The Journal of Functional and Logic Programming

The MIT Press

Volume 1999, Article 2
15 March, 1999

ISSN 1080–5230. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493, USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://www.cs.tu-berlin.de/journal/jflp/*
- *http://mitpress.mit.edu/JFLP/*
- *gopher.mit.edu*
- *ftp://mitpress.mit.edu/pub/JFLP*

The Journal of Functional and Logic Programming is a peer-reviewed and electronically published scholarly journal that covers a broad scope of topics from functional and logic programming. In particular, it focuses on the integration of the functional and the logic paradigms as well as their common foundations.

A Fine-Grained Notation for Lambda Terms and Its Use in Intensional Operations

Gopalan Nadathur

15 March, 1999

Abstract

We discuss issues relevant to the practical use of a previously proposed notation for lambda terms in contexts where the intensions of such terms have to be manipulated. This notation uses the "nameless" scheme of de Bruijn, includes expressions for encoding terms together with substitutions to be performed on them, and contains a mechanism for combining such substitutions so that they can be effected in a common structure traversal. The combination mechanism is a general one and consequently difficult to implement. We propose a simplification to it that retains its functionality in situations that occur commonly in β-reduction. We then describe a system for annotating terms to determine if they can be affected by substitutions generated by external β-contractions. These annotations can lead to a conservation of space and time in implementations of reduction by permitting substitutions to be performed trivially in certain situations and can also foster a greater sharing in reduction. The use of the resulting notation in the reduction and comparison of terms is examined. Notions of head normal forms and head reduction sequences are defined in context and shown to be useful in equality computations. Our head reduction sequences generalize the usual ones for lambda terms so that they subsume the sequences of terms produced by a variety of graph- and environment-based reduction procedures for the lambda calculus. They can therefore be used in correctness arguments for such procedures. This fact and the efficacy of our notation are illustrated in the context of a particular reduction procedure that we present. The relevance of the present discussions to the unification of lambda terms is also outlined.

1 Introduction

We are concerned in this paper with a representation for lambda terms that might be employed when these terms are used as data structures. There are significant reasons for interest in this matter: lambda terms provide a convenient means for representing objects whose structures incorporate the notion of binding [Chu40] and are used for this purpose in a number of computer systems and programming languages that support the manipulation of formulas, programs, proofs, and other similar objects [Bru80, CAB$^+$86, CH88, GMW79, HHP93, NM88, Pau90, Pfe89]. In a sense specifically pertinent to this paper, objects are represented directly by lambda terms in systems like L_λ [Mil91], λProlog [NM88], Isabelle [Pau90] and Elf [Pfe89], to be manipulated by some form of higher-order unification [Hue75, Nip93]. The considerations here are motivated by implementation questions that arise in the context of such systems, particularly in the context of λProlog.

The choice of a suitable representation for lambda terms is obviously governed by the operations that need to be performed on them in the intended realm of application. The comparison of the structures of terms is intrinsic to several of these operations and a satisfactory representation must, therefore, make these readily available. At a level of detail, such comparisons must ignore differences between terms that are based on the names of bound variables. Thus, equality of lambda terms up to α-convertibility must be easy to determine. Finally, an operation of special significance is β-reduction — this operation provides an encoding of substitution and is useful in realizing several higher-level operations. An acceptable representation must enable this operation to be performed efficiently.

In a previous paper [NW98], we have described a notation for lambda terms called the *suspension* notation that provides a basis for meeting these various requirements.[1] This notation is similar in spirit to those proposed in [ACCL91] and [Fie90], based on the Categorical Combinators of Curien [Cur86]. It also shares features with data structures that have been used in implementing β-reduction, most notably those in [AP81]. At a level of detail, the suspension notation uses a scheme suggested by de Bruijn [Bru72] for eliminating variable names from lambda terms. Further, it incorporates a generalized notion of an environment as a mechanism for delaying substitu-

[1] This paper and [NW98] represent a complete development of ideas first presented in [NW90].

tions.[2] The ability to perform substitutions lazily, and the reflection of this ability into the notation, has several potential advantages. First, delaying substitutions arising from different β-contractions makes it possible to combine them and hence to perform them in one structure traversal.[3] Second, carrying substitutions out incrementally can lead to conservation of work in situations where their effects do not have to be computed in entirety. Finally, the explicit treatment of the substitution process in the notation makes it easy to describe a wide variety of reduction procedures and to verify the correctness of these procedures.

In this paper we describe modifications to the suspension notation with an eye to its practical use. This notation includes a mechanism for combining substitutions that is quite general but also difficult to implement as such. We therefore propose a simplification of this mechanism that retains its functionality in situations that occur commonly in β-reduction while making it more implementable. We then refine the resulting notation by adding annotations to terms that indicate whether or not they can be affected by substitutions generated by external β-contractions. These annotations are intended to be added to terms at the outset by a preprocessing phase. Our rewrite rules then conspire to use the annotations and to preserve them in the course of reducing expressions. One virtue of the annotations is that they permit substitutions to be carried out trivially in certain situations. Effecting substitutions in this manner can also have other benefits: it can lead to a conservation of space and can foster a greater sharing of work in a graph-based implementation of reduction that uses our notation. These benefits can be significant in practice, *e.g.*, in the implementation of λProlog [BR91].

This paper also considers the use of the notation developed in the comparison of lambda terms. This task is of special significance to us because it

[2] As discussed in [NW98], the main difference between the suspension notation and those in [ACCL91] and [Fie90] is in the way they encode the changes that must be made to a term in an environment when this term is actually substituted into a particular context. We also conjecture that our notation preserves strong normalization, unlike the other calculi [Mel95].

[3] The suspension notation is more general than other recently proposed calculi such as $\lambda\upsilon$ [BBLRD96], $\lambda\zeta$ [Muñ96], and λs_e [KR97] in that it has mechanisms for realizing such combinations. (With respect to the λs_e calculus, we observe that the suspension notation can be augmented with metavariables over terms without affecting its known confluence and strong normalization properties.) The simplification studied in this paper retains some of this generality, at least in comparison with $\lambda\upsilon$ and $\lambda\zeta$.

is a part of several operations on lambda terms that become important when they are employed as representational devices. The comparison of lambda terms is usually based on their head normal forms. As a first step, we lift the notion of a head normal form to the new notation and show that it can be used instead in carrying out comparisons. Following this course permits the benefits of laziness in substitution to be realized more fully in the comparison process. We also describe the idea of head reduction sequences that are intended to produce head normal forms for terms in our notation, and we show that these sequences terminate whenever the usual head reduction sequences on the underlying lambda terms do. Our notion of head reduction sequences is actually of *independent* interest. It generalizes the usual notion in such a way that it subsumes the sequences produced by a variety of graph-based and environment-based reduction procedures for lambda terms and can therefore be used in proving the correctness of these procedures. We illustrate this capability by using our notion to establish the correctness of a particular reduction procedure that we present. This procedure is similar to interpreters for the lambda calculus proposed by Henderson and Morris [HM76] and Field [Fie90]; it differs from these mainly in that it finds head normal forms instead of only weak head normal forms. It is also closely related to the simplifier presented by Aiello and Prini [AP81]. With respect to the latter, we note that our notation facilitates a *complete* proof of correctness of our reduction procedure and also permits an intermingling of normalization operations with operations such as comparisons of terms. Benefits such as these have also been noted for the $\lambda\sigma$-calculus [ACCL91] and ΛCCL [Fie90] and have been exploited recently in the description of a higher-order unification procedure [DHK95].

The rest of this paper is structured as follows. The next two sections summarize notions pertaining to rewrite systems, the de Bruijn notation, and the suspension notation that are relevant to this paper; the presentation of the full suspension notation here is justified by its usefulness in understanding the nature of later refinements and in proving properties about them. In Section 4, we describe our simplification of the suspension notation. In Section 5 we propose a notion of closedness for lambda terms that is intended to capture independence from external abstractions and show how it can be used in the reduction process. We incorporate the discussions of the preceding two sections into the suspension notation in Section 6 and establish a correspondence between the resulting rewrite system and β-reduction. In Section 7 we consider the use of our notation in comparing lambda terms. We conclude

the paper in Section 8 with remarks on the pertinence of the discussions here to a situation where the unification of lambda terms is considered.

2 Preliminaries

We outline in this section the "nameless" notation for lambda terms proposed by de Bruijn [Bru72] and describe notions pertaining to it that are used in subsequent discussions. The mechanism of rewrite systems is employed in presenting operations on terms in this and other notations, and we therefore first recapitulate the relevant vocabulary.

2.1 Terminology pertaining to rewrite systems

The rewrite systems that we shall be concerned with in this paper are each 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. Such a schema typically contains occurrences of metalanguage variables that range over indicated categories of expressions. Particular rules may be obtained from the schema by choosing suitable instantiations for 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. The relation defined by a collection of rule schemata is the union of the relations defined by each schema in the collection. Let $\triangleright$ denote such a relation. We express the fact that t is related to s by virtue of $\triangleright$ by writing $t \triangleright s$. We refer to occurrences in expressions of instances of the lefthand sides of the rule schemata defining $\triangleright$ as $\triangleright$-*redexes*. The correspondence to a particular rule schema may also be expressed by referring to the subexpression as a redex occurrence of the schema. The reflexive and transitive closure of $\triangleright$ is denoted by $\triangleright^*$, a relation that is, once again, written in infix form. Intuitively, $t \triangleright^* s$ signifies that t can be rewritten to s by a (possibly empty) sequence of applications of the relevant rule schemata. Accordingly, we refer to the relation $\triangleright$ as a *rewrite* or *reduction* relation, and we say that t $\triangleright$-reduces to s if $t \triangleright^* s$.

Suppose that we are given a collection of expressions and that $\triangleright$ is a rewrite relation on this collection. We refer to an expression t as a $\triangleright$-*normal form* just in case there is no expression s such that $t \triangleright s$. A $\triangleright$-normal form *of* an expression r is an expression t such that $r \triangleright^* t$ and t is a $\triangleright$-normal form. A collection of rewrite rule schemata is often intended as a set of equality axioms in some logical system. Using them to rewrite expressions is useful in determining equality in the case that a unique normal form exists for every expression relative to the reduction relation defined by the rule schemata: the equality of two expressions can be checked in this situation by reducing them to their normal forms and then examining these for identity.

Certain properties of reduction relations are pertinent to determining the existence and uniqueness of normal forms. The rewrite relation $\triangleright$ is *noetherian* if and only if every sequence of rewritings relative to $\triangleright$ terminates. If $\triangleright$ is noetherian, a $\triangleright$-normal form must exist for every expression. The relation $\triangleright$ is said to be *confluent* if, given any expressions t, s_1, and s_2 such that $t \triangleright^* s_1$ and $t \triangleright^* s_2$, there must be some expression r such that $s_1 \triangleright^* r$ and $s_2 \triangleright^* r$. The following proposition, whose proof is straightforward, explains the interest in this property:

Proposition 1 *If $\triangleright$ is a confluent reduction relation, then an expression has at most one $\triangleright$-normal form.*

2.2 The de Bruijn notation for lambda terms

Definition 1 *The collection of* de Bruijn terms*, denoted by the syntactic category $\langle DTerm \rangle$, is given by the rule*

$$\begin{aligned}\langle DTerm \rangle \ ::= \ & \langle Cons \rangle \mid \#\langle Index \rangle \mid \\ & (\langle DTerm \rangle \ \langle DTerm \rangle) \mid (\lambda \, \langle DTerm \rangle)\end{aligned}$$

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 (1) $\#i$ is referred to as an index or a variable reference, (2) $(\lambda \, t)$ is called an abstraction and is said to have t as its scope, and (3) $(t_1 \ t_2)$ 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.

Lambda terms in the notation described, for instance, in [HS86] can be translated into de Bruijn terms by the following process. Let the *level* of a subterm in a term be the number of abstractions in the term within which the subterm is embedded. Further, assume that there is a fixed listing of variable names with respect to which we can talk of the n-th variable name. Then, an occurrence of the k-th variable name at level j is transformed into the index $\#(j-i)$ if it is bound by an abstraction at level i and into $\#(i+k)$ if it is free. It is easily seen that, under the translation just outlined, lambda terms that are α-convertible in the conventional notation correspond to the *same* de Bruijn term. Thus, the de Bruijn notation obviates α-conversion in the comparison of terms.

Definition 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. *$S((\lambda\, t); s_1, s_2, s_3, \ldots) = (\lambda\, S(t; \#1, s'_1, s'_2, s'_3, \ldots))$, assuming that, 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.

The main complexity in the above definition is in the last case. Towards understanding this case, we note that within a term of the form $(\lambda\, t)$, the first free variable is, in fact, denoted by the index #2, the second by #3, and so on. This requires that the indices for free variables in 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.

Definition 3 *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) defined by this rule schema is denoted by $\triangleright_\beta$ and is called β-contraction. The corresponding reduction relation is also referred to as β-reduction.

The following proposition is an easy consequence of the discussions in [NW98].

Proposition 2 *Let $t_0, t_1, t_2, \ldots$ be de Bruijn terms.*

1. *If $l \rightarrow r$ is a β-contraction rule, then*

$$S(l; t_1, t_2, t_3, \ldots) \rightarrow S(r; t_1, t_2, t_3, \ldots)$$

   is also a β-contraction rule.

2. *If $t_i \triangleright_\beta^* t_i'$ for $i \geq 0$, then $S(t_0; t_1, t_2, t_3, \ldots) \triangleright_\beta^* S(t_0'; t_1', t_2', t_3', \ldots)$.*

The following proposition is proved (for de Bruijn terms) in [Bru72].

Proposition 3 *The relation $\triangleright_\beta$ is confluent.*

Propositions 1 and 3 provide the basis for a procedure for comparing de Bruijn terms in a situation where two terms are considered to be equal if one β-contracts to the other: we attempt to reduce the two terms to a $\triangleright_\beta$-normal form and, if we succeed in doing so, we compare the resulting terms for identity. The outlined procedure becomes a decision method in the case that $\triangleright_\beta$-normal forms exist for the given terms. Although this property does not hold for arbitrary de Bruijn terms, it usually holds for subcollections of these terms that are used for representational purposes. It holds, for instance, for the class of (de Bruijn) terms restricted by a typing scheme based on simple types [And71, Chu40] that underlie the λProlog language.

The notion of a head normal form provides a means for interleaving the reduction to $\triangleright_\beta$-normal form with the checking for identity in determining the equality of terms.

Definition 4 *A head normal form relative to $\triangleright_\beta$ or a $\triangleright_\beta$-hnf is a de Bruijn term of the form $(\lambda \ldots (\lambda\, (\ldots (h\ t_1)\ \ldots\ t_n)) \ldots)$, where h is either a constant or a variable reference. The number of abstractions at the front of such a term is referred to as its binder length, h is called its head, and the terms*

$t_1, \ldots, t_n$ are said to be its arguments; in particular instances, there may be no arguments and the binder length may be 0, i.e.*, the binder may be empty. A de Bruijn term is said to be a weak $\triangleright_\beta$-hnf if it is a $\triangleright_\beta$-hnf or if it is of the form $(\lambda\, t)$. We say t is a $\triangleright_\beta$-hnf of a de Bruijn term s if t is a $\triangleright_\beta$-hnf and $s \triangleright_\beta^* t$.*

It is easily seen that two $\triangleright_\beta$-hnfs are equal in the sense being considered if and only if their binder lengths and heads are identical, they each have the same number of arguments and their arguments, taken pairwise, are equal. Further, a de Bruijn term has a $\triangleright_\beta$-normal form only if it has a $\triangleright_\beta$-hnf. Thus, in determining if two de Bruijn terms are equal, we may proceed by first reducing them to $\triangleright_\beta$-hnfs, then checking the binder lengths, the heads, and the number of arguments of these for identity and, finally, comparing the arguments of the $\triangleright_\beta$-hnfs if this is still relevant.

A procedure for reducing de Bruijn terms to $\triangleright_\beta$-hnfs is, thus, of interest. There are certain kinds of reduction sequences that are guaranteed to produce such a form from a given term whenever one exists. The following definition identifies a sequence of this kind.

