INFORMATION AND COMPUTATION 72, 46–65 (1987)

A Refinement of Strong Sequentiality for Term Rewriting with Constructors*

SATISH THATTE

Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109

The notion of *call-by-need* evaluation has turned out to be very fruitful when applied to dialects of the λ-calculus (Henderson and Morris, 1976, *in* "Proc. 3rd ACM Symp. on the Principles of Programming Languages, Atlanta; Kahn and Mac Queen, 1977, *in* "IFIP 77," North-Holland, Amsterdam; Turner, 1979, Software Practice and Experience, Vol. 9; Vuillemin, 1974, *J. Comput. Systems Sci.* 9, 332–354; Wadsworth, 1971, Ph.D. dissertation, Oxford University, England). The analogous idea of *sequentiality* for term rewriting systems described by first-order equations has been considered by Hoffman–O'Donnell (1979, *in* "Proc. 5th ACM Symp. on the Principles of Programming Languages, San Antonio," 1984, *in* "Proc. 11th ACM Symp. on the Principles of Programming Languages, Salt Lake City) and Huet–Levy (1979, Technical Report No. 359, INRIA, Le Chesney, France), of which the latter is generally considered to be the most complete theoretical treatment of the subject to date. Huet–Levy (1979) defined the notion of strong sequentiality to describe the class of linear term rewriting systems for which call-by-need computation is practical. This paper introduces an improved version of strong sequentiality called *left* sequentiality. Unlike strong sequentiality, left sequentiality is based on possible rather than arbitrary (and often impossible) sequences of reductions. We show that left sequentiality is more general than strong sequentiality when applied to individual terms, but is equivalent to the latter when cosidered as a property of admissible sets of left-hand sides for systems of equations. Huet–Levy (1979) showed that there are safe redex selection algorithms, i.e., algorithms deriving normal forms whenever possible, for systems based on strongly sequential sets of left-hand sides. We show that there is no algorithm which is safe for *all* systems based on a set of left-hand sides if that set is not left sequential. In other words, left sequentiality is not only sufficient, it is also *necessary* for safe computation based on the analysis of left-hand sides alone. © 1987 Academic Press, Inc.

1. INTRODUCTION

Call-by-need evaluation amounts to leftmost–outermost–next evaluation for ordinary recursive equations and λ-expressions. The formal property one is looking for is "safety," i.e., the guarantee that the evaluation of an

* This work was supported in part by a Faculty Fellowship and Grant from the Rackham School of the University of Michigan.

expression will reduce it to its (normal form) value whenever possible. The critical problem in safe evaluation is to find a "needed" redex, which is any redex that *must* be reduced somewhere along the way to the normal form. The simplicity of the usual call-by-need evaluation comes from the fact that the leftmost–outermost redex is always needed, and any needed redex eventually becomes leftmost–outermost. None of these nice properties holds for term rewriting systems defined by equations, not even when they belong to a nice class like *regular* systems (Hoffmann–O'Donnell, 1984) which always satisfy the Church–Rosser property. Not only is there no standard safe evaluation rule, but it is in general not possible to find a needed redex without looking ahead to check its future behavior. A (regular) system is *sequential* when it allows a needed redex to be found without lookahead in any term not in normal form. But there is one further complication. There is no algorithm for finding needed redices which works for all sequential regular systems. There is also no algorithm to decide if a given regular system is sequential.

Sequentiality is therefore too general a property for the practical application of call-by-need evaluation to regular systems. Huet–Levy (1979) showed that a restricted notion of sequentiality, the so-called *strong* sequentiality was more tractable. The technical formulation of strong sequentiality pays attention only to the left-hand sides of equations by using the idea of arbitrary reduction sequences, i.e., sequences in which a redex is replaced at each step by an arbitrary ground term. This is not an entirely satisfactory notion since such sequences often cannot be produced by *any* rewriting system based on the given left-hand sides.

In this paper, we define a more refined version of strong sequentiality called *left* sequentiality, which is based on *possible* rather than arbitrary sequences. We show that left sequentiality is more general than strong sequentiality when applied to individual terms. For the theoretically restricted but important case of regular systems based on constructors, we prove two results of practical importance. First, left sequentiality is equivalent to strong sequentiality when considered as a property of whole sets of left-hand sides for systems of equations. Second, there is no algorithm which is safe for *all* systems based on a set of left-hand sides if that set is not left sequential, or, equivalently, not strongly sequential. In other words, left sequentiality is *necessary* for safe computation based on the analysis of left-hand sides alone. These results nicely complement the result of Huet–Levy (1979) that one can always construct a safe redex selection algorithm for any regular system based on a strongly sequential set of left-hand sides.

It is instructive to compare sequentiality with strictness analysis (Mycroft, 1980). Despite the differences in motivation and methods, the two are quite similar in meaning. Strictness analysis is applied to recursive equations and λ-expressions in order to detect whether a function is strict in some arguments, so that those arguments can be safely evaluated in call-by-value mode. The motivation is to save execution time, not to ensure safety: the usual call-by-need method is known to be safe. The analysis relies exclusively on the *right*-hand sides, since left-hand-side patterns are trivial. Strictness is usually applied when arguments belong to flat domains. Its meaning it not obvious when an argument belongs to a non-flat domain such as a function-space.

Sequentially can also be thought of as a kind of strictness in the sense that one is trying to identify for each function those of its arguments and parts of arguments that must be reducible to "head-normal form" for computation of the function to proceed. Strictness in this sense means that an application of a function is unsolvable if supplied with an unsolvable argument in a position for which the function is strict. This kind of strictness can be applied to non-flat domains, especially those for infinite data-structures such as steams. It is not of much use for call-by-value evaluation, since one is only looking for a head-normal form for the arguments, but it may be useful for exploiting parallelism in reduction machines such as the "four stroke reduction engine" (Clack and Peyton-Jones, 1986) which insist on ensuring that each parallel process is actually needed.

In reading the rest of this paper, familiarity with the work of Huet–Levy (1979) would be helpful, but the treatment is self-contained enough to be meaningful on its own.

2. Term Rewriting

The material in the initial part of this section is a review of the terminology and constructions commonly used in the literature on term rewriting systems. The latter part introduces some notation and terminology for the convenience of later sections.

A reduction system operates in a non-empty *ranked alphabet* Σ which contains all symbols in the system. **T** denotes the set of all *ground* Σ-terms for the alphabet. Σ-terms, in general, may contain *variables* drawn from a countable set **X**. A Σ-term is said to be *linear* iff no variable occurs more than once in it. We drop the prefix signifying the alphabet and, without confusion, speak of terms whenever possible.

A *path* p is a possibly empty string of integers. We say that p *reaches* subterm t/p in term t. The empty string Λ reaches the term itself, the string k reaches the kth argument, km reaches the mth argument of the kth argument, etc. Given paths p and q, $p \cdot q$ denotes their *concatenation.* The symbols $\leqslant$ and $<$ denote the *prefix* and *proper prefix* relations on paths respectively. Paths(t) denotes the set of all paths that reach some subterm in t. Paths(t) is partitioned into XPaths(t) and ΣPaths(t), where XPaths(t) is the subset that reaches variables in t, We often refer to a path $p \in$ Paths(t) as an *occurrence* of the subterm t/p in t. The expression $t[p \leftarrow w]$ denotes the term obtained by replacing t/p at p by w.

