

Research Report

COMPLETENESS OF REWRITE RULES AND REWRITE STRATEGIES FOR FP

Joseph Y. Halpern
John H. Williams
Edward L. Wimmers

IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).



Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

Copies may be requested from:
IBM Thomas J. Watson Research Center
Distribution Services
Post Office Box 218
Yorktown Heights, New York 10598

COMPLETENESS OF REWRITE RULES AND REWRITE STRATEGIES FOR FP

Joseph Y. Halpern
John H. Williams
Edward L. Wimmers

IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099

Abstract: We consider languages whose operational semantics is given by a set of rewrite rules. For such languages, it is important to be able to determine that there are enough rules to be able to compute the correct meaning of all expressions, but not so many that the system of rules is inconsistent. We develop a formal framework in which to give a precise treatment of these completeness and soundness issues, and then investigate them in the context of an extended version of the functional programming language FP. We show that the rewrite rules of FP are sound and complete with respect to three different notions of completeness. In the latter half of the paper we turn our attention to rewrite strategies. In order to implement a language based on rewrite rules, it does not suffice to know that there are "enough" rules in the language; we also need to have a good strategy for determining the order in which to apply them. But what is "good"? Corresponding to each of our notions of completeness there is a notion of a good rewrite strategy. We examine and characterize these notions of goodness, and give examples of a number of natural good strategies. Although we have confined ourselves to FP here, we believe that our techniques (some of which are nontrivial extensions of techniques first used in the context of λ -calculus) will apply well beyond the realm of FP rewriting systems.¹

¹ Preliminary versions of portions of this paper appear in the Proceedings of the 12th ACM Conference on "Principles of Programming Languages and the Proceedings of the IEEE Symposium on Logic in Computer Science".

1. Introduction

Recently there have appeared in the literature a number of programming languages in which *rewrite* (or *reduction*) rules are used to provide an operational semantics or a means of proving program equivalence ([Pl77,O'D77,Ba78,Tu79,BMS80,Bo82,Wi82,GMP82,HO84]). When rewrite rules are used in this way, there are two obvious questions:

1. Are there too many rules, thus making equivalent two expressions that should be different?
2. Are there enough rules to be able to compute the correct meaning of all expressions?

In this paper we develop a formal framework in which to give precise answers to these soundness and completeness questions. We investigate these issues in the context of an extended version of the functional programming language FP [Ba78]; however, the applicability of these techniques extends far beyond the realm of FP rewriting systems.

Our first step is to give a formal denotational semantics to FP. This in itself is a nontrivial task. FP can be viewed as an untyped combinatory calculus, so we have the usual difficulties of giving semantics in such a way that every expression can be viewed as both a function and an argument to a function (cf. [Me82]), as well as the problems involved in representing infinite sequences.

Once we have given the semantics, answering the first question above becomes straightforward. We must simply show that any two expressions that we intend to be distinct are assigned different meanings, and then show that the reduction rules preserve meaning. Thus if $x \rightarrow y$ is a reduction rule, we must show that $\mu(x) = \mu(y)$, where μ is the semantic function that assigns a meaning to each FP expression.

Proving completeness is a somewhat more delicate task. Part of the problem is that it is not even clear what it means for there to be “enough” rules. We might hope to be able to prove that whenever two FP expressions x and y have the same meaning (i.e., $\mu(x) = \mu(y)$) it is possible to show that there exists some z such that both x and y reduce to z . This turns out to be too much to expect. Just as for combinatory calculus, we know that all recursive functions can be encoded in FP [Wi81], so doing this would be at least as hard as proving the equality of two recursive functions, a problem well known not to be recursively enumerable (cf. [Ro67]). Thus we must aim for somewhat weaker notions of completeness.

The first notion of completeness we consider depends on the observation that in FP, although programmers can write higher-order functions, programs are usually written to compute *observable expressions*, where an observable expression is one that can be formed from atoms using the sequence constructor. Roughly speaking, what we will show is that if the meaning of an FP expression x is the same as that of an observable expression y , then we have enough rewrite rules to reduce x to y . We call this type of completeness *observable completeness*.

one which eventually rewrites every redex, is S-safe. This lets us define a number of natural S-safe rewrite strategies. It is interesting to note that some previously well-studied strategies such as *leftmost-outermost* (cf. [MNV]) turn out not even to be O-safe in the context of FP.

Our techniques for proving completeness are very similar to those used by Wadsworth ([Wa76,Wa78]) for proving limiting completeness for the untyped λ -calculus. However, FP presents two difficulties that do not appear in the λ -calculus. The first difficulty is caused by having two-sided infinite sequences. The second (and more serious) difficulty is that the rewrite rules for FP have much more structure than those of λ -calculus. For example, in the λ -calculus, any expression of the form $(\lambda x.M)N$ is a redex. Any rewriting done to M and N results in an expression of the form $(\lambda x.M')N'$ which is still a redex. But in FP, we may start with an expression of the form $tl:x$ which is not a redex, rewrite x to an expression x' such that $tl:x'$ is a redex, and then rewrite x' further to an expression x'' such that $tl:x''$ is not a redex. Additional problems are caused by the added structure imposed by having sequences. The net result is that it is much more difficult to carry out the proofs for FP than for λ -calculus, and new, nontrivial insights are required.

In other related work, Dosch and Moller [DM84] develop a set of rewriting rules similar to ours, but they take for their semantic domain the term model underlying the algebra of rules. Because their semantic domain is so intimately related to the reduction rules, it does not seem possible for them to even state a completeness theorem of the same type as ours. For them, essentially *by definition*, the meaning of x is the same as that of y iff there is some z such that both x and y can be reduced to z . (For these and other reasons we also feel that the semantic domain we introduce here is a more natural one for programmers to use in reasoning about their programs and for us to use in designing the language.)

The rest of the paper is organized as follows. In the next section we discuss the syntax and semantics of the version of FP that we use, which differs somewhat from the original FP of [Ba78]. In Section 3 we present the rewrite rules for FP and discuss the notions of observable completeness and strong completeness. In Section 4 we discuss strong and weak normalization for FP and show that strong completeness holds for a large subset of FP (namely, all those expressions that are weakly normalizable). In Section 5 we introduce FP' , show that it is strongly normalizable, and use that fact to show that strong completeness holds for full FP. In Section 6 we consider good rewriting strategies. We define the notions of O-safety and S-safety for rewriting strategies, and characterize S-safe rewrite strategies in terms of FP' . We also show that fair strategies are S-safe. In Section 7 we consider limiting completeness and the corresponding notion for rewrite strategies, L-safety. We characterize L-safety in terms of FP' , and then use that characterization to show that S-safety implies L-safety. We conclude in Section 8 with some directions for future work and some remarks on the Church-Rosser property for FP. We have written the paper with two audiences in mind: the researcher in the field of programming semantics and the casual reader with little

provide programming ease and power are included; in fact, we do not even have the basic arithmetic operations. To include them would only clutter our proofs with uninteresting details and trivial case analyses, without adding any conceptually different mathematical considerations. In the following definition we present a language that contains the mathematically interesting and challenging essence of extended FP when considered from the viewpoint of rewrite rule soundness and completeness.