Definition 5 *The head $\triangleright_\beta$-redex of a de Bruijn term that is not a $\triangleright_\beta$-hnf is identified as follows: If t is a $\triangleright_\beta$-redex, then it is its own head $\triangleright_\beta$-redex. Otherwise t must be of the form $(t_1\ t_2)$ or $(\lambda\, t_1)$. In either case, the head $\triangleright_\beta$-redex of t is identical to that of t_1. The weak head $\triangleright_\beta$-redex of a de Bruijn term that is not a weak $\triangleright_\beta$-hnf is defined similarly, except that the term in question cannot be an abstraction. The head $\triangleright_\beta$-reduction sequence of a de Bruijn term r is the sequence $r = r_0, r_1, r_2, \ldots, r_n, \ldots$, where, for $i \geq 0$, there is a de Bruijn term succeeding r_i if r_i is not a $\triangleright_\beta$-hnf and, in this case, r_{i+1} is obtained from r_i by rewriting the head $\triangleright_\beta$-redex using a β-contraction rule. Such a sequence is obviously unique and terminates just in case there is an $m \geq 0$ such that r_m is a $\triangleright_\beta$-hnf.*

The following proposition is proved for lambda terms in the conventional notation in, for instance, [Bar81], and this proof can be readily adapted to de Bruijn terms.

Proposition 4 *A de Bruijn term t has a $\triangleright_\beta$-hnf if and only if the head $\triangleright_\beta$-reduction sequence of t terminates.*

3 The suspension notation

Head $\triangleright_\beta$-reduction sequences permit β-contractions to be performed lazily in determining the equality of terms. There are practical benefits to permitting a laziness in the substitution operation inherent in β-contraction as well. For example, consider the task of determining whether the two de Bruijn terms

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

are equal modulo the β-contraction rule; s and t denote arbitrary de Bruijn terms here. It can be seen that they are not, simply by observing that these terms have as $\triangleright_\beta$-hnfs the terms $(\lambda\,(\lambda\,(\#2\ s')))$ and $(\lambda\,(\lambda\,(\#1\ t')))$, respectively, where s' and t' result from s and t by appropriate substitutions. This conclusion can, in fact, be reached *without* explicitly carrying out the potentially costly operation of substitution on the arguments. Along a different direction, laziness in substitution can be utilized to combine structure traversals needed in β-reduction, leading to gains in efficiency [AP81].

The suspension notation for lambda terms [NW98] refines the de Bruijn notation so as to support laziness in substitution. We summarize the important aspects of this notation in this section, thereby providing a background to later discussions concerning its enhancement and use.

3.1 Informal description of the notation

We provide first an intuition into the structure of the suspension notation towards making the technical definitions that follow more accessible. In essence, this notation enhances de Bruijn terms by including one new category of terms: those that encode terms with a pending substitution. The mechanism used for this purpose is similar in spirit to that of a *closure* that is employed in implementations of functional programming languages. However, the suspension notation reflects this mechanism into the syntax of terms and also generalizes the notion of an environment to permit the propagation of substitutions and the rewriting of β-redexes *inside* abstractions.

The precise structure of terms in the new category is based on the information that is to be recorded in them. In the simplest case, such terms are to encode 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 in question has the shape $((\lambda \ldots (\lambda \ldots (\lambda \ldots t \ldots) \ldots)\ s)$. Rewriting this term produces one of

the form $(\ldots(\lambda\ldots(\lambda\ldots t'\ldots)\ldots)\ldots)$. The objective is to represent the term t' that appears in the latter expression as the term t together with the substitutions that are to be performed on it.

The variable references within t can, from the perspective of interest, be factored into two groups: those corresponding to variables that are free relative to the given β-redex and those corresponding to variables bound by one of the abstractions occurring within the β-redex. Let us refer to the number of abstractions enclosing a term in a given context as its *embedding level* in that context. Thus, the embedding level of t relative to the β-redex depicted above is 3. Rewriting the β-redex eliminates an abstraction and hence changes the embedding level of t to 2. Let us refer to the embedding levels before and after the rewriting step as the *old* and *new* embedding levels and 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 $i > ol$. 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, the variable references in this group and the substitutions to be made for them are determined simply by recording the old and new embedding levels with t.

Substitutions for variable references in the second group are recorded explicitly in an environment. At a concrete level, the term t' in the situation considered is represented by an expression of the form $[\![t, ol, nl, e]\!]$, where e is a list of length ol of substitutions, presented in reverse order to the (old) embedding levels of the abstractions binding the relevant variable references. Such an expression is called a *suspension*. The elements of the environment in a suspension pertain, in general, 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 is maintained in either case is the new embedding level at the relevant abstraction, this being interpreted as that just within its scope in the case of an abstraction that persists. This quantity is called the *index* of the corresponding environment element. Certain "consistency" properties hold over the list of indices of environment elements: they form a non-increasing sequence and none of them is 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 is retained, the environment element taking the form $@l$ where $l+1$ is the value of the index. The new value of a variable reference bound by this abstraction is then given simply by $\#(nl-l)$. The environment element for an abstraction that disappears due to a β-contraction has, on the other hand, 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 the free variables in s will have to be renumbered prior to this replacement. The needed renumbering is, in fact, completely captured in the expression $[\![s, 0, (nl-l), nil]\!]$, in which nil represents the empty environment.

The addition of suspensions to de Bruijn terms permits β-contraction to be realized through atomic steps that generate and propagate substitutions over terms. These steps, whose content should already be intuitively clear, are presented in detail in Section 3.3. In the course of using these steps, it is possible that two suspensions "meet" each other. For example, consider the term $((\lambda\,((\lambda\,t_1)\;t_2))\;t_3)$. We might rewrite the two β-redexes in this term to produce the expression $[\![[\![t_1, 1, 0, (t_2, 0) :: nil]\!], 1, 0, (t_3, 0) :: nil]\!]$. An expression of this kind can be transformed into a de Bruijn term by first reducing the inner suspension to such a form and then repeating this process with the outer one. However, following this course could lead to repeated walks over the structure of the term being substituted into. Thus, in the example considered, two walks would be made over the structure of t_1, one for substituting in each of t_2 and t_3.

Towards supporting the combination of substitution walks, the suspension notation provides 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 $[\![t, ol', nl', e']\!]$. In understanding the shape of the new term, it is useful to note 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 scanned in the second walk are 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 clear that the two 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. Accordingly, ol' should be the appropriate one of these values. Similarly, 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, ol_2 is identical to the length of e_2 and the indices of the elements of e_1 are bounded by nl_1. Clearly e' has a length greater than ol_1 only if $ol_2 > nl_1$ and, in this case, 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 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 write as $\langle\!\langle et, nl_1, ol_2, e_2\rangle\!\rangle$. Let the difference between nl_1 and the index of et be h. Now, 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 when $h < ol_2$. 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 generating the inner suspension and the substitution for this abstraction in the merged environment must be the one contained for it in e_2. However, the index of this element of e_2 will have to be "normalized" if the merged environment represents substitutions for a longer sequence of abstractions than does the outer abstraction: in particular, if $nl_1 > ol_2$, then the index 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'. Then the "term" component of the relevant element in the merged environment is t

modified by the substitutions in e_h and is given precisely by the expression $[\![t, ol_2 - h, l', e_h]\!]$, and the index of this element is l', normalized as before if $nl_1 > ol_2$.

The description above assumes that the final forms of expressions are calculated in one step. Such a viewpoint runs counter to the overarching goal of providing a fine-grained control over β-reduction and substitution. The actual suspension notation corrects this situation by permitting the various computations to be broken up into a sequence of genuinely atomic steps.

3.2 The syntax of suspension expressions

We now make precise the syntax of the various expressions discussed informally above.

Definition 6 *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:*

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

We assume that $\langle Cons\rangle$ and $\langle Index\rangle$ are as in Definition 1 and that $\langle Nat\rangle$ is the category of natural numbers. We refer to suspension terms, environments, and environment terms 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.

Definition 7 *The immediate subexpression(s) of a suspension expression x are given as follows: (1) If x is a suspension term, then if (i) x is $(t_1\ t_2)$, these are t_1 and t_2, (ii) if x is $(\lambda\, t)$, this is t, and (iii) if x is $[\![t, ol, nl, e]\!]$, these are t and e. (2) If x an environment, then (i) if x is $et :: e$, these are et and e, and (ii) if x is $\{\!\{e_1, i, j, e_2\}\!\}$, these are e_1 and e_2. (3) If x is an environment term, then (i) if x is (t, l), then this is t, and (ii) if x is $\langle\!\langle et, i, j, e\rangle\!\rangle$, then these are et and e. The subexpressions of a suspension*

expression are the expression itself and the subexpressions of its immediate subexpressions. Subexpressions that are suspension terms are also referred to as subterms. A proper subexpression is any subexpression that is distinct from the given suspension expression.

The syntax of environments and environment terms includes complex forms that are used in merging suspensions. Part of the objective in this paper is to eliminate these forms while still permitting substitutions to be combined. The simpler syntax for suspension expressions is identified below.

Definition 8 *A simple suspension term, environment and environment term is an expression of the appropriate category that does not have subexpressions of the form* $\langle\langle et, j, k, e\rangle\rangle$ *or* $\{\!\{e_1, j, k, e_2\}\!\}$*. 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*.*

The following definition formalizes an already encountered notion. The symbol $\dot{-}$ used here denotes the subtraction operation on natural numbers.

Definition 9 *The length of an environment* e*, denoted by* $len(e)$*, is given as follows: (1) if* e *is* nil *then* $len(e) = 0$*; (2) if* e *is* $et :: e'$ *then* $len(e) = len(e') + 1$*; and (3) 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 shall mean the index of the l-th element of the environment if there is such an element, and the quantity 0 otherwise. This notion is formalized below, together with the notion of the index of an environment term. For expressions of the form $\{\!\{e_1, i, j, e_2\}\!\}$ and $\langle\langle et, i, j, e\rangle\rangle$, this definition reflects the simple environments and environment terms to which they are intended to correspond, according to the earlier informal description.

Definition 10 *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 induction as follows:*

1. *If* et *is* $@m$*, then* $ind(et) = m + 1$*.*
2. *If* et *is* (t', m)*, then* $ind(et) = m$*.*
3. *If* et *is* $\langle\langle et', j, k, e\rangle\rangle$*, let* $m = (j \dot{-} ind(et'))$*. Then*

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

4. *If e is nil, then $ind_l(e) = 0$.*

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

6. *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 $ind_0(e)$.

Certain constraints were noted to hold of suspension expressions when these are used as intended. These constraints are formalized as wellformedness conditions.

Definition 11 *A suspension expression is well formed if the following conditions hold of every subexpression s of the expression: (1) if s is of the form $[\![t, ol, nl, e]\!]$, then $len(e) = ol$ and $ind(e) \leq nl$, (2) if s is of the form $et :: e$, then $ind(e) \leq ind(et)$, (3) if s is of the form $\langle\!\langle et, j, k, e\rangle\!\rangle$, then $len(e) = k$ and $ind(et) \leq j$, and (4) if s is of the form $\{\!\{e_1, j, k, e_2\}\!\}$, then $len(e_2) = k$ and $ind(e_1) \leq j$.*

The qualification of wellformedness is henceforth implicitly assumed of suspension expressions.

3.3 Reduction relations on suspension expressions

Three categories of rewrite rules on suspension expressions, called the β_s-*contraction* rules, the *reading* rules and the *merging* rules, are presented through the schemata in Figures 1, 2, and 3, respectively. The β_s-contraction rules generate suspended substitutions corresponding to β-contractions, the reading rules propagate these substitutions, and the merging rules facilitate the combination of substitutions. 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

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

Figure 1: The β_s-contraction rule schema

(r1)	$[\![c, ol, nl, e]\!] \rightarrow c$, provided c is a constant.
(r2)	$[\![\#i, 0, nl, nil]\!] \rightarrow \#(i + nl)$.
(r3)	$[\![\#1, ol, nl, @l :: e]\!] \rightarrow \#(nl - l)$.
(r4)	$[\![\#1, ol, nl, (t, l) :: e]\!] \rightarrow [\![t, 0, (nl - l), nil]\!]$.
(r5)	$[\![\#i, ol, nl, et :: e]\!] \rightarrow [\![\#(i - 1), (ol - 1), nl, e]\!]$, provided $i > 1$.
(r6)	$[\![(t_1\ t_2), ol, nl, e]\!] \rightarrow ([\![t_1, ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!])$.
(r7)	$[\![(\lambda\, t), ol, nl, e]\!] \rightarrow (\lambda\, [\![t, (ol + 1), (nl + 1), @nl :: e]\!])$.

Figure 2: Rule schemata for reading suspensions

suspension terms, *et* for environment terms, e for environments, i for positive numbers, and ol, nl, l, and m for natural numbers. The applicability of several of the rule schemata are dependent on "side" conditions that are presented together with them. 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. By a harmless abuse of notation, these operations are indicated by including them in the schema for the expression to be produced.

Definition 12 *The reduction relations defined by the rule schemata in Figures 1, 2, and 3 are denoted by $\rhd_{\beta_s}$, $\rhd_r$, $\rhd_m$, 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 $\rhd_{rm\beta_s}$.*

The following proposition, proved in [NW98], establishes the legitimacy of the above definitions.

(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) $\{\!\{nil, nl, ol, et :: e\}\!\} \rightarrow \{\!\{nil, (nl - 1), (ol - 1), e\}\!\}$,
provided $nl, ol \geq 1$.

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

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

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

Figure 3: Rule schemata for merging environments

Proposition 5 *Let x be a well-formed suspension 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 suspension expression.*

Some useful properties of our reduction relations are stated below. Proofs of these properties, where omitted, may be found in [NW98].

Proposition 6 *The relation $\rhd_{rm}$ is noetherian and confluent. Further, a suspension expression is in $\rhd_{rm}$-normal form just in case (1) it is a de Bruijn term, (2) it is an environment term of the form $@l$ or (t, l), where t is a de Bruijn term, or (3) it 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.*

Propositions 1 and 6 justify the following definition:

Definition 13 *The $\rhd_{rm}$-normal form of a suspension expression t is denoted by $|t|$.*

A suspension term is intended to encapsulate a de Bruijn term with a "pending" substitution. The following proposition shows that this correspondence is as expected.

Proposition 7 *Let $t = [\![t', ol, nl, e]\!]$ and $e' = |e|$. Then*

$$|t| = S(|t'|; s_1, s_2, s_3, \ldots)$$

where

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

The following ordering relation on suspension expressions will be used in inductive arguments.

Definition 14 *Let $\sqsupset$ be the following relation on suspension expressions: $x \sqsupset y$ just in case $x \rhd_{rm} y$ or y is a proper subexpression of x. The relation $\succ$ is then the transitive closure of $\sqsupset$.*

Theorem 1 *$\succ$ is a well-founded partial ordering relation on suspension expressions.*

Proof of Theorem 1 Clearly, $\succ$ is a partial ordering relation. Suppose that there is an infinite sequence of the form $x_1 \succ x_2 \succ \ldots \succ x_n \succ \ldots$. It is easy to see that there then is an infinite sequence $y_1, y_2, \ldots, y_m, \ldots$ of suspension expressions such that either y_{i+1} is a proper subexpression of y_i for all $i \geq 1$ or $y_i \triangleright_{rm} y_{i+1}$ for all $i \geq 1$. Neither is possible, and so $\succ$ must be well founded.

Proof of Theorem 1 □

A consequence of Proposition 7 is the following correspondence between the β- and β_s-contraction rule schemata:

Proposition 8 *If $l \rightarrow r$ is an instance of the β_s-contraction rule schema, then $|l| \rightarrow |r|$ is an instance of the β-contraction rule schema.*

The above proposition implies also a correspondence between $\triangleright_\beta$ and $\triangleright_{\beta_s}$:

Proposition 9 *Let t be a de Bruijn term and let $t \triangleright_\beta s$. Then there is a suspension term r such that $t \triangleright_{\beta_s} r$ and $|r| = s$.*

Proposition 9 states only part of the relationship between $\triangleright_{rm\beta_s}$ and $\triangleright_\beta$. A complete statement requires the notion of β-reduction to be extended to suspension terms.

Definition 15 *The relation on suspension expressions defined by the β-contraction rule schema is denoted by $\triangleright_{\beta'}$.*

The soundness and relative completeness of $\triangleright_{rm\beta_s}$-reduction is then stated as follows:

Proposition 10

1. *If x and y are suspension expressions such that $x \triangleright^*_{rm\beta_s} y$, then $|x| \triangleright^*_{\beta'} |y|$.*
2. *If x and y are suspension expressions in $\triangleright_{rm}$-normal form such that $x \triangleright^*_{\beta'} y$, then $x \triangleright^*_{rm\beta_s} y$.*

Proposition 10 and the confluence of $\triangleright_{\beta'}$ can be used to show that $\triangleright_{rm\beta_s}$ is confluent. We refer the reader to [NW98] for details.