A *substitution* is a map from variables to terms. The meaning of a substitution can be extended naturally to a map from terms to terms. The application of a substitution β to a term t is conventionally denoted by $t\beta$, where $t\beta$ is the *instance* of t produced by *simultaneously* substituting $\beta(x)$ for every variable x in t. We use the notation $t \leqslant u$ to denote that u is an instance of t; $t < u$ means $t \leqslant u$ and $t \neq u$. If neither $t \leqslant u$ nor $u \leqslant t$ then t and u are said to be independent. The relation $\leqslant$ is clearly a partial order.

The usual first-order unification algorithm of Robinson (1965) is denoted by Unify. If two terms t and u have no common instance, then Unify(t, u) fails, otherwise it succeeds and returns a substitution β such that $t\beta = u\beta$ is the *least* common instance of t and u.

DEFINITION 2.1. A *base* $\mathscr{L}$ for a reduction system in the alphabet Σ is a finite set $\{l_i, 1 \leqslant i \leqslant m\}$ of linear Σ-terms such that:

(1) $\exists t \in \mathrm{T}$ such that $l_i \nleqslant t$, $1 \leqslant i \leqslant m$.

(2) If $l_1, l_2 \in \mathscr{L}$ and $p \in \Sigma$ Paths(l_1), then Unify(l_1/p, l_2) fails *unless* $p = \Lambda$ and $l_1 = l_2$.

Condition (1) states that there are normal forms, and (2) states that there are no "critical pairs" (Knuth and Bendix, 1970), which is used to ensure that any reduction system based on $\mathscr{L}$ will be confluent (Huet, 1977).

DEFINITION 2.2. A reduction system $\mathscr{S}$ in alphabet Σ is a finite set $\{\langle l_i, r_i \rangle, 1 \leqslant i \leqslant m\}$ of pairs of Σ-terms such that:

(1) $\mathscr{L} = \{l_i, 1 \leqslant i \leqslant m\}$ is a base.

(2) Each variable in r_i also appears in l_i, $1 \leqslant i \leqslant m$.

The definition just given defines the so-called *regular* systems used in (Hoffmann–O'Donnell, 1984) and (Huet–Levy, 1979). For the purposes of this paper we restrict ourselves to constructor systems, as defined in

DEFINITION 2.3. Suppose $\mathscr{L} = \{l_i, 1 \leqslant i \leqslant m\}$ in a base. Let $l_i = f_i(u_{i1}, ..., u_{in_i})$, $1 \leqslant i \leqslant m$, $\Delta_{\mathscr{L}} = \{f_i, 1 \leqslant i \leqslant m\}$, and $\Theta_{\mathscr{L}} = \Sigma - \Delta_{\mathscr{L}}$. $\Theta_{\mathscr{L}}$ is called the set of *constructors* of $\mathscr{L}$. A term $f(u_1, ..., u_m)$ is said to be a *$\Theta_{\mathscr{L}}$-pattern* iff $f \in \Delta_{\mathscr{L}}$ and each u_i is a $\Theta_{\mathscr{L}}$-term, $1 \leqslant i \leqslant m$. $\mathscr{L}$ is said to be a *constructor base* iff all $l_i \in \mathscr{L}$ are $\Theta_{\mathscr{L}}$-patterns. A system based on such an $\mathscr{L}$ is called a *constructor system*.

Throughout the following, we shall deal with a fixed set Σ of function symbols, a fixed constructor base $\mathscr{L}$, a fixed partition of Σ into Δ and Θ, and a fixed constructor system $\mathscr{S}$ based on $\mathscr{L}$, unless mentioned otherwise. In many definitions, the entities being defined are qualified with the subscript $\mathscr{L}$ or $\mathscr{S}$ signifying the context. These subscripts are dropped whenever the base or system concerned is the fixed one.

Any $u \in \mathbf{T}$ such that $u \geqslant l$, for some $l \in \mathscr{L}$, is called a *redex*. If t/p is a redex for some $p \in \Sigma$ Paths(t), then p is called a *redex occurrence* in t. The set of all redex occurrences in t is denoted by $\mathrm{RO}_{\mathscr{L}}(t)$. The set $\mathrm{NF}_{\mathscr{L}} \subseteq \mathbf{T}$ is the set of *normal forms*, and $t \in \mathrm{NF}_{\mathscr{L}}$ iff $\mathrm{RO}_{\mathscr{L}}(t) = \varnothing$.

A *simple reduction* $t \rightarrow^{s} u$ occurs in $\mathscr{S}$ iff $t/s = l\beta$ for some $l \in \mathscr{L}$, $\langle l, r\rangle \in \mathscr{S}$, and $u = t[s \leftarrow r\beta]$. We write $t \rightarrow u$ iff $t \rightarrow^{s} u$ for some $s \in \mathrm{RO}(t)$. We shall also need the rather unusual notion of an *arbitrary reduction*, denoted by $\rightsquigarrow$. We shall write $t \rightsquigarrow^{s} u$ iff $s \in \mathrm{RO}(t)$ and $u = t[s \leftarrow v]$ where v is an arbitrary ground term. Clearly, arbitrary reductions depend only on the *base*, not on the whole system. Finally, $t \rightarrow^{*} u$ and $t \rightsquigarrow^{*} u$ are the reflexive transitive closures of $\rightarrow$ and $\rightsquigarrow$, respectively. We use the notation $A\colon t \rightarrow u$ in order to attach a name (in this case A) to a simple reduction, and similarly also $A\colon t \rightsquigarrow u$, $B\colon t \rightarrow^{*} u$, and $B\colon t \rightsquigarrow^{*} u$. The symbol ε ambiguously denotes all empty reduction sequences. As with paths, we use the notation $A \cdot B$ to denote the concatenation of reduction sequences A and B. Such concatenation is meaningul only if the last term in A is identical to the first term in B.

A *redex selection* algorithm $\mathscr{A}$ is a function of a base $\mathscr{L}$ and a ground term t such that, $\mathscr{A}(\mathscr{L}, t)$ either fails or returns some $q \in \mathrm{RO}_{\mathscr{L}}(t)$. If $\mathscr{L}$ is fixed then the specialized or "curried" version of $\mathscr{A}$ is denoted by $\mathscr{A}_{\mathscr{L}}$. Given a selection algorithm $\mathscr{A}$, and $t \in \mathbf{T}$, the *evaluation sequence* for t produced by $\mathscr{A}$ is the reduction sequence $t_0 \rightarrow \cdots \rightarrow t_n \rightarrow \cdots$ such that:

(1) $t_0 = t$,

(2) if $\mathscr{A}_{\mathscr{L}}(t_n)$ fails the sequence *terminates* in t_n, else $\mathscr{A}_{\mathscr{L}}(t_n)$ returns r *and* $t_n \rightarrow^{r} t_{n+1}$.

The sequence is denoted by $\sigma_{\mathscr{A},\mathscr{S}}(t)$. If the sequence terminates in t_n for some n, then we write $\sigma_{\mathscr{A},\mathscr{S}}(t)\downarrow = t_n$, else we write $\sigma_{\mathscr{A},\mathscr{S}}(t)\uparrow$. If $t \rightarrow^{*} u$ in $\mathscr{S}$ and $u \in \mathrm{NF}_{\mathscr{L}}$, then t is said to *normalize to* u in $\mathscr{S}$. A selection algorithm $\mathscr{A}$ is said to be *safe* for a system $\mathscr{S}$ based on $\mathscr{L}$ iff given any $u \in \mathrm{NF}_{\mathscr{L}}$ and any t such that t normalize to u in $\mathscr{S}$, $\sigma_{\mathscr{A},\mathscr{S}}(t)\downarrow = u$.