Syntax. We assume the existence of a set $\mathcal{A}$ of *atoms*, which includes the integers and strings of letters. We require that T and F be atoms. We also assume the existence of a set $\mathcal{F}$ of primitive functions symbols that is disjoint from $\mathcal{A}$. The set $\mathcal{F}$ we consider in this paper is {al, first, tl, ar, last, tr, null, comp, cons, apply, K, id, cond}. This set of functions is sufficiently rich to allow us to encode all recursive functions.

Definition 2.1. The *FP expressions* are the least set containing $\mathcal{A} \cup \mathcal{F}$ and closed under:

1. The formation of sequences: if $x_1, \dots, x_n$ are FP expressions (for $n \geq 0$), then $\langle x_1, \dots, x_n \rangle$ is also an FP expression.
2. Application of expressions: if x_1 and x_2 are FP expressions, then $(x_1 : x_2)$ is also an FP expression.

As usual, we include parentheses only for the sake of clarity and assume that application associates to the left, so that we write $x:y:z$ rather than $(x:y):z$. The following are examples of FP expressions: a , $\langle a, \langle b \rangle \rangle$, $\text{null} : \langle \rangle$, $\text{comp} : \langle \text{first}, \text{tl} \rangle : (\text{al} : \langle a, \langle b \rangle \rangle)$, $\text{comp} : \text{ar}$, and $a : \text{tl}$.

Semantics. We provide a denotational semantics for FP as usual by defining a domain D^* and a mapping μ from FP expressions to D^* . The construction we present is elementary and completely avoids the use of category theoretic notions. Before doing so, however, we give a brief intuitive description of the intended meaning of each of the primitive functions and of our treatment of infinite objects. In this informal description of the primitives we anticipate the definition of μ by using “ $\mu(x) = \mu(y)$ ” as an abbreviation for “the meaning of x is the same as the meaning of y .”

1. **al** takes a pair as argument and forms a new sequence by adding the first argument to the left of the second argument (if the second argument is a sequence); thus $\mu(\text{al} : \langle x, \langle y_1, \dots, y_n \rangle \rangle) = \mu(\langle x, y_1, \dots, y_n \rangle)$.
2. **first** takes a sequence as argument and returns the first element of the sequence; thus $\mu(\text{first} : \langle x_1, \dots, x_n \rangle) = \mu(x_1)$.
3. **tl** removes the first element of a sequence, so $\mu(\text{tl} : \langle x_1, \dots, x_n \rangle) = \mu(\langle x_2, \dots, x_n \rangle)$.
4. There are symmetric operations **ar**, **last**, and **tr** which operate on the right side of the sequence instead of the left in an analogous manner to **al**, **first**, and **tl**.
5. **null** returns T when given the empty sequence and F when given a non-empty sequence or an atom. Thus, for example, $\mu(\text{null} : \langle \rangle) = \mu(T)$ and $\mu(\text{null} : \langle x_1, \dots, x_n \rangle) = \mu(F)$.

right hand sides (via first and last). Therefore, we can safely identify such sequences as $\langle x, *, y \rangle$ and $\langle x, *, z, *, y \rangle$ (since the z between the $*$'s in the latter sequence can never be accessed).

We remark that $\langle * \rangle$ is identified with $\perp$ in the domain. We could have included it and treated it as the unique minimal sequence item, but we chose not to, since there is no primitive that is capable of distinguishing it from $\perp$.

We now present a formal construction of a domain D^* that satisfies the domain equation $D^* = \mathcal{A} + Seqs(D^*) + (D^* \rightarrow D^*)$ where $(D^* \rightarrow D^*)$ is the set of continuous functions (a term to be defined later) from D^* to itself and $Seqs(D^*)$ contains all sequences (including infinite sequences) whose elements are members of D^* . More precisely, every element of $Seqs(D^*)$ will have one of the five forms (where all x_i and y_i are members of D^*): $\langle x_1, \dots, x_n \rangle$, $\langle x_1, \dots, x_m, *, y_1, \dots, y_n \rangle$, $\langle x_1, \dots, x_m, \dots, *, y_1, \dots, y_n \rangle$ (an infinite number of x_i 's), $\langle x_1, \dots, x_m, *, \dots, y_n, \dots, y_1 \rangle$ (an infinite number of y_i 's), or $\langle x_1, \dots, x_m, \dots, *, \dots, y_n, \dots, y_1 \rangle$ (an infinite number of both x_i 's and y_i 's). To construct a suitable domain we would like to be able simply to give the domain equation and then rely upon the well-developed body of work on denotational semantics to guarantee us that its solution would serve as the desired domain ([St77, SP82]). Unfortunately, because of the two-sided nature of our sequences, no solution to the above equation is possible using the standard constructors $+$, $\times$, and $\rightarrow$. For, consider the set of partial sequences $\{\langle a, * \rangle, \langle *, a \rangle\}$. Both $\langle a \rangle$ and $\langle a, *, a \rangle$ are minimal upper bounds for this set, but since they are incomparable this set has no *least* upper bound. But we know ([SP82]) that any solution to a domain equation built with the constructors $+$, $\times$, and $\rightarrow$ has the property that any set in the domain that has an upper bound *does* have a least upper bound. Nickl ([Ni85]) encountered a similar difficulty in attempting to provide a denotational semantics for a somewhat different approach to extending FP ([DM84]). She started with standard constructions but then was forced to make some special *ad hoc* modifications to obtain the proper domain. (We remark that our construction does not work if we allow more than one star in the outermost level of a sequence item. See [SP82] for a treatment of sufficient conditions for a domain equation to have “nice” solutions.)

We choose to begin from basics, both because we feel that doing so results in a semantic domain that is simpler and more easily understood by the programmer, and because by doing so we can illuminate the difference between this approach and others and develop useful terminology for our subsequent completeness proofs. Our approach is of course very similar to ([SP82, St77, Wa77]) in that we define an ascending chain of domains each with a $\leq$ relation and close off under a limiting process. However, we are able to avoid the use of inverse limits and category-theoretical machinery. In order to describe our domains, we recall a few standard technical definitions.