4 Eliminating the merging rules

The suspension notation for lambda terms is a general one that permits β-reduction to be realized through a variety of rewriting sequences. However, the full generality of this notation may not be useful in practice and implementation considerations favor a simpler notation. In this spirit, we consider the elimination of the merging rules and the restriction of the syntax to only simple suspension expressions. We note first that the merging rules can be dispensed with if the sole purpose is to simulate $\triangleright_\beta$-reduction. This observation is a special case of the following lemma:

Lemma 1 *Let x be a simple suspension expression and let y be such that $x \triangleright^*_{rm\beta_s} y$. Then there is a suspension expression z such that $x \triangleright^*_{r\beta_s} z$ and $y \triangleright^*_{rm} z$.*

Proof of Lemma 1 We use two observations about the rules for rewriting suspension expressions. First, a simple suspension expression that is not a $\triangleright_{rm}$-normal form can always be rewritten by a reading rule. Second, rewriting a simple suspension expression by means of a reading rule or a β_s-contraction rule produces another simple expression.

Now, since $x \triangleright^*_{rm\beta_s} y$, it follows from Proposition 10, that $|x| \triangleright^*_{\beta'} |y|$. We claim that $x \triangleright^*_r |x|$ and that $|x| \triangleright^*_{r\beta_s} |y|$. These facts yield the lemma: letting $z = |y|$, we see that $x \triangleright^*_{r\beta_s} z$ and $y \triangleright^*_{rm} z$.

The first claim, that $x \triangleright^*_r |x|$, follows immediately from the fact that x is a simple suspension expression. For the second claim, we note that if r is a $\triangleright_{rm}$-normal form and $r \triangleright_{\beta'} s$, then there is some t such that $r \triangleright_{\beta_s} t$ and $s = |t|$; this can be seen by using Proposition 9 and an induction on the structure of r. Thus, using the observations about the rewrite rules, $r \triangleright^*_{r\beta_s} s$. An induction on the length of the sequence by which $|x|$ $\triangleright_{\beta'}$-reduces to $|y|$ now shows that $|x| \triangleright^*_{r\beta_s} |y|$.

Proof of Lemma 1 □

While the merging rules and the more complex forms for suspension expressions are not needed for implementing $\triangleright_\beta$-reduction, it is only through these that substitutions embodied in different environments are combined. It is possible, however, to recognize useful sequences of applications of merging rules and to devise new rules over simple suspension expressions that encode

these sequences. The use of such derived rules can have the additional advantage of reducing a sequence of rewritings into a single step. We identify two kinds of situations in which merging rules are useful below and we codify these into new rules that preserve the underlying equality relation.

The first kind of situation arises when an attempt is made to reduce a given term to a $\rhd_{rm\beta_s}$-normal (or $\rhd_\beta$-normal) form using the strategy of rewriting the outermost and leftmost redex at each stage and merging rules are employed in combining environments that are generated by multiple uses of the β_s-contraction rule schema in this process. For example, consider the term $((\lambda\,((\lambda\,((\#1\ \#2)\ \#3)))\ t_2))\ t_3)$, in which t_2 and t_3 are arbitrary de Bruijn terms. Based on the process described, the first step would be to use the β_s-contraction rule schema to rewrite the given term to

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

The reading rules will then be used repeatedly, culminating in the production of the term

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

At this stage, the β_s-contraction rule schema would be used again to produce the term

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

This term corresponds to the term $(\lambda\,((\#1\ \#2)\ \#3))$ embedded within two suspensions. The rule schema (m1) will now be employed to merge these suspensions, yielding the term

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

The term displayed above is not a simple suspension term. However, it can be reduced to the simple suspension term

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

by repeatedly using the merging rule schemata. While the production of this term calls for a deviation from the strategy of rewriting only the outermost and leftmost redex, there is benefit to doing this: subsequent "lookups" of the environment are simplified.

The sequence of rewriting steps starting from the second use of the β_s-contraction rule schema and ending in the final simple suspension term shown can be collapsed into one use of a more "powerful" β_s-contraction rule schema.

Definition 16 *Let t_1 and t_2 be schema variables for suspension terms, e for environments, and ol and nl for natural numbers. Then the β'_s-*contraction rule schema *is*

$$((\lambda\, [\![t_1, ol+1, nl+1, @nl :: e]\!])\ t_2) \rightarrow [\![t_1, ol+1, nl, (t_2, nl) :: e]\!],$$

with the proviso that the index of the environment instantiating e is less than or equal to the natural number instantiating nl.

The proviso associated with the above rule schema cannot at the moment be easily verified. However, we shall restrict our attention in Section 6 to a class of expressions all of which satisfy this condition. The β'_s-contraction rule schema will then be applicable in a genuinely atomic fashion.

The soundness of the β_s-contraction rule schema is shown through the following lemmas.

Lemma 2 *Let $t \rightarrow s$ be an instance of the β'_s-contraction rule schema. Then there are suspension terms r and u such that $t \rightarrow r$ is a β_s-contraction rule and $s \triangleright^*_{rm} u$ and $r \triangleright^*_{rm} u$.*

Proof of Lemma 2 Since $t \rightarrow s$ is an instance of the β'_s-contraction rule schema, t is a suspension term of the form $((\lambda\, [\![t_1, ol+1, nl+1, @nl :: e]\!])\ t_2)$, s is, correspondingly, of the form $[\![t_1, ol+1, nl, (t_2, nl) :: e]\!]$, and, further, e is such that $ind(e) \leq nl$. Now let r be the suspension term

$$[\![[\![t_1, ol+1, nl+1, @nl :: e]\!], 1, 0, (t_2, 0) :: nil]\!].$$

Clearly, $t \rightarrow r$ is a β_s-contraction rule. Thus, the lemma will follow if it can be shown that s and r $\triangleright^*_{rm}$-reduce to a common expression. We do this assuming that the subexpression e of s and r is a simple environment. If this is not true, then there are suspension terms s' and r' such that $s \triangleright^*_{rm} s'$ and $r \triangleright^*_{rm} r'$ that have forms similar to s and r and that also satisfy the mentioned condition. The argument we provide here can be applied to these, eventually yielding the desired conclusion with respect to s and r. Now, by virtue of rule schemata (m1), (m5), and (m8), we see that

$$r \rhd_{rm} [\![t_1, ol+1, nl, (t_2, nl) :: \{\!\{ e, nl+1, 1, (t_2, 0) :: nil \}\!\}]\!].$$

An induction on the length of e using rule schemata (m2), (m5), (m6), and (m10) and the facts that e is a simple environment and $ind(e) \leq nl$ shows that $\{\!\{ e, nl+1, 1, (t_2, 0) :: nil \}\!\} \rhd^*_{rm} e$. From these observations it follows that both r and s $\rhd^*_{rm}$-reduce to s.

Proof of Lemma 2 □

Lemma 3 *Let r result from the suspension expression s by a use of the β'_s-contraction rule schema. Then there is a suspension expression u such that $s \rhd^*_{rm\beta_s} u$ and $r \rhd^*_{rm} u$.*

Proof of Lemma 3 By induction on the structure of suspension expressions using Lemma 2.

Proof of Lemma 3 □

The following could also have been chosen for the more powerful β_s-contraction rule schema:

$$([\![(\lambda\, t_1), ol, nl, e]\!]\ t_2) \rightarrow [\![t_1, ol+1, nl, (t_2, nl) :: e]\!].$$

This rule schema is in keeping with the behavior of reduction procedures that use environments [AP81, CCM87, HM76] and also parallels the auxiliary rule described in conjunction with the $\lambda\sigma$-calculus in [ACCL91]. The rule schema that we have actually chosen is better suited to the use made of our notation in Section 7, where reduction procedures will be expected to return suspension terms that are constants, variable references, abstractions or applications at the top-level.

There is an interaction between the use of the merging rules and the sharing of work in reduction that is worth understanding. Towards this end, we examine an alternative reduction sequence for the term considered earlier, namely $((\lambda\,((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2))\ t_3)$. Suppose that, as before, the outermost β_s-redex in this term is rewritten first to produce the term $[\![((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2), 1, 0, (t_3, 0) :: nil]\!]$. At this point, the embedded $\rhd_{\beta_s}$-redex $((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2)$ may be rewritten in contrast to the choice exercised earlier, producing the term

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

Assuming that a graph-based implementation of reduction is being used, following this course has the apparent advantage that $((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2)$ is rewritten before the propagation of the substitution "breaks" a possible sharing relative to it. Notice, however, that the full benefit of such a sharing of reduction is realized only if the substitutions encoded in the two suspensions in the resulting term are performed in *separate* walks over the structure of $(\lambda\,((\#1\ \#2)\ \#3))$. Further, opportunities for a different kind of sharing in reduction can be missed if the merging (or the derivative β'_s-contraction) rules are not used. In the case being considered, t_3 has eventually to be substituted into t_2. If the merging rules are not used, then this substitution would have to be carried out separately in the different copies that might be made of t_2. In general, the trade-off between different reduction strategies is more complex than it initially appears to be. A notation such as the one considered here provides the basis for a careful analysis of this issue.

We now consider the second kind of situation in which the merging rules are useful. These situations arise when suspensions have to be substituted into particular contexts and the indices within the de Bruijn terms they represent have to be adjusted as a result. We continue with the reduction of the term $((\lambda\,((\lambda\,(\lambda\,((\#1\ \#2)\ \#3)))\ t_2))\ t_3)$ to provide an illustration. This term had been rewritten to

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

Using the reading rules repeatedly in a leftmost-outermost fashion, it can be further transformed into

$$\begin{array}{l}(\lambda\,((\#1\ [\![[\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0, 1, nil]\!]) \\ \qquad\qquad [\![\#3, 3, 1, @0 :: ([\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0) :: (t_3, 0) :: nil]\!])).\end{array}$$

The subterm $[\![[\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0, 1, nil]\!]$ of this term corresponds to t_2 embedded within two suspensions. The inner suspension represents the result of substituting t_3 for the first free variable within t_2 and arises from the use of a β_s-contraction rule. The outer suspension represents the "bumping up" of the indices for the free variables in the resulting term by 1 and is necessitated by the substitution of the term in question into an abstraction. The merging rules provide a means for combining these different substitutions and thereby performing them in the same walk over the structure of t_2. In particular, these rules can be used to rewrite $[\![[\![t_2, 1, 0, (t_3, 0) :: nil]\!], 0, 1, nil]\!]$ into $[\![t_2, 1, 1, (t_3, 0) :: nil]\!]$. Such uses of the merging rules can also be codified in a new rule schema.

Definition 17 *The (bump) rule schema is the following*

$$[\![[\![t, ol, nl, e]\!], 0, nl', nil]\!] \rightarrow [\![t, ol, nl + nl', e]\!],$$

where t is a schema variable for a suspension term, e is a schema variable for an environment and ol, nl, and nl' are schema variables for natural numbers.

The following lemma justifies the use of the above rule schema in reduction sequences.

Lemma 4 *Let r result from the suspension expression s by a use of the (bump) rule schema. Then there is an expression u such that $s \triangleright^*_{rm} u$ and $r \triangleright^*_{rm} u$.*

Proof of Lemma 4 It suffices to show the lemma assuming that s is an instance of the lefthand side of the rule schema being considered. Thus, s is a term of the form $[\![[\![t, ol, nl, e]\!], 0, nl', nil]\!]$. As in the proof of Lemma 2, we may assume that e is a simple environment. Now, using rule schema (m1), $s \triangleright^*_{rm} [\![t, ol, nl + nl', \{\!\{e, nl, 0, nil\}\!\}]\!]$.

By an induction on $len(e)$ and using rule schemata (m2), (m5), and (m6), it can be seen that $\{\!\{e, nl, 0, nil\}\!\} \triangleright^*_{rm} e$. Letting u be r, the lemma follows.

Proof of Lemma 4 □

The new rule schemata presented in this section encapsulate certain patterns of application of the merging rules that are likely to find use within a particular scheme for reducing expressions to $\triangleright^*_{rm\beta_s}$-normal form. They do not, of course, capture all potential uses of the merging rules. If subterms of a term are rewritten in arbitrary order using the β_s-contraction rules, then two environments that can be merged only by using the general rules may appear in juxtaposition. Such "mixed" orders of rewriting can occur in reasonable reduction procedures; they might occur, for instance, within an implementation that permits a sharing of subterms and, consequently, of reduction steps. It can sometimes be recognized that the substitutions in an outer suspension have no effect on an inner one, thereby obviating the merging of these suspensions. These situations are discussed in the next section. Moreover, missing some opportunities for combining substitution walks is arguably an acceptable trade-off for a simpler syntax and set of reduction rules.

5 Some rules for simplifying suspensions

It can sometimes be recognized that the substitution embodied in a suspension does not affect the term being substituted into. The suspension can, in such cases, be rewritten directly to its term component. We identify two kinds of such situations in this section. The first of these corresponds to the case where the term being substituted into does not contain variable occurrences that could be bound by abstractions external to it. The idea of closedness that is defined below approximates this property of terms.

Definition 18 *Given natural numbers i and nl, the notions of being i-closed for a suspension term and (i, nl)-closed for an environment are defined simultaneously by structural recursion as follows: A suspension term t is said to be i-closed if (1) t is a constant or of the form $\#k$, where $k \leq i$; or (2) t is of the form $(t_1\ t_2)$, where t_1 and t_2 are i-closed; or (3) t is of the form $(\lambda\, t_1)$, where t_1 is $(i+1)$-closed; or (4) t is of the form $[\![t_1, ol, nl, e]\!]$, where t_1 is $(\max(ol, i - (nl - ol)))$-closed and e is (i, nl)-closed. An environment e is said to be (i, nl)-closed if (5) it is nil, or (6) it is of the form $@j :: e'$ where $(nl - j) \leq i$ and e' is (i, nl)-closed, or (7) it is of the form $(t, j) :: e'$ and t is $(i \dot{-} (nl - j))$-closed and e' is (i, nl)-closed. A suspension term is said to be closed if it is 0-closed.*

The assessment of closedness is, by definition, restricted to simple suspension terms. Even within this context, the judgment can be conservative. It is possible, for example, to manifest a term of the form $[\![t, ol, nl, e]\!]$ that is not deemed to be closed but whose $\triangleright_{rm}$-normal form is a de Bruijn term that does not contain unbound variable references. However, an assessment of closedness is sound. This is a special case of the following lemma.

Lemma 5 *If t is an i-closed suspension term, then so is $|t|$.*

Proof of Lemma 5 Since t is i-closed, it must be a simple suspension term. As noted in the proof of Lemma 1, this implies that $t \triangleright_r^* |t|$. The lemma will therefore follow if we can show that if x is a (simple) i-closed suspension term such that $x \triangleright_r y$, then y is also i-closed.

It is actually more convenient to prove a stronger form of the required observation: (1) if x is an i-closed suspension term such that $x \triangleright_r y$, then y is also i-closed, and (2) if x is an (i, nl)-closed environment such that $x \triangleright_r y$, then y is also (i, nl)-closed. An inspection of the reading rules verifies the

truth of (1) and (2) in the case that $x \rightarrow y$ is an instance of one of these rules. A straightforward induction on the structure of x extends this to the general situation.

Proof of Lemma 5 □

We now present the simplification rules of the first kind.

Definition 19 *The (cl) rule schema is*

$$[\![t, ol, nl, e]\!] \rightarrow t$$

where t is a schema variable for closed suspension terms, e is a schema variable for environments, and ol and nl are schema variables for natural numbers.

The proviso on the use of the (cl) rule schema is difficult to verify at the moment. However, Section 6 describes an annotation system whose purpose is to identify closed suspension terms.

The following lemma justifies the use of the (cl) rule schema in reductions.

Lemma 6 *Let r result from the suspension expression s by an application of the (cl) rule schema. Then there is an expression u such that $s \rhd_{rm}^* u$ and $r \rhd_{rm}^* u$.*

Proof of Lemma 6 It suffices to show the lemma assuming that s is an instance of the lefthand side of the rule schema in question. Further, if s is of the form $[\![t, ol, nl, e]\!]$, we may assume that t is a $\rhd_{rm}$-normal form. Observing that $[\![t, ol, nl, e]\!] \rhd_{rm}^* [\![|t|, ol, nl, e]\!]$, $t \rhd_{rm}^* |t|$ and that, by Lemma 5, $|t|$ is closed if t is, the lemma is easily extended to the general case.