The usual notion of residuals is meaningless in the context of arbitrary reductions, which is the only context in which we shall have occasion to use them. We therefore use the following simpler notion: suppose $A\colon t \rightsquigarrow^{r} u$ and $q \in$ Paths(t). The set $q \backslash A$, called the *preresiduals* of q after A, is defined as follows:

(1) if $r \nleq q$ then $q \backslash A = \{q\}$,

(2) otherwise $q \backslash A = \varnothing$.

Suppose $A\colon t \rightsquigarrow u$ and $Q \subseteq \text{Paths}(t)$. Then $Q \backslash A$ is $\bigcup_{q \in Q} q \backslash A$. This can be extended to sequences of reductions by composition, i.e., $Q \backslash \varepsilon = Q$, and $Q \backslash B \cdot C = (Q \backslash B) \backslash C$. If $q \in \text{Paths}(t)$, we use the notation $q \backslash B$ instead of $\{q\} \backslash B$.

The following Proposition is a succinct statement of the reason for excluding critical pairs in Definition 2.1.

PROPOSITION 2.4. *$q \in \text{RO}(t)$, $A\colon t \rightsquigarrow^* u$, implies that $q \backslash A \subseteq \text{RO}(u)$.*

Proof. Straightforward by condition (2) in Definition 2.1. ▌

3. VARIETIES OF SEQUENTIALITY

The fundamental concept of sequentiality is not very easy to define. The key idea is to exclude the possibility of using lookahead in choosing redices. Following Huet–Levy (1979), we introduce some purely technical auxiliary notions for this purpose.

A new constant symbol Ω will be added to the set Σ, and the augmented set of terms will be denoted by $\mathbf{T}_\Omega$. Members of $\mathbf{T}_\Omega$ will be called Ω-terms, an irreducible Ω-term will be called an Ω-normal form. Note that only irreducible terms in $\mathbf{T}$ are normal forms, thus reduction to normal form in subsequent definitions implies elimination of any Ω's that may be present in the original term. Intuitively, Ω corresponds to a part of an actual term about which one "knows" nothing; it is rather like the "least informative element" of Scott (1982). An information ordering on Ω-terms is defined by:

(1) $\Omega \sqsubseteq t$ for each $t \in \mathbf{T}_\Omega$.

(2) $f(t_1, ..., t_n) \sqsubseteq f(u_1, ..., u_n)$ iff each $t_i \sqsubseteq u_i$.

The following definition of *sequential predicate* is due to Kahn and Plotkin (1978).

DEFINITION 3.1. Let the truth values be ordered by false $\sqsubseteq$ true. Let P be a monotonic predicate on $\mathbf{T}_\Omega$. A path q in t is said to be an *index* of P in t iff

(I) $t/q = \Omega$,

(II) $u \sqsupseteq t$ and $P(u) = \text{true}$ implies that $u/q \neq \Omega$.

P is said to be *sequential at t* iff *one* of the following holds:

(1) there is an index of P in t,

(2) there is no $u \sqsupseteq t$ such that $P(u) = \text{true}$,

(3) $P(t) = \text{true}$.

P is sequential iff it is sequential at every Ω-term t.

The ordering on truth values and the insistence on monotonic predicates will not have much significance for us. Their appearance in the original definition is motivated by a desire to characterize the class of predicates one is interested in. Typical examples are predicates that answer positive questions such as "does this term have a normal form?" or "is this term a redex?" Increasing information about the given term (in the sense of the ordering $\sqsubseteq$) can only change the answer to such questions from false to true, never the other way; hence monotonicity and false $\sqsubseteq$ true. An index is precisely an occurrence of Ω that needs to be expanded in order to change the answer to a favorable one if at all possible. For example, suppose $\mathit{isredex}_{\mathscr{L}}(t) = \text{true}$ iff t is a redex with respect to $\mathscr{L}$, and $\mathscr{L} = \{f(a, x)\}$, $x \in \mathbf{X}$. The path 1 is an index of $\mathit{isredex}_{\mathscr{L}}$ in $t = f(\Omega, \Omega)$ because the occurrence of Ω at 1 must expand to a for t to become a redex. On the other hand, 2 is not an index because $f(a, \Omega) \sqsupseteq t$ is a redex.

Sequentially of a predicate is the possibility of systematically expanding any term step by step until either the predicate returns true or it is clear that a positive answer is impossible. In call-by-need evaluation, we really want the answer to the question "what is the normal form for this term?" This question does not have a true/false answer, but the question "does this term have a normal form?" is just as good for generating a redex selection algorithm. If one replaces all redices by Ω's, the resulting term does *not* have a normal form, and if one of the Ω's must be expanded in order to change the answer, then the corresponding redex is clearly "needed." Formally, suppose for any $t \in \mathbf{T}_\Omega$, $\mathrm{nf}_{\mathscr{S}}(t) = \text{true}$ iff $t \to^* u$ in $\mathscr{S}$ and u is in normal form. The system $\mathscr{S}$ is said to be *sequential* iff $\mathrm{nf}_{\mathscr{S}}$ is a sequential predicate. It is not hard to see that this kind of sequentiality is undecidable. Moreover, the corresponding indices cannot in general be found effectively even if a given system is known to be sequential. The more restrictive notion of *strong* sequentiality ignores the right-hand sides of equations, so it will be convenient to make it a property of a base. For any Ω-term t, let $\mathrm{snf}_{\mathscr{L}}(t) = \text{true}$ iff $t \rightsquigarrow^* u$ and u is in normal form. The base $\mathscr{L}$ is said to be *strongly sequential* iff $\mathrm{snf}_{\mathscr{L}}$ is sequential at every Ω-normal form. This restriction to Ω-normal forms is due to the fact that the following property of nf does not apply to snf (see Huet–Levy, 1979, pp. 26–27):

PROPOSITION 3.2. *The predicate* $\mathrm{nf}_{\mathscr{S}}$ *is sequential iff it is sequential at every Ω-normal form.*

The refinement of strong sequentiality we propose is embodied in the new predicate lnf, such that $\mathrm{lnf}_{\mathscr{L}}(t) = \text{true}$ iff $t \to^* u$ in *some* system based on $\mathscr{L}$, and u is in normal form. A path is an index of $\mathrm{lnf}_{\mathscr{L}}$ in t iff it is an index of $\mathrm{nf}_{\mathscr{S}}$ in t for *every* $\mathscr{S}$ based on $\mathscr{L}$. Such a path sometimes fails to qualify as an index of $\mathrm{snf}_{\mathscr{L}}$, as in the counterexample in Proposition 3.4. The new predicate shares the technical shortcoming of snf with respect to the property expressed in Proposition 3.2. For example, consider $\mathscr{L} = \{f(a, b, x, y)\}$, $x, y \in \mathbf{X}$, and $t = f(f(a, b, \Omega, \Omega), f(a, b, \Omega, \Omega), \Omega, \Omega)$. Although $\mathrm{lnf}_{\mathscr{L}}$ is sequential at every Ω-normal form corresponding to $\mathscr{L}$, it is not sequential at t. We therefore say that a base $\mathscr{L}$ is *left sequential* iff $\mathrm{lnf}_{\mathscr{L}}$ is sequential at every Ω-normal form. Despite this similarity, snf is a strictly weaker predicate than lnf, as the following propositions show.