Definition 2.2. A *pre-ordered domain* $(D, \leq)$ consists of a set D and a reflexive, transitive ordering $\leq$. Intuitively, $x \leq y$ if y has at least as much “information” as x . (Note that we

$$P_m^n(\langle x_1, \dots, x_i \rangle) = \begin{cases} \langle P_{m-1}^{n-1}(x_1), \dots, P_{m-1}^{n-1}(x_i) \rangle & \text{if } i \leq 2m \\ \langle P_{m-1}^{n-1}(x_1), \dots, P_{m-1}^{n-1}(x_m), *, P_{m-1}^{n-1}(x_{i-m+1}), \dots, P_{m-1}^{n-1}(x_i) \rangle & \text{if } i > 2m \end{cases}$$

if $\langle x_1, \dots, x_i \rangle$ is a sequence of definite length

$$P_m^n(\langle x_1, \dots, x_i, *, y_1, \dots, y_j \rangle) = \langle P_{m-1}^{n-1}(x_1), \dots, P_{m-1}^{n-1}(x_{i'}), *, P_{m-1}^{n-1}(y_{j'}), \dots, P_{m-1}^{n-1}(y_j) \rangle$$

where $i' = \min(i, m)$, $j' = \max(1, j - m + 1)$

$$P_m^n(f) = \begin{cases} f & \text{if } f \in (D_k \rightarrow_M D_k), k < m \\ g & \text{if } f \in (D_k \rightarrow_M D_k), k \geq m \end{cases}$$

where $g \in (D_{m-1} \rightarrow_M D_{m-1})$ is defined by $g(x) = P_{m-1}^{n-1}(f(x))$.

3. For $n \leq N$, we have

$$f \cdot_n x = \begin{cases} f(P_k^n(x)) & \text{if } f \in (D_k \rightarrow_M D_k), k < n \\ \perp & \text{otherwise.} \end{cases}$$

4. For $1 \leq n \leq N$, $\leq_n$ is the least reflexive, transitive relation on D_n satisfying:

- a. $\perp \leq_n x$ for every item $x \in D_n$,
 - b. $\langle x_1, \dots, x_l \rangle \leq_n \langle y_1, \dots, y_l \rangle$ if $x_j \leq_{n-1} y_j$ or $x_j = * = y_j$, for $j = 1, \dots, l$,
 - c. $\langle x_1, \dots, x_l, *, y_1, \dots, y_m \rangle \leq_n \langle x_1, \dots, x_l, z_1, \dots, z_k, y_1, \dots, y_m \rangle$ for $k \geq 0$,
 - d. $f \leq_n g$ if $\forall x \in D_{n-1} (f \cdot_n x \leq_{n-1} g \cdot_n x)$, where f and g are function items.
5. P_m^n is a monotonic function for $m \leq n \leq N$.
6. D_n is a finite set for all $n \leq N$.

As we remarked above, we can now define D_{N+1} using clause 1 (with n replaced by $N+1$ of course), P_m^{N+1} for $m \leq N+1$ using clause 2, $\cdot_{N+1}$ using clause 3, and $\leq_{N+1}$ using clause 4. We must check that P_m^{N+1} is well defined; in particular, we must check that if $f \in (D_k \rightarrow D_k)$, $k \geq m$, then $P_m^{N+1}(f)$ as defined is indeed a monotonic function, and so is an element of $(D_{m-1} \rightarrow_M D_{m-1})$. Since $(P_m^{N+1}(f))(x) = P_{m-1}^N(f(x))$ in this case, the result follows from the monotonicity of f and P_{m-1}^N . Clauses 1-4 hold by definition, and clauses 5-6 are easy to check.

This completes the definition of the chain of pre-ordered domains. A few remarks may be helpful. The definition of $\leq_n$ guarantees that function items, sequence items, and atomic items are pairwise incomparable. As observed earlier, least upper bounds do not necessarily exist. In particular, the set $\{\langle a_1, * \rangle, \langle *, a_1 \rangle\}$ in D_2 has no least upper bound (although it has minimal upper bounds $\langle a_1 \rangle$ and $\langle a_1, *, a_1 \rangle$). Finally, $\leq_n$ is not a partial order since it is not antisymmetric. For example, if f is the function in D_3 that maps all elements of D_2 into a_1 and g is the function in D_5 that maps all elements of D_4 into a_1 , then $f \equiv_5 g$.

The definition of P_m^n on sequence items is a bit complicated. The conditions are necessary to deal with the restrictions on lengths of sequences in D_m . For example, if we want to project a sequence item of definite length in D_n to D_m , there is no problem if the length is at most $2m$; however, if its length is greater than $2m$, we must add a $*$ to the middle of the

If $X_1, \dots, X_n$ are elements of D^* , we can view $\langle X_1, \dots, X_n \rangle$ as an element of D^* by identifying it with $\{\langle y_1, \dots, y_n \rangle \mid y_i \in X_i\}^d$. Thus we can view D^* as closed under the formation of sequences. We extend the terminology for D to D^* by saying that an item X in D^* is an atom (resp. sequence item, function item) iff one of the elements of X is an atom (resp. sequence item, function item). Since X is directed, this terminology is well defined.

We extend the $:$ operation to D^* by defining $:$ on $D^* \times D^*$ by

$$X:Y = \{x:y \mid x \in X, y \in Y\}.$$

Proposition 2.5. *If $X, Y \in D^*$, then $X:Y \in D^*$.*

Proof. $X:Y$ is clearly non-empty. To check that $X:Y$ is directed, let $z_1, z_2 \in X:Y$. Thus, $z_i = x_i:y_i$ for $i = 1, 2$. Since X and Y are both directed, we can find $x_3 \in X$ and $y_3 \in Y$ such that $x_1, x_2 \leq_I x_3$ and $y_1, y_2 \leq_I y_3$. It is easy to check that $z_1, z_2 \leq_I x_3:y_3 \in X:Y$. To check that $X:Y$ is downward-closed, let $z \in X:Y$ and let $z' \leq_I z$. We may clearly assume that $z \neq \perp$. Thus, $z = x:y$ for some $x \in X, y \in Y$. Since $z \neq \perp$, we must have $x \in (D_k \rightarrow_M D_k)$ for some k . Define $f \in (D_k \rightarrow_M D_k)$ by

$$f(t) = \begin{cases} z' & \text{if } z' \leq_I x(t) \\ \perp & \text{otherwise.} \end{cases}$$

It is easy to see that f is monotonic (thus in $D_k \rightarrow_M D_k$) and that $f \leq_I x$. It follows that $f \in X$, since X is downward closed. Thus $z' = f:y \in X:Y$ as desired. $\square$

The construction of D^* results in a number of nice properties. In order to state them, we recall some standard domain-theoretic terminology.

Definition 2.6. Let $(D', \leq)$ be a domain. $(D', \leq)$ is *complete* iff every directed subset of D' has a least upper bound in D' . The least upper bound of the directed set X is denoted by $\sqcup X$. x is a *finite element* of D' iff whenever $x \leq \sqcup Y$ for some nonempty directed subset Y of D' , then $x \leq y$ for some $y \in Y$. $(D', \leq)$ is *algebraic* iff for every $x \in D'$, the set $\{y \leq x \mid y \text{ is a finite element of } D'\}$ is directed and $x = \sqcup \{y \leq x \mid y \text{ is a finite element of } D'\}$.