For the case being considered, we actually make a slightly stronger claim. Let i be a natural number. Letting $@(nl + i - 1) :: \ldots :: @nl :: e$ stand for e in the case that $i = 0$ and assuming that t is an i-closed $\rhd_{rm}$-normal form, we claim that

$$[\![t, ol + i, nl + i, @(nl + i - 1) :: \ldots :: @nl :: e]\!] \rhd_{rm}^* t.$$

The claim is proved by induction on the structure of t. It is obviously true if t is a constant. Let t be of the form $\#j$. Since t is i-closed, $j \leq i$. (Clearly, $i > 0$ in this case.) Thus

$$[\![t, ol + i, nl + i, @(nl + i - 1) :: \ldots :: @nl :: e]\!] \rhd^*_{rm} \\ \#(nl + i - (nl + i - j)) = t.$$

In the case that t is an abstraction or an application, the claim is shown by a straightforward recourse to the inductive hypothesis.

Proof of Lemma 6 □

The second kind of situation where a suspension can be simplified to its term component is one in which the substitution involved is vacuous. Such substitutions can arise in practice from attempts to renumber variable references. The following definition presents the relevant simplification rule schema.

Definition 20 *The (nulle) rule schema is the following*

$$[\![t, 0, 0, nil]\!] \rightarrow t$$

where t is a schema variable for a suspension term.

This rule schema is shown to be sound by the following lemma.

Lemma 7 *Let s be a suspension expression and let r result from s by an application of the (nulle) rule schema. Then there is a suspension expression u such that $s \rhd^*_{rm} u$ and $r \rhd^*_{rm} u$.*

Proof of Lemma 7 Let i be a natural number and let $@i :: @(i-1) :: \ldots :: @0 :: nil$ denote nil if i is 0. We claim that if t is a $\rhd_{rm}$-normal form, then $[\![t, i, i, @i :: @(i-1) :: \ldots :: @0 :: nil]\!] \rhd^*_{rm} t$. The lemma is an easy consequence of this claim. The claim itself is proved by a structural induction on t.

Proof of Lemma 7 □

Using the rule schemata (cl) and (nulle) whenever they are applicable obviously reduces the work involved in producing a $\rhd_{rm\beta_s}$-normal form. Less obvious consequences of using these schemata are the conservation of space and the possibility for a greater sharing of work in the context of a graph-based implementation of reduction. To appreciate these aspects, let us consider the reduction to $\rhd_{rm\beta_s}$-normal form of the term $[\![((\lambda\, t_1)\ t_2), ol, nl, e]\!]$, assuming that $((\lambda\, t_1)\ t_2)$ is known to be a closed term. Using the (cl) rule

schema, this term can be rewritten directly to $((\lambda\, t_1)\ t_2)$. This rewriting step preserves any sharing that might exist with respect to $((\lambda\, t_1)\ t_2)$ and, consequently, leaves unaltered the possibility of sharing the reduction work pertaining to this subterm. Additionally, no new structures are created as a result of the rewriting. In contrast, if the (cl) rule schema is not available, then the given term would be rewritten to $([\![(\lambda\, t_1), ol, nl, e]\!]\ [\![t_2, ol, nl, e]\!])$, assuming a leftmost-outermost reduction strategy. Proceeding in this fashion destroys the sharing with regard to the outermost $\rhd_{\beta_s}$-redex and thus precludes its reduction from providing a benefit in some other context. Furthermore, the propagation of the substitution eventually results in the replication of the entire structure of the term $((\lambda\, t_1)\ t_2)$. The unnecessary use of space and time in computing such substitutions has been noted to be significant in practice [BR91], and the simplification rules presented in this section are of value from this perspective.

6 A refinement to the suspension notation

We now modify the suspension notation to take into account the discussions of the last two sections. One aspect of this modification is a restriction to simple suspension expressions and the elimination of merging rules. The second change that is considered is that of including annotations in suspension expressions to indicate that certain expressions do not contain variables that could be bound by abstractions in whose scope they appear. In particular, our annotation scheme categorizes abstractions, applications, and suspensions as either closed expressions or expressions that could contain unbound variable references. These annotations permit the (cl) rule schema to be utilized in an effective manner. For these annotations to be eventually useful, the information in them must be preserved in the course of reduction. We modify the rewrite rules for (simple) suspension expressions to achieve this effect.

6.1 Annotated suspension expressions and associated reduction relations

Our refinement to suspension expressions is given by the following definition.

Definition 21 *The categories of annotated suspension terms, annotated environments, and annotated environment terms, denoted by* $\langle ATerm\rangle$, $\langle AEnv\rangle$, *and* $\langle AETerm\rangle$, *respectively, are defined by the following syntax rules:*

$$\begin{array}{lll}
\langle ATerm\rangle & ::= & \langle Cons\rangle \mid \#\langle Index\rangle \mid \\
& & (\langle ATerm\rangle\ \langle ATerm\rangle)_o \mid (\langle ATerm\rangle\ \langle ATerm\rangle)_c \mid \\
& & (\lambda_o\,\langle ATerm\rangle) \mid (\lambda_c\,\langle ATerm\rangle) \mid \\
& & [\![\langle ATerm\rangle, \langle Nat\rangle, \langle Nat\rangle, \langle AEnv\rangle]\!]_o \mid \\
& & [\![\langle ATerm\rangle, \langle Nat\rangle, \langle Nat\rangle, \langle AEnv\rangle]\!]_c \\
\langle AEnv\rangle & ::= & nil \mid \langle AETerm\rangle :: \langle AEnv\rangle \\
\langle AETerm\rangle & ::= & @\langle Nat\rangle \mid (\langle ATerm\rangle, \langle Nat\rangle)
\end{array}$$

We assume that $\langle Cons\rangle$, $\langle Index\rangle$, *and* $\langle Nat\rangle$ *correspond to the same categories of expressions as in Definition 6. Annotated suspension terms, annotated environments and annotated environment terms are referred to collectively as annotated suspension expressions.*

Annotated suspension expressions are structurally similar to simple suspension expressions. The following translation function explicates this similarity.

Definition 22 *The suspension expression underlying an annotated suspension expression* t *is denoted by* $\overline{t}$ *and is given as follows: (1) if* t *is* c, $\#i$ *or* nil, *then* $\overline{t} = t$, *(2) if* t *is* $(\lambda_o\, t_1)$ *or* $(\lambda_c\, t_1)$, *then* $\overline{t} = (\lambda\, \overline{t_1})$, *(3) if* t *is* $(t_1\ t_2)_o$ *or* $(t_1\ t_2)_c$, *then* $\overline{t} = (\overline{t_1}\ \overline{t_2})$, *(4) if* t *is* $[\![t_1, ol, nl, e]\!]_o$ *or* $[\![t_1, ol, nl, e]\!]_c$, *then* $\overline{t} = [\![\overline{t_1}, ol, nl, \overline{e}]\!]$, *(5) if* t *is* $@l :: e$, *then* $\overline{t} = @l :: \overline{e}$, *and (6) if* t *is* $(t_1, l) :: e$, *then* $\overline{t} = (\overline{t_1}, l) :: \overline{e}$.

We exploit the structural similarity to speak of (annotated) abstractions, applications and suspensions and also to clarify the notion of subexpressions. The restriction to simple expressions permits several related definitions to be simplified.

Definition 23 *The index of an annotated environment term* et, *written* $ind(et)$, *is* $(m+1)$ *if* et *is* $@m$ *and* m *if* et *is* (t, m). *The index of an environment* e, *written* $ind(e)$, *is* 0 *if* e *is* nil *and* $ind(et)$ *if* e *is of the form* $et :: e'$. *The length of an annotated environment* e, *written* $len(e)$, *is* 0 *if* e *is* nil *and* $len(e') + 1$ *if* e *is of the form* $et :: e'$. *If* $len(e) = n$, *then obviously* e *is of the form* $et_1 :: et_2 :: \ldots :: et_n :: nil$. *As before, for* $0 < i \leq n$, *we write* $e[i]$ *to denote the environment term* et_i.

The restriction in structure also allows us to place a slightly stronger condition of wellformedness on our expressions than is implicit from the translation. The new requirement ensures that the proviso on (a suitably adapted version of) the β'_s-contraction rule schema will be satisfied by all relevant annotated terms.

Definition 24 *An annotated suspension expression is well formed if the following is true of all its subexpressions: If this is of the form* $@l :: e$ *or* $(t, l) :: e$*, then* $ind(e) \leq l$*. If this is of the form* $[\![t, ol, nl, e]\!]_o$ *or* $[\![t, ol, nl, e]\!]_c$*, then* $len(e) = ol$ *and* $ind(e) \leq nl$*.*

The following lemma is obvious.

Lemma 8 *If* t *is a well-formed annotated suspension expression, then* $\overline{t}$ *is a well-formed simple suspension expression.*

Annotations on suspension expressions are intended to indicate whether or not they may be considered closed in the sense of Section 5. A requirement of consistency can therefore be placed on these annotations. Such a requirement is spelled out in the following definition.

Definition 25 *An annotated suspension term* t *is said to be c-annotated, or closed annotated, if it is of the form* $(\lambda_c\, t_1)$*,* $(t_1\ t_2)_c$*, or* $[\![t_1, ol, nl, e]\!]_c$*. An annotated suspension expression* t *is said to be consistently annotated if the following conditions hold for each c-annotated subterm* s *of* t*: (1)* $\overline{s}$ *is a closed term, (2) if* $s = (s_1\ s_2)_c$*, then, for* $i = 1$ *and* $i = 2$*,* s_i *either is a constant or is c-annotated, and (3) if* $s = [\![s_1, ol, nl, e]\!]_c$*, then* s_1 *either is a constant or is c-annotated if* $ol = 0$*, and otherwise this property holds for every* s' *such that* $e[i] = (s', l)$ *for some* i *between* 1 *and* ol*.*

Figures 4 and 5 present rule schemata for rewriting annotated suspension expressions. The interpretation of these schemata is similar to those for rewriting suspension expressions with the difference that the tokens t, et, and e, used, perhaps, with subscripts or superscripts, are now schema variables for *annotated* suspension expressions of the relevant categories and that new "annotation" schema variables u and v appear in these rules that may be replaced by either o or c. The choice of rule schemata reflects the discussions in the preceding two sections. Note, in particular, that the schema (aβ'_s) corresponds to the (β'_s) schema; (ar8), (ar9), and (ar10) correspond to (cl); (ar11) corresponds to (bump); and (ar12) corresponds to (nulle). An assumption

(aβ_s)	$((\lambda_u\, t_1)\ t_2)_v \rightarrow [\![t_1, 1, 0, (t_2, 0) :: nil]\!]_v$
(aβ'_s)	$((\lambda_u\, [\![t_1, ol+1, nl+1, @nl :: e]\!]_o)\ t_2)_v \rightarrow [\![t_1, ol+1, nl, (t_2, nl) :: e]\!]_v$

Figure 4: The β_s-contraction rule schemata for annotated suspension expressions

(ar1)	$[\![c, ol, nl, e]\!]_u \rightarrow c$, provided c is a constant.
(ar2)	$[\![\#i, 0, nl, nil]\!]_u \rightarrow \#(i + nl)$.
(ar3)	$[\![\#1, ol, nl, @l :: e]\!]_u \rightarrow \#(nl - l)$.
(ar4)	$[\![\#1, ol, nl, (t, l) :: e]\!]_u \rightarrow [\![t, 0, nl - l, nil]\!]_u$.
(ar5)	$[\![\#i, ol, nl, et :: e]\!]_u \rightarrow [\![\#(i-1), ol - 1, nl, e]\!]_u$, provided $i > 1$.
(ar6)	$[\![(t_1\ t_2)_u, ol, nl, e]\!]_v \rightarrow ([\![t_1, ol, nl, e]\!]_v\ [\![t_2, ol, nl, e]\!]_v)_v$.
(ar7)	$[\![(\lambda_u\, t), ol, nl, e]\!]_v \rightarrow (\lambda_v\, [\![t, ol+1, nl+1, @nl :: e]\!]_o)$.
(ar8)	$[\![(t_1\ t_2)_c, ol, nl, e]\!]_u \rightarrow (t_1\ t_2)_c$.
(ar9)	$[\![(\lambda_c\, t), ol, nl, e]\!]_u \rightarrow (\lambda_c\, t)$.
(ar10)	$[\![[\![t, ol, nl, e]\!]_c, ol', nl', e']\!]_u \rightarrow [\![t, ol, nl, e]\!]_c$.
(ar11)	$[\![[\![t, ol, nl, e]\!]_o, 0, nl', nil]\!]_o \rightarrow [\![t, ol, nl + nl', e]\!]_o$.
(ar12)	$[\![t, 0, 0, nil]\!]_u \rightarrow t$.

Figure 5: Rule schemata for reading annotated suspensions

of consistency of annotations is important for (ar8), (ar9), and (ar10) to be sound implementations of (cl). We envisage that terms are annotated in a consistent fashion at the outset. The rules then manipulate annotations in a way that preserves their consistency, a fact we observe in Theorem 2.

Definition 26 *The reduction relations on annotated suspension expressions that are defined by the rule schemata in Figures 4 and 5 are denoted by $\rhd_{a\beta_s}$ and $\rhd_{ar}$, respectively. The union of these two relations is denoted by $\rhd_{ar\beta_s}$.*

Theorem 2 *Let t be a well-formed annotated suspension expression that is consistently annotated. If $t \rhd_{ar} s$ or $t \rhd_{a\beta_s} s$, then s is also well formed and consistently annotated.*

Proof of Theorem 2 *Wellformedness.* Instances of the lefthand sides of the rule schemata in Figures 4 and 5 can only be annotated suspension terms. From this and an inspection of Definition 24 it follows easily that s would be well formed if whenever an instance of the lefthand side of one of the relevant rule schemata is well formed, then the corresponding instance of the righthand side is also well formed. An inspection of the rule schemata verifies that this is the case.

Consistency of annotations. Let l be the subterm of t that is replaced by one of the rule schemata in obtaining s, and let r be the term that replaces l. By assumption, l is consistently annotated. We claim that (a) if $\bar{l}$ is i-closed (for any i), then $\bar{r}$ is i-closed, and (b) r is consistently annotated. If these conditions hold, s must be consistently annotated: the subexpressions of t that are not affected by the replacement continue to be consistently annotated, and from (a) and (b) it follows easily that subexpressions that are affected are changed only in ways that preserve the consistency of annotations.

Claim (a) is established by considering in turn the various possibilities for the rule schema that is used. If this is (aβ_s), (ar12) or one of (ar1)–(ar6), the argument is routine. In the case that the rule schema is one of (ar8)–(ar10), we need the easily verified observation that if a term t is i-closed for some i, then it is j-closed for any $j > i$. The consistency of annotation of the lefthand side ensures that $\bar{r}$ is 0-closed in each of the relevant cases and hence it must be i-closed for any i. For the cases of (aβ'_s) and (ar7), we need an additional easily confirmed fact: if e is an environment such that $ind(e) \leq nl$, then e is $(i+1, nl+1)$-closed if and only if e is (i, nl)-closed. The remainder of the argument in these cases is straightforward.

The only rule schema left to be considered is (ar11). In this case, l and r are of the form $[\![[\![t, ol, nl, e]\!]_o, 0, nl', nil]\!]_o$ and $[\![t, ol, nl + nl', e]\!]_o$, respectively. Since $\bar{l}$ is i-closed and $\max(0, i - nl') = (i \dot{-} nl')$, it follows that $\bar{t}$ is $\max(ol, (i \dot{-} nl') - (nl - ol))$-closed and that $\bar{e}$ is $(i \dot{-} nl', nl)$-closed. From the wellformedness of $[\![t, ol, nl, e]\!]_o$ (and the consequent wellformedness of $[\![\bar{t}, ol, nl, \bar{e}]\!]$) it follows that $ind(\bar{e}) \leq nl$. Now, it is easily seen that if e' is a simple environment such that $ind(e') \leq n$ and e' is $(j \dot{-} k, n)$-closed, then e' is $(j, k+n)$-closed. Thus, it must be the case that $\bar{e}$ is $(i, nl + nl')$-closed. We further note that $ol \geq 0$, and, therefore,

$$\max(ol, (i \dot{-} nl') - (nl - ol)) = \max(ol, i - (nl + nl' - ol)).$$

But then t is $\max(ol, i - (nl + nl' - ol))$-closed. It is now easily seen that $[\![\overline{t}, ol, nl + nl', \overline{e}]\!]$, *i.e.*, $\overline{r}$, is i-closed.

Claim (b) follows from a routine inspection of the rule schemata using claim (a); the only case that requires some consideration is that of rule schema (ar6), but even here the argument is simple.