PROPOSITION 3.3. $\forall t \in \mathbf{T}_{\Omega}$, $\mathrm{lnf}_{\mathscr{L}}(t) \Rightarrow \mathrm{snf}_{\mathscr{L}}(t)$; *but the converse is false.*

Proof. The implication is obvious from the respective definitions of the predicates. That the converse does not hold is shown by the following counterexample: suppose $\mathscr{L} = \{f(a, b, x)\}$, where $x \in \mathbf{X}$, and $t = f(f(a, b, \Omega), f(a, b, \Omega), \Omega)$, then $\mathrm{snf}_{\mathscr{L}}(t) = \text{true}$, but $\mathrm{lnf}_{\mathscr{L}}(t) = \text{false}$. ▮

PROPOSITION 3.4. *For all Ω-normal forms t, $\mathrm{lnf}_{\mathscr{L}}$ is sequential at t if* $\mathrm{snf}_{\mathscr{L}}$ *is sequential at t; but the converse is false.*

Proof. Note that for every Ω-term t, $\exists u \geqslant t$ such that $\mathrm{lnf}_{\mathscr{L}}(u) = \text{true}$, and the sequentiality of $\mathrm{lnf}_{\mathscr{L}}$ and $\mathrm{snf}_{\mathscr{L}}$ at t reduces to the presence of the respective indices in t. However, it follows directly from Proposition 3.3 that whenever $\mathrm{snf}_{\mathscr{L}}$ has an index in t so does $\mathrm{lnf}_{\mathscr{L}}$. That the converse does not hold is again shown by a counterexample: Let $\mathscr{L} = \{f(a, b, x), f(x, a, b), f(b, x, a), g(a)\}$, $x \in \mathbf{X}$, and $t = f(g(\Omega), g(\Omega), \Omega)$; then 3 is an index of $\mathrm{lnf}_{\mathscr{L}}$ in t, but $\mathrm{snf}_{\mathscr{L}}$ does not have an index in t. Note that this counterexample owes nothing to the fact that we restrict ourselves to constructor systems. ▮

COROLLARY 3.5. *Every strongly sequential base is also left sequential.*

The converse of Corollary 3.5 is far from obvious given the propositions preceding it. It turns out to be true because anomalies like the counterexample in Proposition 3.4 can only arise in the context of a base $\mathscr{L}$ which in fact is *not* left sequential. To show the equivalence of strong and left sequentiality, we will use yet another kind of sequentiality which we call *strict* sequentiality. Strict sequentiality is defined in terms of the existence of *necessary redex occurrences* rather than indices. Intuitively, a redex occurrence is necessary when it must be reduced in order to reach normal form. The following definition of necessity relies on an analysis of the syntactic structure of terms in $\mathscr{L}$ rather than on this intuitive property, for technical reasons. It is therefore more restrictive than the idea of a *strongly needed* redex in Huet–Levy (1979), which captures the intuitive property exactly. The difference is subtle and is similar to the distinction made by Huet–Levy (1979) between ordinary and increasing indices. Intuitively, the former may sometimes qualify on the basis of technical hair-splitting, while the latter are more "real" in the sense that they are the ones likely to be found and used by actual redex selection algorithms.

Given a ground term t, $u = \text{Known}(t)$ is the largest linear Θ-term or Θ-pattern such that $u \leqslant t$. Note that the set of all linear Θ-terms or patterns w such that $w \leqslant t$ is nonempty and finite, and hence contains a largest member by the properties of Unify. The variables in Known(t) are assumed to be new. For a ground term t, define the set $\mathscr{L}_t$ of equations compatible with t as $\mathscr{L}_t = \{l \in \mathscr{L} \mid \text{Unify}(l, \text{Known}(t)) \text{ succeeds}\}$. The significance of Known(t) is that it extracts the "known" information about the possibility that t is a redex. Safety requires that the normal form for a term be built from the outside inwards, lest a potentially unnecessary inner computation cause divergence. If it is clear that t can never become a redex then the outermost symbol(s) of t are already arranged as they would be in any eventual normal form and attention turns to the largest subterms that need to be reduced to normal form—the situation dealt with in case (3) of Definition 3.6. Otherwise a needed redex is one that will help settle the question whether t can become a redex. These considerations lead to the definition of a necessary redex occurrence given below.

Let $\text{PDNO}(t) = \text{XPaths}(\text{Known}(t))$. Each $p \in \text{PDNO}(t)$ is said to be *potentially directly necessary* for t. If, in addition, $p \in \Sigma\,\text{Paths}(l)$ for each $l \in \mathscr{L}_t$, then p is said to be *directly necessary* for t, denoted by $p \in \text{DNO}(t)$. The notion of necessary redex occurrences essentially iterates diectly necessary occurrences until a redex is reached.

DEFINITION 3.6. $p \in \text{RO}(t)$ is said to be *necessary* for t, written as $p \in \text{NRO}(t)$, iff *one* of the following holds:

(1) $p = \Lambda$ or $p \in \text{DNO}(t)$,

(2) $q \in \text{DNO}(t)$, $p = q \cdot r$, and $r \in \text{NRO}(t/q)$,

(3) $\exists q \in \text{DNO}(t)$ such that $\text{RO}(t/q) = \varnothing$, $r \in \text{PDNO}(t)$, $p = r \cdot s$, and $s \in \text{NRO}(t/r)$.

$\mathscr{L}$ is said to be *strictly sequential* iff whenever RO(t) is nonempty, NRO(t) is also nonempty.

Example 1 illustrates these definitions. The set NRO(t) is quite similar to the set $\mathscr{I}(\omega'(t))$ of "increasing indices" defined by Huet–Levy (1979), where $\omega'(t)$ is obtained from t by replacing all redices with Ω's.

EXAMPLE 1.

$$\mathscr{L} = \{f(x, a, b), f(b, x, a), f(a, b, x)\}$$
$$t = f(a, f(b, a, f(a, a, b)), f(b, b, a))$$
$$u = t/2 = f(b, a, f(a, a, b))$$

$\text{Known}(t) = f(a, y1, y2)$	$\text{PDNO}(t) = \{2, 3\}$
$\text{DNO}(t) = \{2\}$	$\text{RO}(t) = \{2 \cdot 3, 3\}$
$\text{Known}(u) = f(b, a, y3)$	$\text{PNDO}(u) = \{3\}$
$\text{DNO}(u) = \{3\}$	$\text{RO}(u) = \{3\}$
$\text{NRO}(u) = \{3\}$	$\text{NRO}(t) = \{2 \cdot 3\}$

4. MAIN RESULTS

There are two main results. First, in spite of the differences noted above, strong and left sequentiality coincide when considered as properties of bases. Second, the left sequentiality of a base $\mathscr{L}$ is both necessary and sufficient for the existence of a redex selection algorithm that is safe for all systems based on $\mathscr{L}$.

Strong sequentiality has already been shown to imply left sequentiality (Corollary 3.5). Strict sequentiality turns out to be the glue that holds the rest of the argument together, which is not surprising, since this variety was formulated to capture the intuitive *operational* requrements for call-by-need computation. First, we exhibit and verify a decision procedure Check for strict sequentiality. It is then shown that a base rejected by Check cannot be left sequential, i.e., left sequentiality implies strict sequentiality. We shall also show that in most cases, there is no safe redex selection algorithm for a base rejected by Check, thus strict sequentiality is necessary for safe sequential computation. Finally, we show that whenever the redices in a reducible term are replaced by Ω's, a necessary redex occurrence always becomes an index of snf in the resulting Ω-normal form, and therefore strict sequentiality implies strong sequentiality. This completes the circle of implications proving the three varieties of sequentiality equivalent.

4.1. *A Decision Procedure for Strict Sequentiality*

As the structure of Check below indicates, strict sequentiality is a property of the group of equations defining each individual function. In fact, fcheck is simply an abstract version of a translator that transforms each such group into a *single* equation involving nested conditionals, selectors, and discriminators. The path p chosen in each call of fcheck corresponds to the argument or part of argument whose structure needs to be queried next, given that the structure of the application of f discovered so far corresponds to z. The argument z plays no part in the decision procedure. Its presence is solely a device to facilitate statements and proofs of the properties of Check. An example of a base rejected by Check is found in Example 2 later in this section.