Proposition 2.7.

1. D^* is a cpo (complete partial order). Moreover, if $\mathcal{X}$ is a non-empty, directed subset of D^* , then $\sqcup \mathcal{X} = (\cup_{X \in \mathcal{X}} X)$.
2. $X \in D^*$ is a finite element of D^* iff $X = \{x\}^d$ for some $x \in D$.
3. D^* is an algebraic domain.

Proof. Clearly D^* is partially ordered by $\leq$. Let $\mathcal{X}$ be a directed subset of D^* and let $Y = \cup_{X \in \mathcal{X}} X$. Clearly Y is a subset of any upper bound of $\mathcal{X}$ in D^* , so once we show that $Y \in D^*$, it will follow that Y is the least upper bound of $\mathcal{X}$ in D^* . Y is easily seen to be a non-empty and downward closed subset of D , so all we must show is that it is directed. If $y_1, y_2 \in Y$, then, by construction, there must be $X_i \in \mathcal{X}$ such that $y_i \in X_i$, $i = 1, 2$. Since $\mathcal{X}$ is directed, there be some $X_3 \in \mathcal{X}$ such that $X_1 \subseteq X_3$ and $X_2 \subseteq X_3$. Thus we have $y_1, y_2 \in X_3$. Since X_3 is directed, there must be some $y_3 \in X_3$ such that $y_1, y_2 \leq_I y_3$. Since $y_3 \in Y$ by construction,

Intuitively, this function ought to be continuous; such a function is easily definable in FP. Let $X = \{\{\langle a, \perp \rangle\}^d, \{\langle \perp, a \rangle\}^d\}$. Then X is a subset of D^* whose least upper bound in D^* is easily seen to be $\{\langle a, a \rangle\}^d$ (note that any directed set that is an upper bound will have to contain $\langle a, a \rangle$). However, f applied to each of the two elements in X is $\perp$ (or, more accurately, $\{\perp\}^d$), while f applied to $\{\langle a, a \rangle\}^d$ is $\{T\}^d$, so f would not be continuous had we required that equality for the least upper bound operation hold even in the case that X was not directed.

Proposition 2.10.

1. $\cdot$ as defined on D^* is continuous in each of its arguments.
2. D^* is isomorphic to $\mathcal{A} + Seqs(D^*) + (D^* \rightarrow D^*)$.
3. If $F, G \in D^*$ are function items and $F:X = G:X$ for all $X \in D^*$, then $F = G$.

Proof. To prove part 1, let $\mathcal{Y}$ be a directed subset of D^* and $X \in D^*$. From Proposition 2.7, it follows that $X:(\sqcup \mathcal{Y}) = X:(\cup_{Y \in \mathcal{Y}} Y) = (\cup_{Y \in \mathcal{Y}} X:Y) = \sqcup \{X:Y \mid Y \in \mathcal{Y}\}$. Thus $\cdot$ is continuous in its second argument. The proof that $\cdot$ is continuous in its first argument is completely analogous.

To prove part 2, following Scott and Stoy (cf. [St77]), we construct two monotonic functions $\Phi: D^* \rightarrow (\mathcal{A} + Seqs(D^*) + (D^* \rightarrow D^*))$ and $\Psi: (\mathcal{A} + Seqs(D^*) + (D^* \rightarrow D^*)) \rightarrow D^*$ such that $\Phi \circ \Psi$ is the identity on $\mathcal{A} + Seqs(D^*) + (D^* \rightarrow D^*)$ and $\Psi \circ \Phi$ is the identity on D^* . Φ maps atomic items to $\mathcal{A}$, sequence items to $Seqs(D^*)$ and function items to $(D^* \rightarrow D^*)$ in the obvious way. For example, if X is a function item, then we define $\Phi(X) \in (D^* \rightarrow D^*)$ via $\Phi(X)(Y) = X:Y$. From part 1 it follows that $\Phi(X)$ is a continuous function from D^* to D^* , i.e., an element of $(D^* \rightarrow D^*)$.

Ψ maps elements of $\mathcal{A}$ to atomic items, elements of $Seqs(D^*)$ to sequence items, and elements of $(D^* \rightarrow D^*)$ to function items. We give the definition of Ψ on elements of $(D^* \rightarrow D^*)$, leaving the other cases to the reader. For $F \in (D^* \rightarrow D^*)$ define $Q_k(F) \in (D_k \rightarrow_M D_k)$ by $Q_k(F)(x) = P_k(F(\{x\}^d))$. Define $\Psi(F) = \cup_i \{Q_i(F)\}^d$. From the fact that $P_{i+1}(Q_j(F)) \equiv Q_i(F)$ for $i \leq j$, it follows that $\Psi(F)$ is a directed subset of D , and hence an element of D^* . It is clear that Ψ and Φ are both monotonic functions.

To show that $\Phi \circ \Psi$ and $\Psi \circ \Phi$ are both the identity on their respective domains, we again restrict our attention to the proof for function items, leaving the remaining cases to the reader. To show that $\Phi \circ \Psi$ is the identity, let $F \in (D^* \rightarrow D^*)$ and $x \in D_k$. Then

$$\begin{aligned} \Phi(\Psi(F))(\{x\}^d) &= \Psi(F):\{x\}^d = \bigcup_i \{Q_i(F)\}^d:\{x\}^d = \bigcup_i \{Q_i(F):x\}^d \\ &= \bigcup_{i \geq k} \{Q_i(F):x\}^d = \bigcup_{i \geq k} \{P_i(F(\{x\}^d))\}^d = F(\{x\}^d). \end{aligned}$$

By continuity, $\Phi(\Psi(F))(X) = F(X)$ for all $X \in D^*$. Therefore, $\Phi(\Psi(F)) = F$ thereby proving that $\Phi \circ \Psi$ is the identity on the functions in $\mathcal{A} + Seqs(D^*) + (D^* \rightarrow D^*)$.

To show that $\Psi \circ \Phi$ is the identity, let $F \in D^*$ be a function item. It suffices to show that $P_{k+1}(\Psi(\Phi(F))) \equiv P_{k+1}(F)$ for all k . Since $P_{k+1}(\Psi(\Phi(F))) = P_{k+1}(\bigcup_i \{Q_i(\Phi(F))\}^d) \equiv Q_k(\Phi(F))$, it suffices to show that $Q_k(\Phi(F)) \equiv P_{k+1}(F)$ for all k . Let $x \in D_k$. Then

refrain from pursuing such considerations and leave them to be described in a forthcoming paper which will concentrate on programming language design issues.