Proof of Theorem 2 □

We assume hereafter that all the annotated suspension expressions we deal with are well formed and consistently annotated.

6.2 Simulation of β-reduction

Conceptually, annotations are to be used in the following fashion: given a de Bruijn term that is to be reduced to a normal form, annotations are introduced into this term, reduction on annotated suspension expressions is used to simulate β-reduction, and eventually annotations are removed. This "procedure" is not intended literally — in particular, annotations need not be removed from terms at the end of a reduction sequence to get the necessary information — but, rather, as a basis for determining the correctness of the reduction relations on annotated suspension expressions. We observe a series of properties of these relations below, eventually verifying their soundness and completeness from this perspective in Theorem 5.

The reading rules have the purpose in the framework described of transforming arbitrary annotated suspension terms into annotated versions of de Bruijn terms. We see that they actually do this in the following lemma.

Theorem 3 *The relation $\rhd_{ar}$ is noetherian. Further, an annotated suspension expression is a $\rhd_{ar}$-normal form if and only if it does not contain any subexpressions of the form $[\![t, ol, nl, e]\!]_c$ or $[\![t, ol, nl, e]\!]_o$. In particular, an annotated suspension term x is a $\rhd_{ar}$-normal form if and only if $\overline{x}$ is a de Bruijn term.*

Proof of Theorem 3 An inspection of the relevant rules shows that if $t \rhd_{ar} s$ then $\overline{t} \succ \overline{s}$. The noetherianity of $\rhd_{ar}$ then follows from the wellfoundedness of $\succ$. An annotated suspension expression that has a well-formed subpart of the form $[\![t, ol, nl, e]\!]_c$ or $[\![t, ol, nl, e]\!]_o$ can be rewritten by one of the rules in

Figure 5. Such an expression cannot, therefore, be a $\rhd_{ar}$-normal form. The last part of the theorem is obvious from Definition 22.

Proof of Theorem 3 □

Although a $\rhd_{ar}$-normal form exists for every annotated suspension expression, this form is not necessarily unique. The reason for this is that the use of certain rules in opposition to others results in more "conservative" annotations. Thus, consider a term of the form $[\![(\lambda_c \#1), ol, nl, e]\!]_o$. Depending on whether rule schema (ar7) or (ar9) is used in rewriting this term, the $\rhd_{ar}$-normal form that is produced with be either $(\lambda_o \#1)$ or $(\lambda_c \#1)$. However, the reading rules still induce a satisfactory reduction relation from our perspective because the $\rhd_{ar}$-normal forms of a given expression differ at most in their annotations. We show that this is the case in Theorem 4.

Lemma 9 *Let t and s be annotated suspension expressions. If $t \rhd_{ar} s$, then there is a suspension expression u such that $\overline{t} \rhd^*_{rm} u$ and $\overline{s} \rhd^*_{rm} u$. If $t \rhd_{ar\beta_s} s$, then there is a suspension expression u such that $\overline{t} \rhd^*_{rm\beta_s} u$ and $\overline{s} \rhd^*_{rm} u$.*

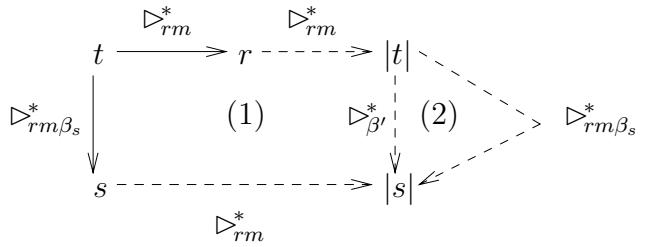
Proof of Lemma 9 It suffices to show the lemma assuming that $t \rightarrow s$ is an instance of one of the rule schemata in Figures 4 and 5. The argument is obvious for all rule schemata other than (aβ'_s) and (ar8)–(ar12). In the case of (aβ'_s), the desired conclusion follows from Lemma 3; as already noted, the wellformedness condition on annotated suspension expressions ensures that $\overline{t}$ satisfies the proviso on the use of the (β'_s)-contraction rule schema. The arguments for (ar11) and (ar12) use Lemma 4 and Lemma 7, respectively. Finally, for the cases of (ar8)–(ar10), use is made of Lemma 6, noting that the consistency of annotations guarantees that $\overline{t}$ satisfies the proviso on the use of the (cl) rule schema in each of these cases.

Proof of Lemma 9 □

Lemma 9 is a soundness observation for single uses of the rule schemata in Figures 4 and 5. The following permutability property of reductions is needed in extending this observation to sequences of applications of these schemata. In diagrams such as the one used in this proof, the dashed arrows are to be interpreted as assertions of the existence of reductions given by the labels on them, depending on the reductions depicted by the solid arrows.

Lemma 10 *Let t be a suspension expression, let $t \rhd^*_{rm\beta_s} s$, and let $t \rhd^*_{rm} r$. Then there is an expression u such that $r \rhd^*_{rm\beta_s} u$ and $s \rhd^*_{rm} u$.*

Proof of Lemma 10 Let $u = |s|$. Then, the following diagram verifies the lemma:

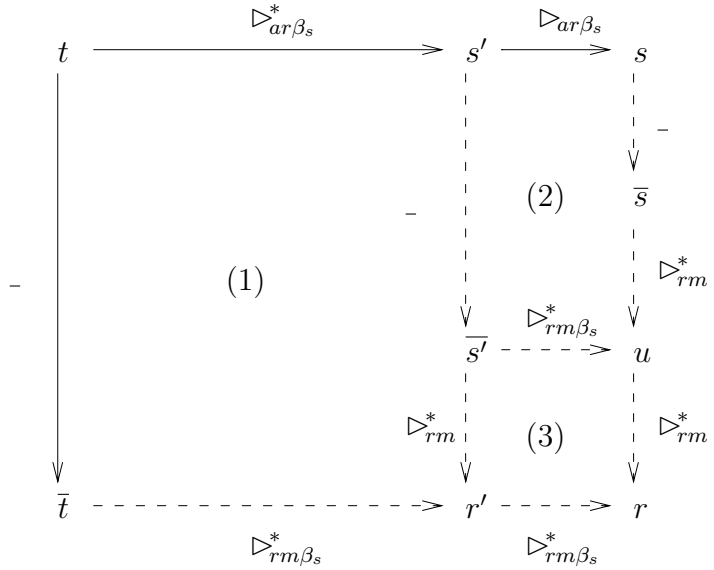


Here, the dashed arrow from r to $|t|$ is justified by Proposition 6, the dashed arrow from $|t|$ to $|s|$ in face (1) is justified by Proposition 10, and the remaining dashed arrow in face (2) is also justified by the same proposition.

Proof of Lemma 10 □

Lemma 11 *Let t be an annotated suspension expression. If $t \triangleright^*_{ar} s$, then there is a suspension expression r such that $\overline{t} \triangleright^*_{rm} r$ and $\overline{s} \triangleright^*_{rm} r$. If $t \triangleright^*_{ar\beta_s} s$, then there is a suspension expression r such that $\overline{t} \triangleright^*_{rm\beta_s} r$ and $\overline{s} \triangleright^*_{rm} r$.*

Proof of Lemma 11 The arguments in both cases are similar and so we consider explicitly only the situation where $t \triangleright^*_{ar\beta_s} s$. We induce on the length of the reduction sequence, the details being represented in the following diagram:



The arrows labeled with - in this diagram correspond to the use of Definition 22. The rectangle labeled (1) in this diagram is completed by using the hypothesis, the rectangle labeled (2) is completed by invoking Lemma 9, and the rectangle labeled (3) is justified by Lemma 10.

Proof of Lemma 11 □

Theorem 4 *Let t be an annotated suspension expression and let s and r be $\rhd_{ar}$-normal forms of t. Then $\overline{s} = \overline{r}$. Further, $|\overline{t}| = \overline{s}$.*

Proof of Theorem 4 From Proposition 6, Definition 22, Theorem 3, and Lemma 11, it follows that s and r are $\rhd_{ar}$-normal forms of t only if $\overline{s}$ and $\overline{r}$ are $\rhd_{rm}$-normal forms of $\overline{t}$. Thus, by Propositions 1 and 6, $\overline{s} = \overline{r} = |\overline{t}|$.

Proof of Theorem 4 □

Theorem 4 justifies the following definition.

Definition 27 *Let t be an annotated suspension expression and let s be a $\rhd_{ar}$-normal form of t. Then $\overline{|t|}_a$ denotes $\overline{s}$.*

We turn now to an explicit consideration of the β_s-contraction rules for annotated suspension expressions. We note first a correspondence between these rules and the β-contraction rules for de Bruijn terms.

Lemma 12 *Let $l \rightarrow r$ be an instance of one of the β_s-contraction rule schemata for annotated suspension expressions. Then $\overline{|l|}_a \rightarrow \overline{|r|}_a$ is an instance of the β-contraction rule schema.*

Proof of Lemma 12 As already observed, $\overline{l} \rightarrow \overline{r}$ is an instance of either the β_s- or the β'_s-contraction rule schema. By Proposition 8 and Lemma 2 it then follows that $|\overline{l}| \rightarrow |\overline{r}|$ is an instance of the β-contraction rule schema. By Theorem 4, $|\overline{l}| = \overline{|l|}_a$ and $|\overline{r}| = \overline{|r|}_a$.

Proof of Lemma 12 □

In Lemma 11 we have seen that any $\rhd_{ar\beta_s}$-reduction sequence can, in a suitable sense, be simulated by a $\rhd_{rm\beta_s}$-reduction sequence. We now observe the converse property.

Lemma 13 *Let t be an annotated suspension expression and let $\overline{t} \rhd^*_{rm\beta_s} s$. Then there is an r such that $t \rhd^*_{ar\beta_s} r$ and $s \rhd^*_{rm} \overline{r}$.*

Proof of Lemma 13 Clearly $\overline{t}$ is a simple suspension expression. By Lemma 1 there must therefore be a u such that $\overline{t}\rhd^*_{r\beta_s}u$ and $s\rhd^*_{rm}u$. Now, corresponding to each schema in Figure 1 and Figure 2, there is a rule schema for rewriting annotated suspension expressions that produces a similar effect. An induction on the length of $\rhd_{r\beta_s}$-reduction sequences using this fact allows us to conclude, as required, that there is some r such that $t\rhd^*_{ar\beta_s}r$ and $\overline{r} = u$.

Proof of Lemma 13 □

The correctness observation for $\rhd_{ar\beta_s}$-reductions desired at the outset is essentially the restriction of the following theorem to (annotated) de Bruijn terms.

Theorem 5 *1. Let x and y be annotated suspension expressions such that $x\rhd^*_{ar\beta_s}y$. Then $\overline{|x|}_a\rhd^*_{\beta'}\overline{|y|}_a$.*

2. Let x and y be suspension expressions in $\rhd_{rm}$-normal form such that $x\rhd^*_{\beta'}y$. If x' is an annotated suspension expression such that $\overline{x'} = x$, then there is an annotated suspension expression z such that $x'\rhd^*_{ar\beta_s}z$ and $\overline{z} = y$.

Proof of Theorem 5 *(1)* By Lemma 11, there is a suspension expression w such that $\overline{x}\rhd^*_{rm\beta_s}w$ and $\overline{y}\rhd^*_{rm}w$. But then it must be the case that $\overline{x}\rhd^*_{rm\beta_s}|\overline{y}|$. By Proposition 10, it must be the case that $|\overline{x}|\rhd^*_{\beta'}||\overline{y}||$. The desired conclusion now follows from noting that $\overline{|x|}_a = |\overline{x}|$, $\overline{|y|}_a = |\overline{y}|$, and $||\overline{y}|| = |\overline{y}|$.

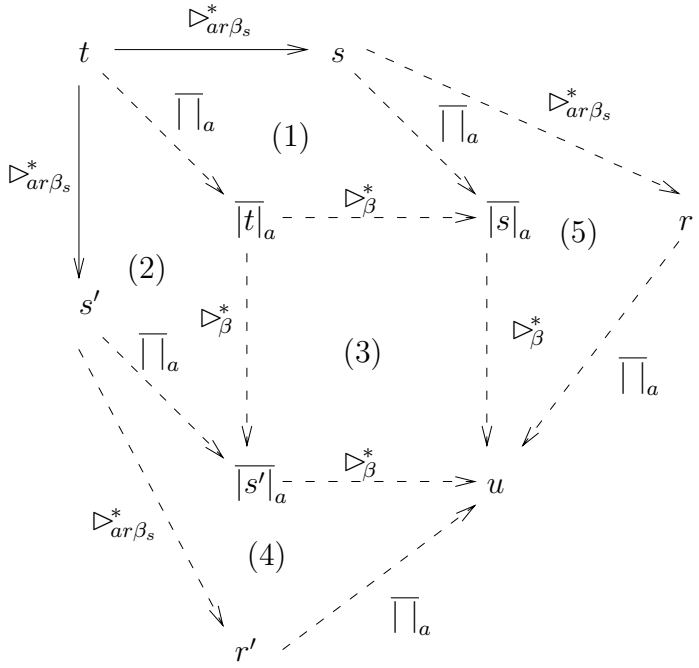
(2) Since $x\rhd^*_{\beta'}y$, by Proposition 10, $x\rhd^*_{rm\beta_s}y$. But then, by Lemma 13, there must be a z such that $x\rhd^*_{ar\beta_s}z$ and $y\rhd^*_{rm}\overline{z}$. However, y is in $\rhd_{rm}$-normal form. Hence $\overline{z} = y$.

Proof of Theorem 5 □

The uniqueness of $\rhd_{ar\beta_s}$-normal forms for any given expression is also a matter of interest. As in the case of $\rhd_{ar}$-normal forms, we have uniqueness up to annotations. This is an immediate consequence of the following "weak" confluence result.

Theorem 6 *Let t be an annotated suspension expression and let s and s' be such that $t\rhd^*_{ar\beta_s}s$ and $t\rhd^*_{ar\beta_s}s'$. Then there are annotated suspension expressions r and r' such that $s\rhd^*_{ar\beta_s}r$, $s'\rhd^*_{ar\beta_s}r'$, and $\overline{r} = \overline{r'}$.*

Proof of Theorem 6 The lemma is evident from the diagram below.



The arrows labeled with $\overline{|\ |}_a$ in this diagram correspond to the use of Definition 27. The dashed arrows in the various faces are justified as follows: those in (1) and (2) by using Theorem 5, the remaining ones in (3) by an obvious extension of Proposition 3, and those in (4) and (5) by using, again, Theorem 5.

Proof of Theorem 6 □

Theorem 6 shows that there is some flexibility in the order of rewriting in simulating β-reduction. It is well known that not all reduction sequences will produce a normal form even when one exists and this situation is not altered in the context of our rewrite system. The particular reduction strategy to be used must be determined by a consideration of this issue as well as issues of efficiency and ease of implementation. At a level of detail, our rewrite system presents some choices in the rule schema to be used even after the redex to be rewritten has been selected. These choices are between (aβ_s) and (aβ'_s), between (ar6) and (ar8), and between (ar7) and (ar9). As already noted, using rule schemata (ar8) and (ar9) in opposition to (ar6) and (ar7)

reduces the reduction work and preserves any existing sharing of terms and is therefore the preferred course. This preference can be systematized by modifying the annotation u in rule schemata (ar6) and (ar7) to o.[4] The choice between (aβ_s) and (aβ'_s) is somewhat more delicate, posing a dilemma between possible sharing in reduction and a sharing in substitution walks.

7 Generalized head normal forms and reduction procedures

We now consider matters relevant to the use of the refined suspension notation in comparing lambda terms. The de Bruijn representation of these terms simplifies this comparison operation by eliminating names for variables, and this attribute obviously carries over to our notation. Our notation additionally permits laziness in substitution. We provide a basis for exploiting this feature in combination with laziness in β-contraction by suitably generalizing the head normal forms encountered in Section 2. We also identify a notion of head reduction sequences as a means for finding such forms. An interesting aspect of our version of this notion is that it permits the *simultaneous* rewriting of shared copies of head redexes and thus subsumes the sequences that might be produced by graph-based and environment-based reduction procedures. We show that our head reduction sequences terminate whenever the usual ones do on the underlying lambda terms. This observation, coupled with the fine-grained nature of the suspension notation, is useful in proving the correctness of actual reduction procedures. We illustrate this fact relative to a particular graph-based procedure that we present for finding head normal forms for annotated suspension terms.

