional and Logic Programming*.

[30] Gopalan Nadathur, Bharat Jayaraman, and Debra Sue Wilson. Implementation considerations for higher-order features in logic programming. Technical Report CS-1993-16, Department of Computer Science, Duke University, June 1993.

[31] Gopalan Nadathur and Dale Miller. An overview of λProlog. In Kenneth A. Bowen and Robert A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827, Seattle, Washington, August 1988. MIT Press.

[32] Gopalan Nadathur and Debra Sue Wilson. A representation of lambda terms suitable for operations on their intensions. In *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 341–348. ACM Press, 1990.

[33] Michael J. O'Donnell and Robert I. Strandh. Towards a fully parallel implementation of the lambda calculus. Technical Report JHU/EECS-84/13, Johns Hopkins University, 1984.

[34] Lawrence C. Paulson. The representation of logics in higher-order logic. Technical Report Number 113, University of Cambridge, Computer Laboratory, August 1987.

[35] Lawrence C. Paulson. The foundations of a generic theorem prover. Technical Report Number 130, University of Cambridge, Computer Laboratory, March 1988.

[36] Frank Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–322, Pacific Grove, California, June 1989. IEEE Computer Society Press.

[37] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, June 1988.

[38] John Staples. A new technique for analysing parameter passing, applied to the lambda calculus. *Australian Computer Science Communications*, 3(1):201–210, May 1981.

[39] Hirofumi Yokouchi. Church-Rosser Theorem for a rewriting system on categorical combinators. *Theoretical Computer Science*, 65:271–290, 1989.