Check is actually a decision procedure for the (match) *sequentiality* of $\mathscr{L}$ in the sense of Huet–Levy (1979). Intuitively, a base $\mathscr{L}$ is match sequential iff the process of determining whether a given term is a redex with respect to $\mathscr{L}$ can be carried out in a sequential manner. Match sequentiality is a stronger (more restrictive) notion than strong sequentiality in the general case of regular systems, although it coincides with the latter in the case of constructor systems as we shall show. Check can also be thought of as a procedure for building a "matching dag" for the given base, again in the sense of Huet–Levy (1979).

Decision Procedure Check. The following algorithm uses the notation that for each $f \in \Sigma$, with arity k,

$$z^f = f(x_1, \ldots, x_k) \qquad \text{all } x_i \in \mathbf{X} \text{ are new,}$$

$$\mathscr{L}^f = \{l \in \mathscr{L} \mid z^f \leqslant l\} \qquad \text{and} \qquad P^f = \{1, \ldots, k\}.$$

```
Function Check($\mathscr{L}$)
  Let $s_f = \text{fcheck}(\mathscr{L}^f, P^f, z^f)$
  Return the conjunction of $s_f$, $f \in \Theta_{\mathscr{L}}$
Function fcheck($L, P, z$)
  **If** $|L| \leqslant 1$ **Then** Return True **Else** Let $Q = \{q \in P \mid q \notin \mathbf{X}\text{Paths}(l), l \in L\}$
  If $Q = \varnothing$ Then Return False Else choose any $p \in Q$
  partition $L$ into $L_c$, $c \in \Theta$, where $l \in L_c$ iff $l/p \geqslant c(x_1, \ldots, x_k)$, $k \geqslant 0$, and
  each $x_i$ is a new variable. Correspondingly, for each $c$, $P_c = P - \{p\} \cup$
  $\{p \cdot j, 1 \leqslant j \leqslant k\}$, $z_c = z[p \leftarrow c(y_1, \ldots, y_k)]$ where each $y_i$ is a new variable
  Return the conjunction of fcheck($L_c, P_c, z_c$), $c \in \Theta$
```

First we prove a few simple technical facts about Check. A call fcheck(L, P, z) is said to be *legitimate* iff it results from the call Check($\mathscr{L}$).

PROPOSITION 4.1.1. In any legitimate call fcheck(L, P, z) the following statements hold:

(1) $P \subseteq \text{Paths}(l)$ for each $l \in L$,

(2) $z \leqslant l$ for every $l \in L$,

(3) for each $l \in \mathscr{L} - L$, Unify(l, z) fails,

(4) $P = \mathbf{X}\text{Paths}(z)$,

(5) $|L| > 1$ implies $P \neq \varnothing$,

(6) if fcheck(L', P', z') is also a legitimate call, then either Unify(z, z') fails, or z and z' are not independent.

Proof. Assertion (5) follow from (2) and (4). The proofs of the rest proceed by induction on the length of the calling chain resulting in the call fcheck(L, P, z).

Basis. The length is 1, and the call is fcheck$(\mathscr{L}^f, P^f, z^f)$ for some f. All assertions except (6) are obviously true. For (6), it is sufficient to note that if a call fcheck(L_1, P_1, z_1) results eventually from a call fcheck(L_2, P_2, z_2), then $z_1 > z_2$, and also that Unify(z^f, z^g) fails whenever $f \neq g$.

Induction. The inductive assumption is that the assertions hold for all calls with calling chains of length $n \geqslant 1$. Suppose the call fcheck(L, P, z) has a chain of length $n+1$. Clearly, the call results immediately from another call with chain length n, and all assertions hold for the latter by the inductive assumption. The inductive step is now straightforward for all assertions except (6). For (6), the same observations suffice as with those in the basis case. ▮

THEOREM 4.1.2. Check$(\mathscr{L})$ *terminates.*

Proof. Let $zlength(z) = \sum_{p \in \Sigma\text{ Paths}(z)} length(p)$. By (2) in Proposition 4.1.1, $zlength(z)$ has a finite upper bound in any legitimate call fcheck(L, P, z) unless $L = \varnothing$. Moreover, $zlength(z_c) > zlength(z)$ by (4) in Proposition 4.1.1. Hence, each legitimate call fcheck(L, P, z) terminates, either because $L = \varnothing$ or because each chain of recursive calls issuing from it is bounded by the limit on the monotonic increasing quantity $zlength(z)$. ▮

The reason Check guarantees strict sequentiality is that, whenever we have a term t that is neither irreducible nor a redex, the path p chosen in the call fcheck(L, P, z) with the largest z compatible with t reaches a directly necessary subterm of t. This is the intuitive idea in

PROPOSITION 4.1.3. Check$(\mathscr{L})$, $t \notin \mathrm{NF}_{\mathscr{L}}$, *and* $\Lambda \notin \mathrm{RO}(t)$ *implies* $\mathrm{DNO}(t) \neq \varnothing$.

Proof. Suppose the antecedents hold. If $\mathscr{L}_t = \varnothing$, then $\mathrm{PDNO}(t) = \mathrm{DNO}(t)$, and it is easy to see that in that case, the antecedents imply $\mathrm{DNO}(t) \neq \varnothing$. Suppose $\mathscr{L}_t \neq \varnothing$. Let Z be the set of Θ-patterns u such that $u \leqslant t$ and fcheck(M, R, u) is a legitimate call for some M and R. Since $\mathscr{L}_t \neq \varnothing$, Z is clearly nonempty, and also finite. By (6) in Proposition 4.1.1, Z is linearly ordered, and hence contains a maximal element z, which occurs in some call fcheck(L, P, z). Let $Q = \{q \in P \mid q \notin \mathrm{XPaths}(l)$ for any

$l \in L\}$. By (3) in Proposition 4.1.1, $L \supseteq \mathscr{L}_t$. By the maximality of z, $Q \subseteq \mathbf{X}\text{Paths}(\text{Known}(t)) = \text{PDNO}(t)$. By (1) in Proposition 4.1.1, and the above considerations, we have $Q \subseteq \text{DNO}(t)$. It remains to show that $Q \neq \varnothing$. If $|L| > 1$ then since Check($\mathscr{L}$) we have $Q \neq \varnothing$. Since $\mathscr{L}_t \neq \varnothing$, the only other possibility is that $L = \{l\}$. Since $\Lambda \notin \text{RO}(t)$, $l \not\leqslant \text{Known}(t)$. Since $P \subseteq \text{Paths}(l)$ and $P = \mathbf{X}\text{Paths}(z)$ by Proposition 4.1.1, if $Q = \varnothing$ then the only way $z \leqslant l$ is if $z = l$. However, by (2) in Proposition 4.1.1, $z \leqslant l$, hence $Q = \varnothing$ implies $z = l$ and $l \leqslant t$, contradicting the assumption that $\Lambda \notin \text{RO}(t)$. ▮

It only remains to iterate this fact by structural induction.

THEOREM 4.1.4. Check($\mathscr{L}$) *implies* $\mathscr{L}$ *is strictly sequential.*

Proof. We must show that whenever RO(t) is nonempty, NRO(t) is also nonempty. The proof proceeds by induction on the structure of t.

Basis. $t \in \Delta$, where $\text{RO}(t) = \text{NRO}(t) = \{\Lambda\}$.