7.1 Head normal forms and head reduction sequences

Our adaptation of the notion of head normal forms to annotated suspension is as follows:

Definition 28 *A head normal form relative to $\rhd_{ar\beta_s}$, or a $\rhd_{ar\beta_s}$-hnf, is an annotated suspension term of the form*

[4]The more general forms of these rule schemata were chosen initially to simplify the proof of confluence; they aided, in particular, in the proof of Lemma 13.

$$(\lambda_{u_1} \ldots (\lambda_{u_n} (\ldots (h\ t_1)_{u_{n+1}}\ \ldots\ t_m)_{u_{n+m}}) \ldots)$$

where, for $1 \leq i \leq (n+m)$, u_i *is either* o *or* c *and* h *is either a constant or a variable reference. We refer to* $t_1, \ldots, t_m$ *as the arguments of such an expression, to* h *as its head, and to* n *as its binder length. A* $\rhd_{ar\beta_s}$*-hnf may, in particular instances, have no arguments and its binder may be empty. An annotated suspension term is said to be a weak* $\rhd_{ar\beta_s}$*-hnf if it is a* $\rhd_{ar\beta_s}$*-hnf or it is of the form* $(\lambda_c\, t)$ *or* $(\lambda_o\, t)$*. A (weak)* $\rhd_{ar\beta_s}$*-hnf* t *is a (weak)* $\rhd_{ar\beta_s}$*-hnf of an annotated suspension term* s *if* $s \rhd^*_{ar\beta_s} t$*.*

The following theorem, which follows immediately from definitions, states a correspondence between head normal forms relative to $\rhd_\beta$ and $\rhd_{ar\beta_s}$. An obvious significance of this theorem is that $\rhd_{ar\beta_s}$-hnfs can be used directly in the comparison of lambda terms.

Theorem 7 *Let* t *be an annotated suspension term. If* t *is a* $\rhd_{ar\beta_s}$*-hnf with arguments* $t_1, \ldots, t_m$*, head* h*, and binder length* n*, then* $\overline{|t|}_a$ *is a* $\rhd_\beta$*-hnf with binder length* n *and, in fact,* $\overline{|t|}_a = (\lambda \ldots (\lambda\, (\ldots (h\ \overline{|t_1|}_a)\ \ldots\ \overline{|t_m|}_a)) \ldots)$*.*

The reduction of annotated suspension terms to head normal form is based on the rewriting of head redexes. We relativize this notion to the collection of rewrite rules currently of interest.

Definition 29 *An annotated suspension term* t *has (weak) head* $\rhd_{ar\beta_s}$*-redexes only if it is not a (weak)* $\rhd_{ar\beta_s}$*-hnf and, in this case, these are given as follows:*

1. *Let* t *be of the form* $(t_1\ t_2)_o$ *or* $(t_1\ t_2)_c$*. If* t *is a* $\rhd_{a\beta_s}$*-redex, then it is its sole (weak) head* $\rhd_{ar\beta_s}$*-redex. Otherwise, the weak head* $\rhd_{ar\beta_s}$*-redexes of* t_1 *are its (weak) head* $\rhd_{ar\beta_s}$*-redexes; notice that* t_1 *cannot be a weak* $\rhd_{ar\beta_s}$*-hnf here.*

2. *Let* t *be of the form* $(\lambda_c\, t_1)$ *or* $(\lambda_o\, t_1)$*. Then the head* $\rhd_{ar\beta_s}$*-redexes of* t *are those of* t_1*. (This case does not arise if* t *is not a weak* $\rhd_{ar\beta_s}$*-hnf.)*

3. *Let* t *be of the form* $[\![t_1, ol, nl, e]\!]_c$ *or* $[\![t_1, ol, nl, e]\!]_o$*. If* t *is a* $\rhd_{ar}$*-redex, then it has itself as a (weak) head* $\rhd_{ar\beta_s}$*-redex. Further, every (weak) head* $\rhd_{ar\beta_s}$*-redex of* t_1 *is a (weak) head* $\rhd_{ar\beta_s}$*-redex of* t*.*

We now define our notion of head reduction sequences. We note here that two subexpressions of a given expression are considered to be non-overlapping just in case neither is contained in the other. Two redexes of this kind can be

rewritten at the same time without the possibility of interference. In anticipation of shared representations of subparts of an expression, our definition of head reduction sequences allows for such simultaneous rewritings.

Definition 30 *A (weak) head $\rhd_{ar\beta_s}$-reduction sequence of an annotated suspension term t is a sequence $t = r_0, r_1, r_2, \ldots, r_n, \ldots$, where, for $i \geq 0$, there is an annotated suspension term succeeding r_i if r_i is not a (weak) $\rhd_{ar\beta_s}$-hnf and, in this case, r_{i+1} is obtained from r_i by simultaneously rewriting a finite set of non-overlapping subterms that includes a (weak) head $\rhd_{ar\beta_s}$-redex using the rule schemata in Figures 4 and 5. Such a sequence terminates if, for some $m \geq 0$, it is the case that r_m is a (weak) $\rhd_{ar\beta_s}$-hnf. A (weak) head $\rhd_{ar\beta_s}$-reduction from t to s is a finite initial segment of some (weak) head $\rhd_{ar\beta_s}$-reduction sequence of t that has s as its last element.*

Head $\rhd_{ar\beta_s}$-reduction sequences are not unique for two reasons: a given term in the sequence may have more than one head $\rhd_{ar\beta_s}$-redex and there may be a choice in additional redexes to be rewritten at any point. However, the redundancy is inconsequential in the sense that *every* head $\rhd_{ar\beta_s}$-reduction sequence of an annotated suspension term terminates if the term has a $\rhd_{ar\beta_s}$-hnf. We establish this result through a sequence of observations that culminate in Theorem 8.

Lemma 14 *Let q be a de Bruijn term of the form*

$$(\ldots(((\lambda\, q_1)\ q_2)\ q_3)\ \ldots\ q_m).$$

Suppose, in addition, that $((\lambda\, q_1)\ q_2) \rightarrow r$ is a β-contraction rule. If $q \rhd_\beta^* p$, then either p is of the form $(\ldots(((\lambda\, p_1)\ p_2)\ p_3)\ \ldots\ p_m)$, where, for $1 \leq i \leq m$, $q_i \rhd_\beta^* p_i$, or $(\ldots(r\ q_3)\ \ldots\ q_m) \rhd_\beta^* p$.

Proof of Lemma 14 Use an induction on the length of the sequence by which $q \rhd_\beta^* p$. The lemma is obvious when the length is 0. If the length is $i+1$, we consider the possibilities for the first rewriting step. If this is a replacement of the subterm $((\lambda\, q_1)\ q_2)$, then the lemma is again obvious. Otherwise, the second term in the sequence has the form $(\ldots(((\lambda\, q_1')\ q_2')\ q_3')\ \ldots\ q_m')$, where, for $1 \leq i \leq m$, $q_i \rhd_\beta^* q_i'$. By hypothesis, either p is of the form $(\ldots(((\lambda\, p_1)\ p_2)\ p_3)\ \ldots\ p_m)$, where, for $1 \leq i \leq m$, $q_i' \rhd_\beta^* p_i$, or, for $r' = S(q_1'; q_2', 1, 2, \ldots)$, it is the case that $(\ldots(r'\ q_3')\ \ldots\ q_m') \rhd_\beta^* p$. In the first case, the lemma follows from the transitivity of $\rhd_\beta^*$ and, in the second case, we also use Proposition 2.

Proof of Lemma 14 □

Lemma 15 *Let r be a de Bruijn term that has a (weak) head $\rhd_\beta$-redex, let s result from r by rewriting this redex using a β-contraction rule, and let t be such that $s \rhd_\beta^* t$. Further, let e_1 and e_2 be simple environments such that $|e_1| \rhd_{\beta'}^* |e_2|$ and the suspension $[\![r, ol, nl, e_1]\!]$ is well formed. Then $|[\![r, ol, nl, e_1]\!]|$ has a (weak) head $\rhd_\beta$-redex, and the term p that results from rewriting this redex by a β-contraction rule is such that $p \rhd_\beta^* |[\![t, ol, nl, e_2]\!]|$.*

Proof of Lemma 15 From Propositions 7 and 2 and the facts that $s \rhd_\beta^* t$ and $|e_1| \rhd_{\beta'}^* |e_2|$ it can be seen that $|[\![s, ol, nl, e_1]\!]| \rhd_\beta^* |[\![t, ol, nl, e_2]\!]|$. Thus, the lemma would follow if $|[\![r, ol, nl, e_1]\!]|$ has a (weak) head $\rhd_\beta$-redex and rewriting this redex by a β-contraction rule produces $|[\![s, ol, nl, e_1]\!]|$. This is seen to be the case by induction on the structure of r. If r is a $\rhd_\beta$-redex, Proposition 2 yields the desired conclusion. If r is of the form $(r_1\ r_2)$ and is not a $\rhd_\beta$-redex, then s is of the form $(s_1\ r_2)$, where s_1 results from r_1 by rewriting a weak head $\rhd_\beta$-redex using a β-contraction rule. Thus, $|[\![r_1, ol, nl, e_1]\!]|$ has a weak head $\rhd_\beta$-redex and rewriting this in the required manner yields $|[\![s_1, ol, nl, e_1]\!]|$. But then $(|[\![s_1, ol, nl, e_1]\!]|\ |[\![r_2, ol, nl, e_1]\!]|)$ results from $(|[\![r_1, ol, nl, e_1]\!]|\ |[\![r_2, ol, nl, e_1]\!]|)$ by rewriting a (weak) head $\rhd_\beta$-redex. The lemma follows from noting that these two terms are identical to $|[\![s, ol, nl, e_1]\!]|$ and $|[\![r, ol, nl, e_1]\!]|$, respectively. A similar argument can be provided when r is an abstraction.

Proof of Lemma 15 □

Lemma 16 *Let q be an annotated suspension term that has a (weak) head $\rhd_{ar\beta_s}$-redex that is also a $\rhd_{a\beta_s}$-redex and let p result from q by simultaneously rewriting a set of non-overlapping subterms of q that includes this redex using the rule schemata in Figures 4 and 5. Then $\overline{|q|}_a$ has a (weak) head $\rhd_\beta$-redex, and the de Bruijn term p' that results from rewriting this redex using a β-contraction rule is such that $p' \rhd_\beta^* \overline{|p|}_a$.*

Proof of Lemma 16 Use induction on the structure of q. If q is itself the (weak) head $\rhd_{ar\beta_s}$-redex in question, then it is the only subterm rewritten and the desired conclusion follows from Lemma 12. Otherwise, we consider the cases for the structure of q.

Suppose q is of the form $(q_1\ q_2)_u$, where u is o or c. Then p is of the form $(p_1\ p_2)_u$, where p_1 is obtained from q_1 by rewriting some non-overlapping subterms that include a weak head $\rhd_{ar\beta_s}$-redex of the required kind and $q_2 \rhd_{ar\beta_s}^* p_2$. By hypothesis, $\overline{|q_1|}_a$ has a weak head $\rhd_\beta$-redex. Further, if p_1' results from rewriting this redex using a β-contraction rule, then $p_1' \rhd_\beta^* \overline{|p_1|}_a$.

Since $\overline{|q|}_a = (\overline{|q_1|}_a\ \overline{|q_2|}_a)$, $\overline{|q|}_a$ has a head $\rhd_\beta$-redex and rewriting this yields $(p'_1\ \overline{|q_2|}_a)$. Using Theorem 5 and the fact that $q_2 \rhd^*_{ar\beta_s} p_2$, $\overline{|q_2|}_a \rhd^*_\beta \overline{|p_2|}_a$. Observing that $\overline{|p|}_a = (\overline{|p_1|}_a\ \overline{|p_2|}_a)$, it follows that $p' = (p'_1\ \overline{|q_2|}_a) \rhd^*_\beta \overline{|p|}_a$.

An argument similar to that above suffices when q is an abstraction. The only remaining case is that of a suspension. Let q be of the form $[\![r, ol, nl, e]\!]_u$, where u is o or c. Then p is of the form $[\![t, ol, nl, e']\!]_u$, where t results from r by rewriting a (weak) head $\rhd_{ar\beta_s}$-redex that is also a $\rhd_{a\beta_s}$-redex together with some other non-overlapping subterms and $e \rhd^*_{ar\beta_s} e'$. By Theorem 5, $\overline{|e|}_a \rhd^*_{\beta'} \overline{|e'|}_a$. By hypothesis, $\overline{|r|}_a$ has a (weak) head $\rhd_\beta$-redex, and rewriting it produces a (de Bruijn) term r' such that $r' \rhd^*_\beta \overline{|t|}_a$. By Lemma 15, it follows that $|[\![\overline{|r|}_a, ol, nl, \overline{|e|}_a]\!]|$ has a (weak) head $\rhd_\beta$-redex, and rewriting it yields a term p' such that $p' \rhd^*_\beta |[\![\overline{|t|}_a, ol, nl, \overline{|e'|}_a]\!]|$. The lemma is now seen to hold in this last case by observing that $\overline{|q|}_a = |[\![\overline{|r|}_a, ol, nl, \overline{|e|}_a]\!]|$ and that $\overline{|p|}_a = |[\![\overline{|t|}_a, ol, nl, \overline{|e'|}_a]\!]|$.

Proof of Lemma 16 □

Lemma 17 *Let t be an annotated suspension term such that the head $\rhd_\beta$-reduction sequence of $\overline{|t|}_a$ terminates. Then any head $\rhd_{ar\beta_s}$-reduction sequence of t terminates.*

Proof of Lemma 17 Let s be an annotated suspension term and let s' be a de Bruijn term such that $s' \rhd^*_\beta \overline{|s|}_a$. We then claim the following: if the head $\rhd_\beta$-reduction sequence of s' terminates, then any head $\rhd_{ar\beta_s}$-reduction sequence of s terminates. The lemma is a consequence of this claim.

We prove the claim by an induction on the length of the head $\rhd_\beta$-reduction sequence of s'.

Suppose this length is 1. In this case, $\overline{|s|}_a$ must be a $\rhd_\beta$-hnf. Thus, by Lemma 16, s cannot have a head $\rhd_{ar\beta_s}$-redex that is a $\rhd_{a\beta_s}$-redex. The noetherianity of $\rhd_{ar}$ now yields the claim.

Suppose the length of the sequence is $i+1$. In this case, s' has the form

$$(\lambda \ldots (\lambda\, (\ldots (((\lambda\, q'_1)\ q'_2)\ q'_3)\ \ldots\ q'_{m+2})) \ldots).$$

Let r' be the second term in the head $\rhd_\beta$-reduction sequence of s'. Then r' has the structure

$$(\lambda \ldots (\lambda\, (\ldots (q'\ q'_3) \ldots q'_{m+2})) \ldots)$$

where $q' = S(q'_1; q'_2, \#1, \#2, \ldots)$. Since $s' \rhd^*_\beta \overline{|s|}_a$, by Lemma 14, either $r' \rhd^*_\beta \overline{|s|}_a$ or $\overline{|s|}_a$ has the form $(\lambda \ldots (\lambda\,(\ldots(((\lambda\, p'_1)\ p'_2)\ p'_3)\ \ldots\ p'_{m+2}))\ldots)$, where, for $1 \leq i \leq (m+2)$, $q'_i \rhd^*_\beta p'_i$. The inductive hypothesis yields the claim in the first case. In the second case, let $p' = S(p'_1; p'_2, \#1, \#2, \ldots)$. Since $q'_1 \rhd^*_\beta p'_1$ and $q'_2 \rhd^*_\beta p'_2$, by Proposition 2, $q' \rhd^*_\beta p'$. Now, by Lemma 16 and the noetherianity of $\rhd_{ar}$, any head $\rhd_{ar\beta_s}$-reduction sequence of s must have a (finite) initial segment $s = s_0, \ldots, s_l$, where s_l is such that

$$(\lambda \ldots (\lambda\,(\ldots(p'\ p'_3)\ \ldots\ p'_{m+2}))\ldots) \rhd^*_\beta \overline{|s_l|}_a.$$

Thus $(\lambda \ldots (\lambda\,(\ldots(q'\ q'_3)\ldots q'_{m+2}))\ldots) \rhd^*_\beta (\lambda \ldots (\lambda\,(\ldots(p'\ p'_3)\ \ldots\ p'_{m+2}))\ldots)$ and hence, by the transitivity of $\rhd^*_\beta$, $r' \rhd^*_\beta \overline{|s_l|}_a$. By the inductive hypothesis, any head $\rhd_{ar\beta_s}$-reduction sequence of s_l must terminate. But then this property must hold also for s.