We have already seen the definition of al_{n+1} ; the remaining approximating functions are:

$$\begin{aligned}
ar_{n+1} : \begin{cases} \langle \langle y_1, \dots, y_m \rangle, x \rangle = \langle y_1, \dots, y_m, x \rangle & \text{if } x \neq * \\ \langle y, x \rangle = \langle *, x \rangle & \text{if } y \neq * \neq x \end{cases} \\
first_{n+1} : \langle x_1, \dots, x_m \rangle = x_1 \text{ if } m \geq 1 \text{ and } x_1 \neq * \\
last_{n+1} : \langle x_1, \dots, x_m \rangle = x_m \text{ if } m \geq 1 \text{ and } x_m \neq * \\
tl_{n+1} : \begin{cases} \langle x_1, * \rangle = \perp \\ \langle x_1, \dots, x_m \rangle = \langle x_2, \dots, x_m \rangle & \text{if } m \geq 1 \text{ and } x_1 \neq * \\ \langle *, x_2, x_3, \dots, x_m \rangle = \langle *, x_3, \dots, x_m \rangle & \text{if } m \geq 3 \end{cases} \\
tr_{n+1} : \begin{cases} \langle *, x_2 \rangle = \perp \\ \langle x_1, \dots, x_m \rangle = \langle x_1, \dots, x_{m-1} \rangle & \text{if } m \geq 1 \text{ and } x_m \neq * \\ \langle x_1, \dots, x_{m-1}, x_m, * \rangle = \langle x_1, \dots, x_{m-1}, * \rangle & \text{if } m \geq 2 \end{cases} \\
null_{n+1} : \begin{cases} \langle \rangle = P_n(T) \\ a = P_n(F) & \text{if } a \text{ an atom} \\ \langle x_1, \dots, x_m \rangle = P_n(F) & \text{if } m \geq 1 \end{cases} \\
cons_{n+1} : \begin{cases} \langle f_1, \dots, f_m \rangle = h \in (D_{n-1} \rightarrow D_{n-1}) \text{ if } f_i \neq *, \text{ where} \\ \quad h(x) = P_{n-1}(\langle f_1 : x, \dots, f_m : x \rangle) \\ \langle f_1, \dots, f_k, *, f_{k+1}, \dots, f_m \rangle = h \in (D_{n-1} \rightarrow D_{n-1}), \text{ where} \\ \quad h(x) = P_{n-1}(\langle f_1 : x, \dots, f_k : x, *, f_{k+1} : x, \dots, f_m : x \rangle) \end{cases} \\
cond_{n+1} : \begin{cases} \langle T, u, v \rangle = u \text{ if } u \neq * \neq v \\ \langle F, u, v \rangle = v \text{ if } u \neq * \neq v \end{cases} \\
comp_{n+1} : \langle f, g \rangle = h \in (D_{n-1} \rightarrow D_{n-1}) \text{ if } f \neq * \neq g, \text{ where } h(x) = f:(g:x) \\
apply_{n+1} : \langle f, x \rangle = f:x \text{ if } f \neq * \neq x \\
K_{n+1} : x = h \in (D_{n-1} \rightarrow D_{n-1}), \text{ where } h(z) = P_{n-1}(x) \\
id_{n+1} : x = x.
\end{aligned}$$

We now complete the definition of μ by taking, for each primitive function symbol f ,

$$\mu(f) = \{f_n \mid n \in \mathbb{N}\}^d.$$

Of course, we must check that $\mu(f)$ is well defined for each primitive function symbol f . In particular, we must check that f_n is indeed a monotonic function. The details in each case are straightforward and left to the reader. We also leave it to the reader to check that $f_i = P_i(f_j)$ for $i \leq j$. Note that $\mu(f) \in D^*$ since $f_i \leq_I f_j$ if $i \leq j$ for each of the primitive functions.

$\text{al: } \langle x, \langle y_1, \dots, y_n \rangle \rangle \rightarrow \langle x, y_1, \dots, y_n \rangle$
 $\text{ar: } \langle \langle y_1, \dots, y_n \rangle, x \rangle \rightarrow \langle y_1, \dots, y_n, x \rangle$
 $\text{first: } \langle x_1, \dots, x_n \rangle \rightarrow x_1 \quad (n \geq 1)$
 $\text{first:}(\text{al: } \langle x, y \rangle) \rightarrow x$
 $\text{first:}(\text{ar: } \langle \text{al: } \langle x, y \rangle, z \rangle) \rightarrow x$
 $\text{first:}(\text{ar: } \langle \text{ar: } \langle x, y \rangle, z \rangle) \rightarrow \text{first:}(\text{ar: } \langle x, y \rangle)$
 $\text{last: } \langle x_1, \dots, x_n \rangle \rightarrow x_n \quad (n \geq 1)$
 $\text{last:}(\text{ar: } \langle x, y \rangle) \rightarrow y$
 $\text{last:}(\text{al: } \langle x, \text{ar: } \langle y, z \rangle \rangle) \rightarrow z$
 $\text{last:}(\text{al: } \langle x, \text{al: } \langle y, z \rangle \rangle) \rightarrow \text{last:}(\text{al: } \langle y, z \rangle)$
 $\text{tl: } \langle x_1, \dots, x_n \rangle \rightarrow \langle x_2, \dots, x_n \rangle \quad (n \geq 1)$
 $\text{tl:}(\text{al: } \langle x, \text{al: } \langle y, z \rangle \rangle) \rightarrow \text{al: } \langle y, z \rangle$
 $\text{tl:}(\text{al: } \langle x, \text{ar: } \langle y, z \rangle \rangle) \rightarrow \text{ar: } \langle y, z \rangle$
 $\text{tl:}(\text{ar: } \langle \text{al: } \langle x, y \rangle, z \rangle) \rightarrow \text{ar: } \langle y, z \rangle$
 $\text{tl:}(\text{ar: } \langle \text{ar: } \langle x, y \rangle, z \rangle) \rightarrow \text{ar: } \langle \text{tl:}(\text{ar: } \langle x, y \rangle), z \rangle$
 $\text{tr: } \langle x_1, \dots, x_n \rangle \rightarrow \langle x_1, \dots, x_{n-1} \rangle \quad (n \geq 1)$
 $\text{tr:}(\text{ar: } \langle \text{ar: } \langle x, y \rangle, z \rangle) \rightarrow \text{ar: } \langle x, y \rangle$
 $\text{tr:}(\text{ar: } \langle \text{al: } \langle x, y \rangle, z \rangle) \rightarrow \text{al: } \langle x, y \rangle$
 $\text{tr:}(\text{al: } \langle x, \text{ar: } \langle y, z \rangle \rangle) \rightarrow \text{al: } \langle x, y \rangle$
 $\text{tr:}(\text{al: } \langle x, \text{al: } \langle y, z \rangle \rangle) \rightarrow \text{al: } \langle x, \text{tr:}(\text{al: } \langle y, z \rangle) \rangle$
 $\text{null: } \langle \rangle \rightarrow T$
 $\text{null: } a \rightarrow F \quad (\text{for each atom } a)$
 $\text{null: } \langle x_1, \dots, x_n \rangle \rightarrow F \quad (n \geq 1)$
 $\text{null:}(\text{al: } \langle x, y \rangle) \rightarrow F$
 $\text{null:}(\text{ar: } \langle x, y \rangle) \rightarrow F$
 $(\text{cons: } \langle f_1, \dots, f_n \rangle):x \rightarrow \langle f_1:x, \dots, f_n:x \rangle$
 $(\text{cons:}(\text{al: } \langle f, g \rangle)):x \rightarrow \text{al: } \langle f:x, (\text{cons:}g):x \rangle$
 $(\text{cons:}(\text{ar: } \langle f, g \rangle)):x \rightarrow \text{ar: } \langle (\text{cons:}f):x, g:x \rangle$
 $\text{cond: } \langle T, x, y \rangle \rightarrow x$
 $\text{cond: } \langle F, x, y \rangle \rightarrow y$
 $(\text{comp: } \langle f, g \rangle):x \rightarrow f:(g:x)$
 $\text{apply: } \langle f, x \rangle \rightarrow f:x$
 $(K:x):y \rightarrow x$
 $\text{id:}x \rightarrow x$