Induction. The inductive assumption is that the required property holds for all proper subterms of t. If t is a redex then $\Lambda \in \text{NRO}(t)$. Suppose it is not, and $r \in \text{RO}(t)$. By Proposition 4.1.3 above we have $\text{DNO}(t) \neq \varnothing$. Suppose $p \in \text{DNO}(t)$. If $\text{RO}(t/p) \neq \varnothing$ then by the inductive assumption $\exists q \in \text{NRO}(t/p)$, and $p \cdot q \in \text{NRO}(t)$. If $\text{RO}(t/p) = \varnothing$ then let $s \in \text{Paths}(t)$ be the nonempty path such that $r = s \cdot w$ and $s \in \text{PDNO}(t)$. By the inductive assumption $\text{NRO}(t/s) \neq \varnothing$, and therefore by (3) in Definition 3.6, $\text{NRO}(t) \neq \varnothing$. ▮

We show that Check is a *complete* decision procedure by defining a function "Strange" which constructs a counterexample to prove that $\mathscr{L}$ is not strictly sequential whenever Check($\mathscr{L}$) fails. Definition 4.1.5 may seem rather too complex for the present purpose, but it turns out to be essential in the construction of the main counterexamples in the next section.

DEFINITION 4.1.5. If a legitimate call fcheck(L, P, z) *directly* returns false, being unable to find a suitable $p \in P$, then, given a function α as an extra parameter, Strange(L, P, z, α) is a term defined as follows. Let $\mathscr{L} = \{l_0, ..., l_m\}$, $m \geqslant 1$. Partition P into $P_0, ..., P_m$, such that $P_i \subseteq \mathbf{X}\text{Paths}(l_i)$. This is possible by the failure condition in fcheck. Let $V_i = \{z/p \mid p \in P_i\}$. By (4) in Proposition 4.1.1, each $V_i \subseteq \mathbf{X}$. Define

$$\text{Strange}(L, P, z, \alpha) \overset{\text{def}}{=} z\beta, \quad \text{where} \quad \beta(x) = \mathbf{If} \quad x \in V_i \quad \mathbf{Then}\ \alpha(x, i) \quad \mathbf{Else}\ x.$$

Note that β is well defined since V_i are disjoint by the construction of z.

EXAMPLE 2. Let $\mathscr{L}$ be as in Example 1. Clearly, fcheck($\mathscr{L}^f$, P^f, z^f) fails directly, and thus Strange($\mathscr{L}^f$, P^f, z^f, α) is well defined. We have,

$P = P^f = \{1, 2, 3\}$, $z = z^f = f(y_0, y_1, y_2)$
$l_0 = f(x, a, b)$, $l_1 = f(b, x, a)$, $l_2 = f(a, b, x)$
Therefore, $P_0 = \{1\}$, $P_1 = \{2\}$, $P_2 = \{3\}$
$V_0 = \{y_0\}$, $V_1 = \{y_1\}$, $V_2 = \{y_2\}$
With $\alpha(y_0, 0) = f(b, a, a)$, $\alpha(y_1, 1) = f(a, b, a)$, and $\alpha(y_2, 2) = f(a, a, b)$,
Strange(L, P, z, α) $= f(f(b, a, a), f(a, b, a), f(a, a, b))$.

PROPOSITION 4.1.6. *Whenever α returns a redex for all arguments, as in Example* 2, *and $w =$* Strange(L, P, z, α) *is well defined,*

(1) *w is a ground term.*
(2) $P = \mathrm{RO}(w) = \varnothing$.
(3) $\mathrm{DNO}(w) = \varnothing$.
(4) $\mathrm{NRO}(w) = \varnothing$.
(5) *At least two of the partitions (P_i) of P are nonempty, independent of α.*

Proof. (1) is a consequence of (4) in Proposition 4.1.1. The $P = \mathrm{RO}(w)$ part of (2) merely asserts that $\Lambda \notin \mathrm{RO}(w)$. To see this, note that since w is well defined $|L| > 1$. Moreover, $\Lambda \in \mathrm{RO}(w)$ would mean $\exists l \in L$ such that $l \leqslant z$, and by (2) in Proposition 4.1.1, $l = z$. This would imply, again by (2) in Proposition 4.1.1, that L is not an independent set, contradicting (2) in Definition 2.1. Assertion (5) also follows similarly since $P \subseteq \mathrm{XPaths}(l_i)$ would imply $z = l_i$. $P \neq \varnothing$ follows from (5) in Proposition 4.1.1 since $|L| > 1$. By the construction of w, Known(w) $= z$, hence by (2) and (3) in Proposition 4.1.1, $L = \mathscr{L}_w$. We know that each $q \in \mathrm{PDNO}(w) = \mathrm{XPaths}(z)$ reaches a variable in some $l \in L = \mathscr{L}_w$ by the construction of w, hence $\mathrm{DNO}(w) = \varnothing$. Since $\mathrm{PDNO}(w) = \mathrm{XPaths}(z) = P = \mathrm{RO}(w)$ by (4) in Proposition 4.1.1 and (2) above, $\mathrm{NRO}(w) = \varnothing$ is obvious. ∎

THEOREM 4.1.7. Check($\mathscr{L}$) = false *implies that $\mathscr{L}$ is not strictly sequential.*

Proof. Check($\mathscr{L}$) = false implies that some legitimate call fcheck(L, P, z) fails directly, and therefore $w =$ Strange(L, P, z, α) is well defined. Assertions (2) and (4) in Proposition 4.1.6 then imply that $\mathscr{L}$ is not strictly sequential. ∎

4.2. *The Necessity of Strict Sequentiality*

We now show that if the decision procedure for strict sequentiality rejects a base, that base cannot be left sequential. This effectively means that strict sequentiality is necessary for call-by-need computation *without look-ahead*. Another way to state the same idea would be to say that if a base $\mathscr{L}$ is not strictly sequential, then no algorithm that selects a redex based only on its context is safe for all systems based on $\mathscr{L}$. The function Strange is again used in constructing the necessary counter example.

THEOREM 4.2.1. *If $\mathscr{L}$ is not strictly sequential then it is not left sequential.*

Proof. We exhibit an Ω-normal form t such that $t \notin \mathrm{T}$ and $\mathrm{Inf}_{\mathscr{L}}$ has no index in t. Since $\mathrm{Check}(\mathscr{L}) = \mathrm{false}$, $t = \mathrm{Strange}(L, P, z, \alpha)$ is well defined for some $\mathscr{L} = \{l_0, ..., l_m\}$, $P = P_0 \cup \cdots \cup P_m$, and z, where $m \geqslant 1$, and α returns Ω for all arguments. Any potential index of $\mathrm{Inf}_{\mathscr{L}}$ in t must be in P. Suppose for the sake of contradiction that q is an index. By the construction of Strange, $q \in P_i \subseteq \mathrm{XPaths}(l_i)$. Clearly, given (5) in Proposition 4.1.6, there is a $u \sqsupseteq t$ in which all occurrences of Ω except q are replaced by suitable redices, and the right-hand sides of the equations (which are completely unconstrained) are arranged in such a way that u becomes an instance of l_i after some reductions, and is normalized thereafter. The occurrence q is therefore not an index contrary to the assumption. ▌

This result can be stated more strongly for sufficiently large bases, for which the (perhaps unrealistic) requirement that the redex selection algorithm be unaware of the "appearance" of the redices is unnecessary.