Proof of Lemma 17 □

Theorem 8 *An annotated suspension term t has a $\rhd_{ar\beta_s}$-hnf if and only if every head $\rhd_{ar\beta_s}$-reduction sequence of t terminates.*

Proof of Theorem 8 The "if" part is obvious. For the "only if," suppose t has a $\rhd_{ar\beta_s}$-hnf. By Theorems 5 and 7, $\overline{|t|}_a$ has a $\rhd_\beta$-hnf. By Proposition 4, the head $\rhd_\beta$-reduction sequence of $\overline{|t|}_a$ terminates. The theorem then follows from Lemma 17.

Proof of Theorem 8 □

Using Theorem 5, Lemma 17, and Proposition 4, we see that an annotated suspension term t has a $\rhd_{ar\beta_s}$-hnf if and only if $\overline{|t|}_a$ has a $\rhd_\beta$-hnf. Suppose, now, that we wish to determine whether the de Bruijn terms t and s are equal. Assuming t' and s' are annotated suspension terms that are such that $\overline{|t'|}_a = t$ and $\overline{|s'|}_a = s$, we may proceed as follows: we reduce t' and s' to $\rhd_{ar\beta_s}$-hnfs, compare the binder lengths and the heads of the resulting terms for identity, and finally, if this is still relevant, recursively compare the arguments for equality.

7.2 A procedure for head normalization

We now present a procedure for finding head $\triangleright_{ar\beta_s}$-normal forms of annotated suspension terms. To lend concreteness to the syntax of programs, we will use the language SML in this presentation. We assume a basic familiarity with this language, such as can be obtained from perusing [Har86].

The procedure that we describe employs a graph-based representation for terms. The details of this representation are given by the following SML type declarations.

```
datatype
  rawterm = const of string
          | bv of int
          | ptr of (rawterm ref)
          | clam of (rawterm ref)
          | olam of (rawterm ref)
          | capp of (rawterm ref) * (rawterm ref)
          | oapp of (rawterm ref) * (rawterm ref)
          | csusp of (rawterm ref) * int * int * (envitem list)
          | osusp of (rawterm ref) * int * int * (envitem list)
and
  envitem = dum of int
          | bndg of (rawterm ref) * int

type term = (rawterm ref)
```

The declarations of the types *rawterm* and *envitem* reflect, for the most part, the possible structures for annotated suspension terms and environment items. A term is represented as a pointer to an expression of type *rawterm*. SML expressions of type *rawterm*, *envitem*, and *term* can be visualized as directed graphs. We refer to such expressions as being acyclic if the graphs they correspond to in this sense are acyclic. An important assumption for us is that all expressions we deal with are acyclic. We expect that "input" expressions satisfy this acyclicity requirement, and we will show that our programs preserve this property. The acyclicity condition is made essential use of in the following definition that states precisely the correspondence between the SML representation and the abstract syntax of annotated suspension expressions.

Definition 31 *The function ζ from the classes of an acyclic SML expressions of type term, envitem, and (envitem list) to, respectively, the classes of annotated suspension terms, annotated environment items, and annotated environments is given recursively as follows:*

1. *if x is of the form ref(const(c)), then $\zeta(x) = c$,*
2. *if x is of the form ref(bv(i)), then $\zeta(x) = \#i$,*
3. *if x is of the form ref(ptr(t)), then $\zeta(x) = \zeta(t)$,*
4. *if x is of the form ref(clam(t)) or ref(olam(t)), then $\zeta(x)$ is, respectively, $(\lambda_c\, \zeta(t))$ or $(\lambda_o\, \zeta(t))$,*
5. *if x is of the form ref(capp(t_1, t_2)) or ref(oapp(t_1, t_2)), then $\zeta(x)$ is, respectively, $(\zeta(t_1)\ \zeta(t_2))_c$ or $(\zeta(t_1)\ \zeta(t_2))_o$,*
6. *if x is of the form ref(csusp(t, ol, nl, e)) or ref(osusp(t, ol, nl, e)), then $\zeta(x)$ is, respectively, $[\![\zeta(t), ol, nl, \zeta(e)]\!]_c$ or $[\![\zeta(t), ol, nl, \zeta(e)]\!]_o$,*
7. *if x is of the form dum(i), then $\zeta(x) = @i$,*
8. *if x is of the form bndg(t, i), then $\zeta(x) = (\zeta(t), i)$,*
9. *if x is nil, then $\zeta(x) = nil$, and*
10. *if x is of the form et::e, then $\zeta(x) = \zeta(et) :: \zeta(e)$.*

We have conflated here the SML representation of constants and natural numbers and the abstract syntax for these objects.

The purpose of the constructor *ptr* in our representation of annotated suspension expressions needs to be clarified. At certain points in our programs, we want to identify (the representations of) terms in a way that makes the subsequent rewriting of one of these correspond also to the rewriting of the others. Such an identification is usually obtained by representing both expressions as pointers to a common location whose contents can be changed to effect shared rewritings. In SML it is possible to update only references and so the common location must itself be a pointer. The constructor *ptr* is used to encode indirections of this kind when they are needed.

Given the possibility for indirection, functions for looking up the value of a term and for assigning one term to another are needed. The functions *deref* and *assign* defined below serve these purposes.

```
fun deref (term as ref(ptr(t))) = deref(t)
  | deref term = term

fun assign (t1,ref(ptr(t))) = assign(t1,t)
  | assign (t1,t2) = t1 := ptr(t2)
```

Invocations of these functions with acyclic expressions as arguments must obviously terminate. Our programs are, in fact, designed to never introduce more than one level of indirection and the definitions of these functions can even be simplified to a non-recursive form. The following additional properties are easily seen to hold of these functions and will be utilized implicitly below: (1) $\zeta(\mathit{deref}(t)) = \zeta(t)$, (2) $\mathit{deref}(t)$ is not of the form $\mathit{ref}(\mathit{ptr}(t))$, and (3) if $t = \zeta(t_2)$ before $\mathit{assign}(t_1, t_2)$ is invoked, then $\zeta(t_1) = t$ and $\zeta(t_2) = t$ after it terminates.

The head normalization procedure that we describe functions as follows: it descends through top-level abstractions and applications looking for a head $\rhd_{ar\beta_s}$-redex that is also a $\rhd_{a\beta_s}$-redex. If such a redex is found, then it is rewritten and the process repeats. Otherwise the procedure terminates, having discovered a head $\rhd_{ar\beta_s}$-normal form. Of course, it is possible to encounter a suspension at the top-level. In this case, before the basic scheme for the procedure can be utilized, it is necessary to rewrite this suspension (that is a head $\rhd_{ar\beta_s}$-redex) so as to expose a structure for the term that corresponds to a constant, a variable reference, an abstraction or an application. This effect is realized by the function *lazy_read* whose definition appears in Figure 6.

In dealing with variable references embedded in suspensions, *lazy_read* combines several rewriting steps into one. Such a combination is justified by the following lemma:

Lemma 18 *Let t be the annotated suspension term $[\![\#i, ol, nl, e]\!]_u$, where u is either c or o. Then there is a (weak) head $\rhd_{ar\beta_s}$-reduction from*

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

Proof of Lemma 18 Use an induction on i using the rule schemata (ar2)–(ar5).

Proof of Lemma 18 □

```
fun nth(x::l,1) = x
  | nth(x::l,n) = nth(l,n-1)

fun lazy_read(term as ref(csusp(t,ol,nl,env))) =
        lazy_read_aux(term,deref(t),ol,nl,env,true)
  | lazy_read(term as ref(osusp(t,ol,nl,env))) =
        lazy_read_aux(term,deref(t),ol,nl,env,false)
  | lazy_read(_) = ()
and
    lazy_read_aux(t,t1 as ref(const(_)),_,_,_,_) = t := !t1
  | lazy_read_aux(t,t1 as ref(capp(_,_)),_,_,_,_) = t := ptr(t1)
  | lazy_read_aux(t,t1 as ref(clam(_)),_,_,_,_) = t := !t1
  | lazy_read_aux(t,t1 as ref(csusp(_,_,_,_)),_,_,_,_) =
        (lazy_read(t1) ; assign(t,t1))
  | lazy_read_aux(t,ref(bv(i)),ol,nl,e,closed) =
        if (i > ol) then t := bv(i+nl-ol)
        else ((fn dum(j) => t := bv(nl-j)
                | bndg(t1,nl1) =>
                    if nl = nl1 then (lazy_read(deref(t1)) ; assign(t,t1))
                    else (fn ref(osusp(ti,oli,nli,ei)) =>
                                (t := osusp(ti,oli,nli +(nl-nl1),ei) ;
                                 lazy_read(t))
                           | _ =>
                                (if closed
                                 then t := csusp(t1,0,nl -nl1,nil)
                                 else t := osusp(t1,0,nl -nl1,nil) ;
                                 lazy_read(t)) ) (deref(t1))
            ) (nth(e,i)))
  | lazy_read_aux(t,ref(oapp(t1,t2)),ol,nl,env,closed) =
        if closed
        then t := capp(ref(csusp(t1,ol,nl,env)),ref(csusp(t2,ol,nl,env)))
        else t := oapp(ref(osusp(t1,ol,nl,env)),ref(osusp(t2,ol,nl,env)))
  | lazy_read_aux(t,ref(olam(t1)),ol,nl,env,closed) =
        if closed then t := clam(ref(osusp(t1,ol+1,nl+1,dum(nl)::env)))
        else t := olam(ref(osusp(t1,ol+1,nl+1,dum(nl)::env)))
  | lazy_read_aux(t,t1,ol,nl,env,closed) =
        (lazy_read(t1) ; lazy_read_aux(t,deref(t1),ol,nl,env,closed))
```

Figure 6: Exposing a top-level non-suspension structure for a term

The following obvious property explains the purpose of the function *nth* that is used in *lazy_read*:

Lemma 19 *Let e be an SML expression of type (envitem list) and let i be a positive integer such that $i \leq len(\zeta(e))$. Then nth(e,i) terminates. Further, if the value returned by it is dum(j), then $\zeta(e)[i] = @j$, and if it is bndg(t,j), then $\zeta(e)[i] = (\zeta(t), j)$.*

Suppose t is an SML expression of type *term* such that *deref(t)*$= t$. Then clearly $\zeta(t)$ is not a suspension at the termination of *lazy_read(t)*. The following lemma, a correctness observation for *lazy_read*, further explains the relationship between the values of t before and after such a function call.

Lemma 20 *Let t be an SML expression of type term and suppose that all our expressions are acyclic. Then lazy_read(t) terminates, preserving the acyclicity property. Further, if $\zeta(t) = x$ prior to the invocation and $\zeta(t) = y$ after the termination, then there is a (weak) head $\rhd_{ar\beta_s}$-reduction from x to y. Finally, if x is an annotated suspension, then this (weak) head $\rhd_{ar\beta_s}$-reduction is of length greater than 1.*

Proof of Lemma 20 Let s' and e' be acyclic SML expressions of type *term* and *(envitem list)* with initial values such that $s = \zeta(s')$ and $e = \zeta(e')$ and let r' be an SML expression of type *term* that is not pointed to from either s' or e'. Also assume that s and e are such that $[\![s, ol, nl, e]\!]_o$ ($[\![s, ol, nl, e]\!]_c$) is consistently annotated. We then claim that *lazy_read_aux(r',deref(s'),ol,nl,e',false)* (*lazy_read_aux(r',deref(s'),ol,nl,e',true)*) terminates and does not introduce cycles into expressions and, further, if $r = \zeta(r')$ at termination, then there is a (weak) head $\rhd_{ar\beta_s}$-reduction of length greater than 1 from $[\![s, ol, nl, e]\!]_o$ ($[\![s, ol, nl, e]\!]_c$) to r. The lemma is an immediate consequence of this claim.

The claim is proved by induction on the ordering induced by $\succ$ on expressions of the form $\overline{[\![s, ol, nl, e]\!]_o}$ ($\overline{[\![s, ol, nl, e]\!]_c}$). From the assumption about r' and an examination of the function definitions, it is easily seen that no cycles are introduced into our expressions if there weren't any to begin with. The remaining requirements are shown by considering each possibility for the structure of s in turn.

Termination is obvious in the cases when s is a constant, an abstraction or an application. Further, in all these cases, $[\![s, ol, nl, e]\!]_o$ ($[\![s, ol, nl, e]\!]_c$) is its own (weak) head $\rhd_{ar\beta_s}$-redex, and the value that r' is set to is such that this expression rewrites to $\zeta(r')$ by one of the reading rules for annotated suspension expressions.

Lemmas 18 and 19 may have to be used in the case when s is a variable reference. We illustrate the argument by considering one situation: that when s is $\#i$ for $i \leq ol$ and $e[i]$ is of the form $(\llbracket s_i, ol_i, nl_i, e_i \rrbracket_o, nl'_i)$. Notice that this situation does not arise if s and e are such that $\llbracket s, ol, nl, e \rrbracket_c$ is consistently annotated. Now, from Lemma 18 and rule schema (ar11), we see that there is a (weak) head $\rhd_{ar\beta_s}$-reduction of length at least 3 from $\llbracket s, ol, nl, e \rrbracket_o$ to $\llbracket s_i, ol_i, (nl_i + (nl - nl'_i)), e_i \rrbracket_o$. From this it can also be seen that

$$\overline{\llbracket s, ol, nl, e \rrbracket_o} \succ \overline{\llbracket s_i, ol_i, (nl_i + (nl - nl'_i)), e_i \rrbracket_o}.$$

Suppose now that m' is an acyclic SML expression that is such that

$$\zeta(m') = \llbracket s_i, ol_i, (nl_i + (nl - nl'_i)), e_i \rrbracket_o$$

at the invocation of *lazy_read(m')*. By the inductive hypothesis, this call must terminate and, if $m = \zeta(m')$ at termination, then there is a (weak) head $\rhd_{ar\beta_s}$-reduction from $\llbracket s_i, ol_i, (nl_i + (nl - nl'_i)), e_i \rrbracket_o$ to m. It follows from this that *lazy_read_aux(r',deref(s'),ol,nl,e',false)* must also terminate and that r' must be set at termination to a value that satisfies the requirements of the claim.

The argument in the case when s is a suspension is similar to that when s is a variable reference, except that the induction hypothesis may have to be invoked twice.

Proof of Lemma 20 □

Our head normalization procedure is given by the function *head_norm* defined in Figure 7. Actually, this function serves a twofold purpose, depending on the value of its second argument. Assuming its first argument (that must be of type *term*) is t, then it is expected to find a $\rhd_{ar\beta_s}$-hnf of $\zeta(t)$ when its second argument is *false* and a weak $\rhd_{ar\beta_s}$-hnf of $\zeta(t)$ when this argument is *true*. The need for a function with this character is understood by considering the processing of an SML expression that represents a term of the form $(t_1\ t_2)_c$ or $(t_1\ t_2)_o$. The head $\rhd_{ar\beta_s}$-redex of this term is identical to that of t_1 except in the case that t_1 is an abstraction. Thus, the construction of a head $\rhd_{ar\beta_s}$-reduction sequence for such a term involves looking for a weak $\rhd_{ar\beta_s}$-hnf of t_1.

The purpose of the function *beta_contract* is expressed in the following lemma whose proof is obvious.

```
fun beta_contract(t,t1 as ref(osusp(t3,ol,nl,dum(nl1)::e)),t2,true) =
        if nl = nl1+1 then t := csusp(t3,ol,nl1, bndg(t2,nl1)::e)
        else t := csusp(t1,1,0,[bndg(t2,0)])
  | beta_contract(t,t1 as ref(osusp(t3,ol,nl,dum(nl1)::e)),t2,false) =
        if nl = nl1+1 then t := osusp(t3,ol,nl1, bndg(t2,nl1)::e)
        else t := osusp(t1,1,0,[bndg(t2,0)])
  | beta_contract(t,t1,t2,true) = t := csusp(t1,1,0,[bndg(t2,0)])
  | beta_contract(t,t1,t2,false) = t := osusp(t1,1,0,[bndg(t2,0)])

fun head_norm(term as ref(capp(t1,t2)),whnf) =
        (head_norm(t1,true) ;
         (fn ref(clam(t)) => (beta_contract(term,t,t2,true) ;
                              head_norm(term,whnf) )
           |     _          => ()
         ) (deref(t1)))
  | head_norm(term as ref(oapp(t1,t2)),whnf) =
        (head_norm(t1,true) ;
         (fn ref(clam(t)) => (beta_contract(term,t,t2,false) ;
                              head_norm(term,whnf) )
           | ref(olam(t)) => (beta_contract(term,t,t2,false) ;
                              head_norm(term,whnf) )
           |     _          => ()
         ) (deref(t1)))
  | head_norm(ref(clam(t)),false) = head_norm(t,false)
  | head_norm(ref(olam(t)),false) = head_norm(t,false)
  | head_norm(term as ref(csusp(_,_,_,_)),whnf) =
        (lazy_read(term) ; head_norm(term,whnf))
  | head_norm(term as ref(osusp(_,_,_,_)),whnf) =
        (lazy_read(term) ; head_norm(term,whnf))
  | head_norm(term as ref(ptr(t)),whnf) =
        (head_norm(t,whnf) ; assign(term,t))
  | head_norm(_,_) = ()
```

Figure 7: The head normalization procedure

Lemma 21 *Let t', t'_1, and t'_2 be SML expressions of type term and let t_1 and t_2 be $\zeta(t'_1)$ and $\zeta(t'_2)$, respectively. If $t = \zeta(t')$ at the termination of beta_contract(t', t'_1, t'_2, true), then $((\lambda_c\, t_1)\ t_2)_c \rightarrow t$ is a β_s-contraction rule for annotated suspension expressions. If $t = \zeta(t')$ at the termination of beta_contract(t', t'_1, t'_2,false), then $((\lambda_o\, t_1)\ t_2)_c \rightarrow t$ and $((\lambda_c\, t_1)\ t_2)_o \rightarrow t$ is such a rule.*

We now prove the correctness of the head normalization procedure.