Figure 1: The FP rewrite rules.

which can in turn be rewritten to $\text{tl:}x''$ which is not a redex. To see this, consider $\text{tl:}(\text{id:}(\text{al: } \langle a, \text{al: } \langle b, \langle c \rangle \rangle \rangle))$. This expression is not itself a redex, but it can be rewritten to $\text{tl:}(\text{al: } \langle a, \text{al: } \langle b, \langle c \rangle \rangle \rangle)$, which is a redex (w.r.t. the rule $\text{tl:}(\text{al: } \langle x, \text{al: } \langle y, z \rangle \rangle) \rightarrow \text{al: } \langle y, z \rangle$). This in turn can be rewritten to $\text{tl:}(\text{al: } \langle a, \langle b, c \rangle \rangle)$ which is again *not* a redex.

x	$\langle \rangle$	$\langle x_1, \dots, x_n \rangle \ (n \geq 1)$	$\langle *, x_2, \dots, x_n \rangle \ (n \geq 2)$	non-seq
$ar:\langle x, y \rangle$	$\langle y \rangle$	$\langle x_1, \dots, x_n, y \rangle$	$\langle *, x_2, \dots, x_n, y \rangle$	$\langle *, y \rangle$
$ar:\langle ar:\langle x, y \rangle, z \rangle$	$\langle y, z \rangle$	$\langle x_1, \dots, x_n, y, z \rangle$	$\langle *, x_2, \dots, x_n, y, z \rangle$	$\langle *, y, z \rangle$
$tl:(ar:\langle ar:\langle x, y \rangle, z \rangle)$	$\langle z \rangle$	$\langle x_2, \dots, x_n, y, z \rangle$	$\langle *, x_3, \dots, x_n, y, z \rangle$	$\langle *, z \rangle$
$tl:(ar:\langle x, y \rangle)$	$\langle \rangle$	$\langle x_2, \dots, x_n, y \rangle$	$\langle *, x_3, \dots, x_n, y \rangle$	$\perp$
$ar:\langle tl:(ar:\langle x, y \rangle), z \rangle$	$\langle z \rangle$	$\langle x_2, \dots, x_n, y, z \rangle$	$\langle *, x_3, \dots, x_n, y, z \rangle$	$\langle *, z \rangle$

Observe that rows 3 and 5 in the table, representing the left and right sides of the rewrite rule, are identical. Thus, for all possible meanings of x , the rule is sound. $\square$

Besides soundness, we also want to know, as we said in the introduction, that we have *enough* rewrite rules. Do the rewrite rules really capture the semantics of the language? Do they allow us to carry out all the computing we want?

Ideally we would like to show that if $\mu(x) = \mu(y)$, then we have some z such that $x \rightarrow^* z$ and $y \rightarrow^* z$. As we mentioned in the introduction, this goal is unattainable. However, it is attainable if we restrict x and y to be expressions whose meaning is observable. Since these are the expressions the programmer most cares about, this is a reasonable sense in which we do indeed have enough rules. To make this precise, we need some definitions.

Definition 3.3. Let the *observable expressions* be those that can be formed starting with atoms and closing off under the sequence constructor. More formally, the set of observable expressions is the least set $\mathcal{O}$ such that:

1. $\mathcal{A} \subseteq \mathcal{O}$,
2. if $x_1, \dots, x_n \in \mathcal{O}$, then $\langle x_1, \dots, x_n \rangle \in \mathcal{O}$.

To each observable expression there corresponds an *observable item* which is obtained by replacing $\langle, \rangle$, and boldface wherever they occur in the observable expression by $\langle, \rangle$, and italics respectively. Note that neither $\perp$ nor $*$ appears in any sequence item which is observable. It is easy to check that every observable item is finite (i.e., has the form $\{x\}^d$ for some $x \in D$). Given an observable item x , let x^e be the observable expression to which it corresponds.

Thus, one reasonable notion of there being “enough” rewrite rules is:

(*) If x is an FP expression such that $\mu(x)$ is an observable item, then $x \rightarrow^* \mu(x)^e$.

For example, if we have an expression x such that $\mu(x) = \langle a, \langle b \rangle \rangle$, where a and b are atomic items, then from (*) it would follow that $x \rightarrow^* \langle a, \langle b \rangle \rangle$, even if x contains subexpressions whose meanings are functions or infinite sequences.

In Section 5, we prove that strong completeness holds for FP. By Proposition 3.6, observable completeness also holds for FP. As we shall see, the proof of strong completeness is strongly intertwined with notions of normalization, which is the subject of the next section. The techniques introduced there will also allow us to give a relatively easy proof of strong completeness for a large subset of FP.

4. Normalization

Since rewriting can be viewed as a computing process, we might want to ask if that process always terminates. (Some computation paths may terminate by producing an expression to which no rules apply, while other computation paths may run forever.) Intuitively, we say an expression x is weakly normalizable iff there is some computation starting with x that terminates; x is strongly normalizable iff all computations starting with x terminate, or equivalently, iff there are no infinite computations starting with x . This motivates the following definitions.

Definition 4.1. An expression is in *normal form* iff no subexpression is a redex. An expression is *weakly normalizable* (respectively, *strongly normalizable*) iff some (respectively, every) reduction sequence for it leads to a normal form. A language is *weakly normalizable* (respectively, *strongly normalizable*) iff every expression in that language is weakly normalizable (respectively, strongly normalizable).