THEOREM 4.2.2. *If $\mathscr{L}$ is not strictly sequential then there is no redex selection algorithm that is safe for all systems based on $\mathscr{L}$ if $\mathscr{L}$ is sufficiently large as a set.*

Proof. Suppose $\mathscr{A}$ is safe for all $\mathscr{S}$ based on $\mathscr{L}$ but $\mathscr{L}$ is not strictly sequential, therefore $\mathrm{fcheck}(L, P, z)$ fails directly for some L, P, z. Suppose $\alpha(x, j)$ returns an instance of a distinct $l \in \mathscr{L}$ (not necessariy in L) for each variable x that occurs in z, ensuring only that $l \neq l_j$. This is always possible for a sufficiently large $\mathscr{L}$. Clearly, $w = \mathrm{Srange}(L, P, z, \alpha)$ can be normalized in some system based on $\mathscr{L}$, hence $\mathscr{A}(\mathscr{L}, w)$ must succeed and return some path $q \in P$. Suppose $q \in P_i$. We can construct a system $\mathscr{S}$ based on $\mathscr{L}$ for which $\mathscr{A}$ is not safe. Suppose $\alpha(z/q, i)$ is an instance of some $l \in \mathscr{L}$. Let $\langle l, l \rangle \in \mathscr{S}$. Since t/q is an instance of l, $\sigma_{\mathscr{A}, \mathscr{S}}(w)\uparrow$. By the construction of w, $q \in \mathrm{XPaths}(l_i)$. Clealry, w can be made an instance of l_i after some reductions by suitably "filling in" $\mathscr{S}$, as in Theorem 4.2.1, since all the redexes that matter are instances of a left-hand side other than l, and their right-hand sides are therefore unconstrained. Thus, w normalizes to some normal form r in $\mathscr{S}$, and for $\mathscr{A}$ to be safe for $\mathscr{S}$, we must have $\sigma_{\mathscr{A}, \mathscr{S}}(w)\downarrow = r$, which is false. ▌

Example 3 below illustrates the construction of the counterexample in the proof of Theorem 4.2.1. The same example can be used to illustrate Theorem 4.2.2 with $w = f(f(b, a, a),\ f(a, b, a),\ f(a, a, b))$. Examples 4 illustrates the kind of pathological situation which makes it necessary to restrict Theorem 4.2.2 to sufficiently large bases. Note that the construction of w in Example 4 is the same as in the proof of Theorem 4.2.2, except that t_2 and t_3 are instances of the same left-hand side since there are not enough left-hand sides to go around. Now suppose $\mathscr{A}(\mathscr{L}, w)$ returns 1. We must turn w into an instance of l_1 in order to defeat $\mathscr{A}$. We therefore need to reduce t_2 and t_3 to a and b, respectively. But they are both instances of l_1 which cannot be given the right-hand side x, since this would cause w to be reduced to an unsolvable term. In the absence of this choice, we cannot reduce t_2 and t_3 to distinct normal forms. It is not clear how to get around such cases so as to eliminate the practically inconsequential but theoretically annoying caveat in the statement of Theorem 4.2.2.

EXAMPLE 3. Let L, P, z be as in Example 2. Let $t = f(\Omega, \Omega, \Omega)$. Suppose 1 is an index of $\mathrm{Inf}_{\mathscr{S}}$ in t. Consider $u = f(\Omega, f(a, b, a), f(a, a, b)) \sqsupseteq t$. One counterexample $\mathscr{S}$ is

$$f(x, a, b) = b, f(b, x, a) = f(b, x, a), f(a, b, x) = a.$$

The other candidate indices can similarly be shown to fail.

EXAMPLE 4. Let $l_1 = f(x, a, b)$, $l_2 = f(a, x, y)$, and $\mathscr{L} = \{l_1, l_2\}$. fcheck($\mathscr{L}$, $\{1, 2, 3\}$, z) fails for $z = f(x_0, x_1, x_2)$. Suppose $t_1 \geqslant l_2$, and $t_2, t_3 \geqslant l_1$, for some redices t_1, t_2, t_3. Consider $w =$ Strange($\mathscr{L}$, $\{1, 2, 3\}$, z, α) $= f(t_1, t_2, t_3)$.

4.3. *The Sufficiency of Strict Sequentiality*

This section shows that strict sequentiality is sufficient to ensure strong sequentiality, which in turn is known to be sufficient to ensure the existence of safe evaluation algorithms (Huet–Levy, 1979). As we mentioned in Subsection 4.1, this really amounts to proving that, in the case of constructor systems, match sequentiality implies strong sequentiality. This result could be indirectly inferred from the relationship between Check and the algorithm for constructing matching dags, together with the decidability theorem for strong sequentiality (Huet–Levy, 1979). But the details are sufficiently involved and inaccessible in the original, and of sufficient interest, to justify inclusion of the following simple and direct treatment of the simpler case of constructor systems for completeness. The proof hinges on two properties of necessary redex occurrences, viz., a necessary redex is always an outermost redex, and necessary redex occurrences persist as such until they are reduced.

PROPOSITION 4.3.1. *$\Lambda \in \mathrm{RO}(t)$ implies* $\mathrm{DNO}(t) = \varnothing$.

Proof. $\Lambda \in \mathrm{RO}(t)$ implies $\mathrm{Known}(t) \geqslant l$ for some $l \in \mathscr{L}_t$, therefore $\mathrm{PDNO}(t) \cap \Sigma\ \mathrm{Paths}(l)$ is clearly empty. ▌

COROLLARY 4.3.2. *$\Lambda \in \mathrm{RO}(t)$ implies* $\mathrm{NRO}(t) = \{\Lambda\}$.

PROPOSITION 4.3.3. *If $p \in \mathrm{NRO}(t)$ and $s \in \mathrm{RO}(t)$ then $s \not< p$.*

Proof. By induction on the structure of t. The basis case is that $t \in \Sigma$, which is trivial. For the inductive step, assume that the proposition holds for all proper subterms of t. There are three cases, according to Definition 3.6.

Case 1. $p = \Lambda$ or $p \in \mathrm{DNO}(t)$. Recall that if $p \in \mathrm{DNO}(t)$ then by Proposition 4.3.1, $\Lambda \notin \mathrm{RO}(t)$, hence $s \neq \Lambda$. The rest follows from the fact that $p \in \mathrm{PDNO}(t)$.

Case 2. $q \in \mathrm{DNO}(t)$, $p = q \cdot r$, and $r \in \mathrm{NRO}(t/q)$. We have $\Lambda \notin \mathrm{RO}(t)$ as before. By Corollary 4.3.2, if $q \in \mathrm{RO}(t)$ then $p = q$. Therefore, $s \neq \Lambda$ and $s \neq q$. The rest follows by the inductive assumption.

Case 3. Similar to Case 2. ▌

PROPOSITION 4.3.4. *Suppose $p \in \mathrm{DNO}(t)$, $A: t \rightsquigarrow^{r} u$, and $p \neq r$, then $p \backslash A = \{p\}$, and $p \in \mathrm{DNO}(u)$.*

Proof. Since $\mathrm{DNO}(t) \neq \varnothing$, $r \neq \Lambda$ by Proposition 4.3.1. By the antecedents and the definition of DNO, $r \not\leq p$, hence $p \backslash A = \{p\}$. Moreover, since $r \neq \Lambda$, $\mathscr{L}_u \subseteq \mathscr{L}_t$, and $p \in \mathrm{DNO}(u)$. ▌

PROPOSITION 4.3.5. *$p \in \mathrm{NRO}(t)$, $A: t \rightsquigarrow^{s} u$, and $p \neq s$ implies that $p \backslash A = \{p\}$ and $p \in \mathrm{NRO}(u)$.*

Proof. By Proposition 4.3.3, $s \not\leq p$; therefore $p \backslash A \{p\}$, and moreover, $s \neq \Lambda$. $p \in \mathrm{RO}(u)$ by Proposition 2.4. $p \in \mathrm{NRO}(u)$ is then straightforward by analysing the cases of Definition 3.6, using Proposition 4.3.4. ▌

A sequence $B: t \rightsquigarrow^{*} u$ is said to *preserve p* iff $p \backslash B = \{p\}$.