Theorem 9 *Let t' be an SML expression of type term and suppose that all our expressions are acyclic. If prior to the function call head_norm(t',false) it is the case that $\zeta(t') = t$ for some t that has a $\rhd_{ar\beta_s}$-hnf, then (1) the call terminates, (2) all expressions continue to be acyclic at termination, and (3) t' has a value at the end such that $\zeta(t')$ is a $\rhd_{ar\beta_s}$-hnf of t.*

Proof of Theorem 9 Since t has a $\rhd_{ar\beta_s}$-hnf, Theorem 8 assures us that every head $\rhd_{ar\beta_s}$-reduction sequence of t terminates. Let s be an annotated suspension term, let s' be an SML expression that is such that $s = \zeta(s')$, and suppose that all our SML expressions are acyclic. We claim the following: If every head $\rhd_{ar\beta_s}$-reduction sequence (weak head $\rhd_{ar\beta_s}$-reduction sequence) of s terminates, then *head_norm(s',false)* (*head_norm(s',true)*) terminates, preserving the property of acyclicity of expressions and setting s' to a value such that $\zeta(s')$ is a $\rhd_{ar\beta_s}$-hnf (weak $\rhd_{ar\beta_s}$-hnf) of s. The theorem is an immediate consequence of this claim.

The two parts of the claim are proved simultaneously by an induction first on the length of the longest (weak) head $\rhd_{ar\beta_s}$-reduction sequence of s and then on the structure of s'. Note that the latter induction requires our SML expressions to be acyclic. The preservation of acyclicity follows easily from Lemma 20 and by observing that none of the assignments in *beta_contract* and *head_norm* introduce cycles where these did not exist already. For the rest, we consider the cases for the structure of s'.

The claim is obviously true if s' is of the form *ref(bv(i))*, *ref(const(c))*, or *ref(ptr(s'_1))*. Suppose s' is of the form *ref(clam(s'_1))* or *ref(olam(s'_1))*. Then s is of the form $(\lambda_c\, s_1)$ or $(\lambda_o\, s_1)$, where $s_1 = \zeta(s'_1)$. Clearly s is its own weak $\rhd_{ar\beta_s}$-hnf. Further, if r_1 is a $\rhd_{ar\beta_s}$-hnf of s_1, then $(\lambda_c\, r_1)$ or $(\lambda_o\, r_1)$ is a $\rhd_{ar\beta_s}$-hnf of s. Finally, the longest head $\rhd_{ar\beta_s}$-reduction sequence of s_1 is at most as long as that of s and the structure of s'_1 is simpler than that of s'. Hence, *head_norm*(s'_1,*false*) terminates setting s'_1 to a value such that

$\zeta(s'_1)$ is a $\rhd_{ar\beta_s}$-hnf of s_1. The claim follows from these observations and an inspection of the definition of *head_norm*.

Suppose s' is of the form *ref(capp(s'_1, s'_2))* or *ref(oapp(s'_1, s'_2))*. Then s is of the form $(s^1_1\ s^1_2)_u$, where u is either c or o and, for $i = 1$ and $i = 2$, $s^1_i = \zeta(s'_i)$. Now, if $s^1_1, \ldots, s^k_1, \ldots$ is a weak head $\rhd_{ar\beta_s}$-reduction sequence of s^1_1 and, for $i \geq 1$, s^{i+1}_2 is obtained from s^i_2 by rewriting some of its subterms that are identical as terms to the weak head $\rhd_{ar\beta_s}$-redex of s^i_1, then $(s^1_1\ s^1_2)_u, (s^1_2\ s^2_2)_u, \ldots, (s^k_1\ s^k_2)_u, \ldots$ is an initial segment of a (weak) head $\rhd_{ar\beta_s}$-reduction sequence of s. Thus, any weak head $\rhd_{ar\beta_s}$-reduction sequence of s^1_1 is at most as long as the longest (weak) head $\rhd_{ar\beta_s}$-reduction sequence of s. Since the structure of s'_1 is simpler than that of s', it follows that *head_norm*(s'_1,*true*) must terminate. Let $r_1 = \zeta(s'_1)$ and $r_2 = \zeta(s'_2)$ at termination. By the argument already outlined, there is a (weak) head $\rhd_{ar\beta_s}$-reduction from s to $(r_1\ r_2)_u$. Now, if r_1 is not of the form $(\lambda_c\, x_1)$ or $(\lambda_o\, x_1)$, then $(r_1\ r_2)_u$ is already a (weak) $\rhd_{ar\beta_s}$-hnf. On the other hand, if it is of either of these forms, then $(r_1\ r_2)_u$ has itself as a (weak) head $\rhd_{ar\beta_s}$-redex. Let $(r_1\ r_2)_u \rightarrow r$ be a β_s-contraction rule for annotated suspension expressions. Then the longest head $\rhd_{ar\beta_s}$-reduction sequence of r is shorter than that of s by at least 1. Hence, if r' is an acyclic SML expression of type term that is such that $r = \zeta(r')$ at the invocation of *head_norm*(r',*false*) (*head_norm*(r',*true*)), then this invocation terminates. Further, if $m = \zeta(r')$ at termination, then m is a (weak) $\rhd_{ar\beta_s}$-hnf of r. However, this term is also a (weak) $\rhd_{ar\beta_s}$-hnf of $(r_1\ r_2)_u$ and, thus, of s. The claim follows easily in the case being considered from these observations, Lemma 21, and an inspection of the definition of *head_norm*.

Suppose s' is of the form *ref(csusp(s'_1, ol, nl, e'))* or *ref(osusp(s'_1, ol, nl, e'))*. As observed already, *lazy_read(s')* terminates. Let $r = \zeta(s')$ at termination. By Lemma 20, there is a (weak) head $\rhd_{ar\beta_s}$-reduction from s to r of length greater than 1. But then the longest (weak) head $\rhd_{ar\beta_s}$-reduction sequence of r is shorter than that of s. Hence, an invocation *head_norm(s',false)* (*head_norm(s',true)*) that follows *lazy_read(s')* must terminate. Further, at the time it does, s' must have a value such that $\zeta(s')$ is a (weak) $\rhd_{ar\beta_s}$-hnf of r. This is also a (weak) $\rhd_{ar\beta_s}$-hnf of s and thus the claim must, once again, be true.

Proof of Theorem 9 □

It is instructive to compare the procedure in [FT90] to *head_norm* when it attempts only to find weak head normal forms. Ignoring the use of anno-

tations, the main difference between these two procedures is in what they do when a term of the form $[\![t, ol, nl, e]\!]_v$ in which t is not a suspension is encountered. The procedure in [FT90] tries to reduce t first, thereby permitting a sharing of this reduction, but leading to a proliferation of structure traversals and possibly preventing other kinds of sharing as noted in Section 4. In contrast, our procedure usually chooses to first percolate the substitution embodied in e over the structure of t, thus destroying the possibility of sharing in the reduction of t. The system of annotations attempts to offset this disadvantage by permitting $[\![t, ol, nl, e]\!]_v$ in certain instances to be reduced directly to t. Even without the benefit of annotations, we believe that the approach used by our procedure is the preferred one in practice. However, there is one kind of situation in which it might be relevant to consider the alternative strategy. This is the case when the suspension encountered is of the form $[\![t, 0, nl, nil]\!]_v$. A term of this kind arises from substituting t into some context and corresponds to an adjustment of variable references in it. Notice that all the adjustments of this kind that arise in the course of reduction to weak head normal forms are vacuous ones, being given by expressions of the form $[\![t, 0, 0, nil]\!]_v$. Our procedure recognizes this fact and simplifies terms of this kind directly to t. It may actually be preferable to reduce t first even when the adjustment of variable references has content. Such an effect may be obtained by modifying our procedures to include suitable invocations of *head_norm* from *lazy_read*.[5] We remark that our definition of (weak) head $\triangleright_{ar\beta_s}$-reduction sequences is general enough to encompass the ones produced by such a strategy as well as the one used in [FT90] and therefore provides a means for verifying the corresponding reduction procedures as well.

8 Conclusion

In this paper we have presented a refinement to the notation for lambda terms described in [NW98]. We have also examined properties of the resulting notation that are relevant to its use in the comparison of lambda terms. In particular, we have shown the correspondence between this notation and the

[5] A behavior of this sort is obtained in [AP81] by using a scheme for identifying bound variables that allows the same "renaming" of free variables to suffice in all contexts. Unfortunately, this alternative treatment of bound variables seems not to have the property of the de Bruijn scheme that is crucial for our purposes: that of obviating (some operation similar to) α-conversion in the comparison of lambda terms.

conventional notation for lambda terms. We have then lifted the notion of head normal forms to its context and have shown how this might be used in checking terms for equality. Using this approach permits the benefits of laziness in substitution to be reaped in the comparison operation. Finally, we have described a procedure for head normalizing terms and have used the tools of analysis made available by our notation in providing a comprehensive proof of its correctness. This procedure has been presented recursively here, but it can be easily rendered into a stack-based form. It has, in fact, been incorporated in this form into an abstract machine for λProlog [NJW93, Nad98].

The comparison of lambda terms considered in this paper is intrinsic to most other operations on the intensions of such terms and the discussions here are therefore relevant in their contexts as well. An operation that is of particular interest to us is that of unifying (typed) lambda terms [Hue75]: this operation is central to λProlog, whose efficient implementation is a major reason for the investigations undertaken here. In its essence, the problem of unifying lambda terms is that of finding substitutions for existentially quantified variables that appear in these terms so as to make the terms equal by virtue of the rules of lambda conversion. In order to deal with this unification problem, it is necessary to extend the class of de Bruijn terms (and, consequently, also the class of suspension terms) with a new category of atomic symbols corresponding to existentially quantified variables.[6] The standard unification procedure, that of Huet, is based on comparing terms in the resulting notation and makes use of the notion of head normal forms. The discussions in this paper are, thus, extremely pertinent to implementing this procedure. In fact, certain aspects of our notation become especially relevant in this context. For example, existentially quantified variables and the terms that are substituted for such variables are closed in the sense that they are unaffected by contractions of enclosing redexes. Our annotation scheme has the capability of recognizing this and, therefore, of being genuinely useful. Similarly, there are benefits to using de Bruijn's scheme beyond the one already mentioned. As an example, when the comparison of terms reduces to that of their arguments, it is technically necessary to carry around a

[6]These variables are to be interpreted here just as they are in the ordinary lambda calculus: substitutions that are made for them must respect the usual non-capture restrictions. An alternative treatment of unification can be provided based on a view of "meta variables" similar to that in [DHK95]. Between these two approaches there is a trade-off whose assessment is beyond the scope of this paper.

context given by the binders of the terms. The use of indices permits this requirement to be realized implicitly. This feature is particularly useful when, as in the case of λProlog, the task of solving a unification problem may have to be suspended and later resumed. Finally, the destructive version of head-normalization that we have presented here is likely to have advantages even in this context, despite the possibility that reductions may have to be backtracked over. In particular, the usual benefits of sharing in reduction will continue to accrue, and the occasional need to trail (and undo) destructive changes will likely be more than offset by the need not to copy the structure of the term above the redex.

Our discussions in this paper have assumed a notion of equality that is based only on α- and β-conversion. It is often desirable to extend this notion by including also the rule of η-conversion. A run-time treatment of this rule in a context where the terms are typed can be obtained by relatively straightforward additions to the discussions in this paper. In essence, the comparison of terms requires the η rule to be used in conjunction with head normal forms, and de Bruijn's scheme and suspensions permit a convenient implementation of this requirement. A discussion of this aspect may be found in [NJW93] and [Nad98].

Acknowledgements

Debra Wilson provided input at an early stage of this work. Useful suggestions were obtained by Dale Miller and anonymous reviewers.

Acknowledgment of support: This work has been supported by NSF grants CCR-89-05825, CCR-92-08465, and CCR-95-96119.

References

[ACCL91] M. Abadi, L. Cardelli, P-L. Curien, and J-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[And71] P. B. Andrews. Resolution in type theory. *Journal of Symbolic Logic*, 36:414–432, 1971.

[AP81] L. Aiello and G. Prini. An efficient interpreter for the lambda-calculus. *The Journal of Computer and System Sciences*, 23:383–425, 1981.

[Bar81] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland Publishing Co., 1981.

[BBLRD96] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalization. *Journal of Functional Programming*, 6(5):699–722, 1996.

[BR91] P. Brisset and O. Ridoux. Naive reverse can be linear. In Koichi Furukawa, editor, *Eighth International Logic Programming Conference*, pages 857–870. MIT Press, June 1991.

[Bru72] 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.

[Bru80] 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.

[CAB+86] 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.

[CCM87] G. Cousineau, P-L. Curien, and M. Mauny. The categorical abstract machine. *The Science of Programming*, 8(2):173–202, 1987.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[Chu40] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Cur86] P-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986.

[DHK95] G. Dowek, Th. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 366–374. IEEE Computer Society Press, June 1995.

[Fie90] J. 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.

[FT90] J. Field and T. Teitelbaum. Incremental reduction in the lambda calculus. In *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 307–322. ACM Press, 1990.

[GMW79] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[Har86] R. Harper. Introduction to Standard ML. Technical Report ECS-LFCS-86-14, Laboratory for Foundations of Computer Science, University of Edinburgh, November 1986. Revised by Nick Rothwell, January 1989, with exercises by Kevin Mitchell.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993.

[HM76] P. Henderson and J. H. Morris. A lazy evaluator. In *Third Annual ACM Symposium on Principles of Programming Languages*, pages 95–103. ACM Press, 1976.

[HS86] J. R. Hindley and J. P. Seldin. *Introduction to Combinatory Logic and Lambda Calculus*. Cambridge University Press, 1986.

[Hue75] G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

[KR97] F. Kamareddine and A. Ríos. Extending the λ-calculus with explicit substitution which preserves strong normalization into

a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, 1997.

[Mel95] P-A. Mellies. Typed λ-calculi with explicit substitutions may not terminate. In *Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334. Springer-Verlag, 1995.

[Mil91] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Muñ96] C. Muñoz. Confluence and preservation of strong normalization in an explicit substitution calculus. In *Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 440–447. IEEE Computer Society Press, July 1996.

[Nad98] G. Nadathur. An explicit substitution notation in a λProlog implementation. Technical Report TR-98-01, Department of Computer Science, University of Chicago, January 1998.

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 64–74. IEEE Computer Society Press, June 1993.

[NJW93] G. Nadathur, B. Jayaraman, and D. S. Wilson. Implementation considerations for higher-order features in logic programming. Technical Report CS-1993-16, Department of Computer Science, Duke University, June 1993.

[NM88] G. Nadathur and D. Miller. An overview of λProlog. In Kenneth A. Bowen and Robert A. Kowalski, editors, *Fifth International Logic Programming Conference*, pages 810–827. MIT Press, August 1988.

[NW90] G. Nadathur and D. S. 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.

[NW98] G. Nadathur and D. S. Wilson. A notation for lambda terms: A generalization of environments. *Theoretical Computer Science*, 198(1-2):49–98, 1998.

[Pau90] L. C. Paulson. Isabelle: The next 700 theorem provers. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[Pfe89] F. Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–322. IEEE Computer Society Press, June 1989.