Not all FP expressions are strongly normalizable or even weakly normalizable for that matter; for example, it is easy to check that fix:id is not weakly normalizable. However, large subsets of FP expressions are strongly normalizable. Let $\text{FP-}\{\text{apply}\}$ (resp. $\text{FP-}\{\text{cons}\}$, $\text{FP-}\{\text{comp}\}$) consist of all FP expressions which do not contain the primitive function apply (resp. cons , comp).

Theorem 4.2. $\text{FP-}\{\text{apply}\}$, $\text{FP-}\{\text{cons}\}$, and $\text{FP-}\{\text{comp}\}$ are all strongly normalizable.

Proof. The idea in all cases is to assign a natural number to the expressions in each of these sublanguages in such a way that every rewrite rule forces the number of the expression to decrease. For the case $\text{FP-}\{\text{apply}\}$, we do this as follows:

$N(a) = 1$ if a is an atom or primitive function,

$N(\langle x_1, \dots, x_n \rangle) = N(x_1) + \dots + N(x_n) + 1$,

$N(f:g) = 2^{N(f)} \times N(g)$.

Whenever we apply a rewrite rule (except for the apply rule), the number of the expression goes down. We prove this for the comp rule:

$$\begin{aligned} N((\text{comp}: \langle x, y \rangle):z) &= 2^{1 \times (N(x) + N(y) + 1)} \times N(z) \\ &> 2^{N(x)} \times 2^{N(y)} \times N(z) \\ &= N(x:(y:z)). \end{aligned}$$

All the other rewrite rules (of $\text{FP-}\{\text{apply}\}$) are just as easy.

of the form $K:y$, $\text{comp}:y$, or $\text{cons}:y$, so the theorem holds, or (d) $\mu(x)$ is a sequence item of indefinite length and x is of the form $\text{al}: \langle y, z \rangle$ or $\text{ar}: \langle y, z \rangle$, so again the theorem holds.

If x_1 is one of K, comp , or cons , then it is easy to see that one of (b) or (c) must hold. If x_1 is of the form id or $K:y$, then we immediately have that $x_1:x_2$ is a redex. If x_1 is of the form apply , then either $\mu(x_1:x_2) = \perp$, or $\mu(x_2) = \langle f, z \rangle$, where $f, z \neq *$. But then by the induction hypothesis, x_2 is of the form $\langle f, z \rangle$, so that $x_1:x_2$ must be a redex. The argument in the case where x_1 is of the form cond , null , $\text{cons}:y$, or $\text{comp}:y$ is similar and left to the reader. If x_1 is of the form al , then either $\mu(x) = \perp$ or $\mu(x_2)$ is of the form $\langle y, z \rangle$, where $y, z \neq *$. Thus, $\mu(x)$ is of the form $\langle w_1, \dots, w_m \rangle$ and, by the induction hypothesis, x_2 is of the form $\langle y, z \rangle$. If $\mu(x)$ is a sequence of definite length, then in fact z must be a sequence of definite length. Thus, by the induction hypothesis, z is of the form $\langle z_1, \dots, z_n \rangle$, and x is a redex. The arguments in the case that x_1 is ar , tl , tr , first , or last are similar and again left to the reader. $\square$

As an immediate corollary to Theorem 4.3, we get

Corollary 4.4. *If x is in normal form and $\mu(x)$ is an observable item, then $x = \mu(x)^e$.*

Proof. By an easy induction on the depth of nesting of $\langle \rangle$'s in $\mu(x)$, similar to the proof of Proposition 3.6. $\square$

From Theorem 4.3, it immediately follows that strong completeness holds for a large subset of FP terms, namely all those terms in $(FP - \{\text{apply}\}) \cup (FP - \{\text{comp}\}) \cup (FP - \{\text{cons}\})$. Indeed, Theorem 4.3 shows that strong completeness holds for all weakly normalizable FP expressions. However, as we have shown, there are some FP expressions that are not weakly normalizable. Nevertheless, we can show that strong completeness holds for full FP: every expression can be rewritten to a transparent expression. This is the subject of the next section.

5. FP' and strong completeness for FP

The key idea in the proof of strong completeness is to construct a variant of FP called FP' on which we impose a notion of *type*. The proof then proceeds in three steps. The first, quite simple, is to show that any FP expression can be approximated arbitrarily well by a corresponding FP' expression (Proposition 5.2). Secondly, we show that FP' reduction sequences correspond in a precise sense to FP reduction sequences (Proposition 5.7). Next we show that FP' is essentially strongly normalizable, in a sense to be made precise below. We then prove an analogue to Theorem 4.3 for FP' . The results then follow using techniques similar to those sketched at the end of the previous section.

We now define FP' , a meaning function μ' that maps FP' expressions to elements of D , a *type* function τ that associates with every FP' expression a natural number (its type), and a “pull-back” function ρ that associates with each FP' expression a corresponding FP expression. FP' expressions look essentially like FP expressions, except that primitive function symbols

$$\begin{cases} k & \text{if } \rho(x) \notin \{\text{al}, \text{ar}\} \\ j-1 & \text{if } \rho(x) = \text{al}, y = P_{m_1}(\dots P_{m_n}(\langle y_1, y_2 \rangle) \dots), n \geq 0, \tau(y_1) < j-1, \text{ and } \mu'(x:y) \in D_{j-1} \\ j-1 & \text{if } \rho(x) = \text{ar}, y = P_{m_1}(\dots P_{m_n}(\langle y_1, y_2 \rangle) \dots), n \geq 0, \tau(y_2) < j-1, \text{ and } \mu'(x:y) \in D_{j-1} \\ j & \text{otherwise} \end{cases}$$

7. If x is an FP' expression, then $P_m(x)$ is an FP' expression; $\mu'(P_m(x)) = P_m(\mu'(x))$, $\tau(P_m(x)) = \min(m, \tau(x))$, and $\rho(P_m(x)) = \rho(x)$.
8. If x is an FP' expression, then $C_{m,n}(x)$ is an FP' expression; $\rho(C_{m,n}(x)) = \rho(x)$, $\tau(C_{m,n}(x)) = \max(m, n, \tau(x))$, and

$$\mu'(C_{m,n}(x)) = \begin{cases} \perp & \text{if } \mu'(x) \text{ is not a sequence} \\ \langle y_1, \dots, y_{\min(i,m)}, *, z_{\max(1, j-n+1)}, \dots, z_j \rangle & \text{if } \mu'(x) = \langle y_1, \dots, y_i, *, z_1, \dots, z_j \rangle \\ \langle y_1, \dots, y_m, *, y_{i-n+1}, \dots, y_i \rangle & \text{if } \mu'(x) = \langle y_1, \dots, y_i \rangle \text{ and } i > m+n \\ \langle y_1, \dots, y_i \rangle & \text{if } \mu'(x) = \langle y_1, \dots, y_i \rangle \text{ and } i \leq m+n \end{cases}$$

We remark that the special case for al and ar in the definition of τ is needed to make Proposition 5.5 below go through. Although the expression looks somewhat complicated, the intuitive idea is motivated by a desire to have $\tau(\text{al}_n: \langle z, \langle z_1, \dots, z_k \rangle \rangle) = \tau(\langle z, z_1, \dots, z_k \rangle)$ for n sufficiently large.