LEMMA 4.3.6. *Suppose $A: t \rightsquigarrow^{*} u$ and $r \in \mathrm{NRO}(t)$. Then either A preserves r and $r \in \mathrm{NRO}(u)$, or $A = A_1 \cdot A_2 \cdot A_3$, where $A_1: t \rightsquigarrow^{*} v$ preserves r, $A_2: v \rightsquigarrow^{r} w$, and $A_3: w \rightsquigarrow^{*} u$.*

Proof. By induction on $|A|$. The basis $|A| = 0$ is trivial. Suppose $|A| = n + 1$. Let $A = A_1 \cdot A_2$, where $A_1: t \rightsquigarrow^{s} t_1$ is the first reduction in the sequence. By Proposition 4.3.5, either $r = s$, or $r \backslash A_1 = \{r\}$ and $r \in \mathrm{NRO}(t_1)$. The rest follows by the inductive assumption. ▌

COROLLARY 4.3.7. *If* $p \in \mathrm{NRO}(t)$, $u = t[p \leftarrow \Omega]$, *and* $A\colon u \leadsto^* w$, *then* A *preserves* p.

Following Huet–Levy (1979), we use the expression $\omega'(t)$ to denote the Ω-normal form obtained by replacing each redex in t by Ω.

LEMMA 4.3.8. $\forall t \in \mathbf{T}$, *every necessary redex occurrence in* t *is an index of* snf *in* $\omega'(t)$.

Proof. Suppose for the sake of contradiction that $p \in \mathrm{NRO}(t)$, but p is not an index of snf in $\omega'(t)$. Then there is $w \geqslant t$ such that $w/p = \Omega$ and $B\colon w \leadsto^* r$, where r is in normal form. Consider $u = w[p \leftarrow t/p]$. We can assume without loss of generality that $u \in \mathbf{T}$. Clearly, there is a sequence $A\colon t \leadsto^* u$ such that A preserves p, and therefore, by Lemma 4.3.6, $p \in \mathrm{NRO}(u)$. Therefore, by Corollary 4.3.7, $B\colon w \leadsto^* r$ preserves p, contrary to the assumption that r is in normal form. ▮

THEOREM 4.3.9. *A base* $\mathscr{L}$ *is strongly sequential if it is strictly sequential.*

Proof. Corresponding to every Ω-normal form z there is a $t \in \mathbf{T}$ such that $z = \omega'(t)$. We are only concerned with the case where $z \notin \mathrm{NF}_{\mathscr{L}}$, hence $\mathrm{RO}(t) \neq \varnothing$. Therefore, given that $\mathscr{L}$ is strictly sequential, $\exists p \in \mathrm{NRO}(t)$, and p is an index of snf in $z = \omega'(t)$ by Lemma 4.3.8. ▮

5. CONCLUSIONS

Theorems 4.2.1 and 4.3.9, and Corollary 3.5 together show the equivalence of strict, strong, and left sequentiality as properties of bases. The necessity of left sequentiality in order to guarantee the existence of safe redex selection algorithms then follows Theorem 4.2.2. The sufficiency of left sequentiality can be inferred from the results of Huet–Levy (1979) regarding strong sequentiality. A much simpler and more direct proof of the sufficiency of strict sequentiality for the existence of such an algorithm in the case of constructor systems can be found in (Thatte, 1984).

The results indicate that left sequentiality is an *exact* characterization of the property of "admissibility for call-by-need computation" when applied to linear unambiguous bases. It is more refined and intuitively more satisfactory than strong sequentiality for theoretical purposes, but is equivalent when used in the analysis of actual programs. Using the simulation technique of Thatte (1985), it is easy to show that these statements apply not only to constructor bases but to the larger subclass of regular bases corresponding to what Huet–Levy (1979) call *simple systems.* It is less obvious that our results apply to the full class of regular systems. However, the crucial construction of the Strange function could perhaps be extended to all regular systems using a fully constructive account, analogous to Check, of Huet–Levy's algorithm for constructing matching dags. The outline of such an account is given in Section 4.8 of Huet–Levy (1979).

ACKNOWLEDGMENTS

I would like to thank the referees of this and a previous version of the paper for comments that led to great improvement in the presentation and content.

RECEIVED July 24, 1985, ACCEPTED July 9, 1986

REFERENCES

CLACK, C., AND PEYTON–JONES, S. L. (1986), The four stroke reduction engine, *in* "Proc. 1986 ACM Conf. on LISP and Functional Programming," Cambridge, Mass.

HENDERSON, P., AND MORRIS, J. M. (1976), A lazy evaluator, *in* "Proc. 3rd ACM Symp. on the Principles of Programming Laguages," Atlanta.

HOFFMANN, C. M., AND O'DONNELL, M. J. (1979), An interpreter generator using tree pattern matching, *in* "Proc. 5th ACM Symp. on the Principles of Programming Languages," San Antonio.

HOFFMANN, C. M., AND O'DONNELL, M. J. (1984), Implementation of an interpreter for abstract equations, *in* "Proc. 11th ACM Symp. on the Principles of Programming Languages," Salt Lake City.

HUET, G. (1977), Confluent reductions: Abstract properties and applications to term Rewriting systems, *in* "Proc. 18th IEEE Conf. on Foundations of Computer Science," Providence, R.I.

HUET, G., AND LEVY, J.-J. (1979), "Computations in Nonambiguous Linear Term Rewriting Systems," Tech. Rep. No. 359, INRIA, Le Chesney, France.

KAHN, G., AND MACQUEEN, D. B. (1977), Coroutines and networks of parallel processes, *in* "IFIP 77" (B. Gilchrist, Ed.), North-Holland, Amsterdam.

KAHN, G. AND PLOTKIN, G. (1978), "Domaines Concretes," Rapport IRIA-LABORIA, No. 336.

KNUTH, D. E., AND BENDIX, P. B. (1970), Simple word problems in universal algebras, *in* "Computational Problems in Abstract Algebra" (J. Leech, Ed.), Pergammon, Elmsford, N.Y.

MYCROFT, A. (1980), "The Theory and Practice of Transforming Call-by-need into Call-by-value," Proc. of Int. Symp. on Programming, Lecture Notes in Computer Science, No. **83**, pp. 269–281, Springer-Verlag, Berlin.

ROBINSON, J. A. (1965), A machine-oriented logic based on the resolution principle, *J. Assoc. Comput. Mach.* **12**, 23–41.

SCOTT, D. (1982), "Domains for Denotational Semantics," Lecture Notes in Computer Science, No. 140, Springer-Verlag, Berlin.

THATTE, S. R. (1984), "Demand Driven Evaluation with Equations," Tech. Rep. CRL-TR-34-84, EECS Department, University of Michigan, Ann Arbor.

THATTE, S. R. (1985), On the correspondence between two classes of reduction systems, *Inform. Process. Lett.* **20** (2), 83–85.

TURNER, D. A. (1979), "A New Implementation Technique for Applicative Languages," Software Practice and Experience, No. 9.

VUILLEMIN, J. (1974), Correct and optimal implementations of recursion in a simple programming language, *J. Comput. Systems Sci.* **9**, 332–354.

Wadsworth, C. (1971), "The Semantics and Pragmatics of the Lambda Calculus," Ph.D. dissertation, Oxford University, England.