The following proposition shows that an expression in FP' semantically approximates the corresponding FP expression. The second half of the proposition shows that this approximation can be made arbitrarily good.

Proposition 5.2.

1. For any FP' expression x' , we have $\mu'(x') \in \mu(\rho(x'))$.
2. If x is an FP expression, $x_0 \in \mu(x)$, then there exists an FP' expression x' such that $x_0 \leq_I \mu'(x')$ and $\rho(x') = x$.

Proof. The first part is immediate by induction on the structure of FP' expressions. For the second part, we proceed by induction on the structure of the FP expression x . If x is an atom, then we can clearly take $x' = x$. If x is a primitive function symbol f , we take $x' = f_n$, where $x_0 \in D_n$. If x is a sequence of the form $\langle x_1, \dots, x_n \rangle$, then there exist $y_1, \dots, y_n \in D$ such that $x_0 \leq_I \langle y_1, \dots, y_n \rangle$ and $y_j \in \mu(x_j)$. (We can actually choose $y_1, \dots, y_n$ so that $x_0 = \langle y_1, \dots, y_n \rangle$ unless x_0 has a $*$ as one of its components, in which case the corresponding y_i 's are taken to be $\perp$'s.) From the induction hypothesis, it follows that we can find FP' expressions $y_1, \dots, y_n$ such that $y_j \leq_I \mu'(y_j)$ and $\rho(y_j) = x_j$. Thus we can take $x' = \langle y_1, \dots, y_n \rangle$. Finally, if x is of the form $y:z$ and $x_0 \in \mu(x)$, then there must be $y_0, z_0 \in D$ such that $y_0 \in \mu(y)$, $z_0 \in \mu(z)$, and $x_0 = y_0:z_0$. By the induction hypothesis we can find FP' expressions y' and z' such that $\rho(y') = y$, $\rho(z') = z$, $y_0 \leq_I \mu'(y')$, and $z_0 \leq_I \mu'(z')$. It is easy to check that $x' = y':z'$ satisfies the proposition. $\square$

Lemma 5.3. If x' is an FP' expression and $\tau(x') = n$, then $\mu'(x') \in D_n$.

Proof. A straightforward induction on the structure of x' . $\square$

Lowering rules:

$$\begin{aligned}
x &\rightarrow_a \Omega(\rho(x)) \text{ if } \mu'(x) = \perp \text{ and } x \neq \Omega(\rho(x)) \\
x:y &\rightarrow_a x:P_{\tau(x)-1}(y) \text{ if } \tau(y) \geq \tau(x) \text{ and } \mu'(x:y) \neq \perp \\
P_m(x) &\rightarrow_a x \text{ if } m \geq \tau(x) \\
C_{r,s}(x) &\rightarrow_a x \text{ if } \mu'(C_{r,s}(x)) = \mu'(x) \\
P_m(f_n) &\rightarrow_a f_{\min(m,n)}
\end{aligned}$$

P-rules (apply only if the left-hand side is well-typed):

$$\begin{aligned}
C_{r,s}(x) &\rightarrow_a C_{i,j}(x) \text{ if } \mu'(C_{r,s}(x)) = \langle y_1, \dots, y_i, *, z_1, \dots, z_j \rangle \text{ and } i+j < r+s \\
P_m(P_n(z)) &\rightarrow_a P_{\min(m,n)}(z) \\
C_{r,s}(C_{u,v}(x)) &\rightarrow_a C_{\min(r,u), \min(s,v)}(x) \\
P_m(C_{r,s}(x)) &\rightarrow_a C_{\min(r,m), \min(s,m)}(P_m(x)) \\
P_m(\langle x_1, \dots, x_i, *(z_1, \dots, z_j), y_1, \dots, y_k \rangle) &\rightarrow_a \\
&\quad \langle P_{m-1}(x_1), \dots, P_{m-1}(x_{\min(i,m)}), \\
&\quad \quad *(\rho(x_{1+\min(i,m)}), \dots, \rho(x_i), z_1, \dots, z_j, \rho(y_1), \dots, \rho(y_{\max(0,k-m)})), \\
&\quad P_{m-1}(y_{\max(1,k-m+1)}), \dots, P_{m-1}(y_k) \rangle \\
P_m(\langle x_1, \dots, x_n \rangle) &\rightarrow_a \langle P_{m-1}(x_1), \dots, P_{m-1}(x_n) \rangle \text{ if } x_i \neq *(z_1, \dots, z_k) \text{ and } n \leq 2m \\
P_m(\langle x_1, \dots, x_n \rangle) &\rightarrow_a \\
&\quad \langle P_{m-1}(x_1), \dots, P_{m-1}(x_m), *(\rho(x_{m+1}), \dots, \rho(x_{n-m})), P_{m-1}(x_{n-m+1}), \dots, P_{m-1}(x_n) \rangle \\
&\quad \text{if } x_i \neq *(z_1, \dots, z_k) \text{ and } n > 2m \\
C_{r,s}(\langle x_1, \dots, x_i, *(z_1, \dots, z_j), y_1, \dots, y_k \rangle) &\rightarrow_a \\
&\quad \langle x_1, \dots, x_{\min(i,r)}, \\
&\quad \quad *(\rho(x_{1+\min(i,r)}), \dots, \rho(x_i), z_1, \dots, z_j, \rho(y_1), \dots, \rho(y_{\max(0,k-s)})), \\
&\quad y_{\max(1,k-s+1)}, \dots, y_k \rangle \\
C_{r,s}(\langle x_1, \dots, x_i \rangle) &\rightarrow_a \langle x_1, \dots, x_r, *(\rho(x_{r+1}), \dots, \rho(x_{i-s})), x_{i-s+1}, \dots, x_i \rangle \\
&\quad \text{if no } x_j = *(z_1, \dots, z_k) \text{ and } i > r+s \\
P_m(al_l: \langle y, z \rangle) &\rightarrow_a al_l: \langle P_{m-1}(y), C_{m-1,m}(P_m(z)) \rangle \\
P_m(ar_l: \langle y, z \rangle) &\rightarrow_a ar_l: \langle C_{m,m-1}(P_m(y)), P_{m-1}(z) \rangle \\
C_{i+1,j}(al_l: \langle y, z \rangle) &\rightarrow_a al_{\max(l,i+2,j+2)}: \langle y, C_{i,j}(z) \rangle \\
C_{i,j+1}(ar_l: \langle y, z \rangle) &\rightarrow_a ar_{\max(l,i+2,j+2)}: \langle C_{i,j}(y), z \rangle \\
P_m(y):z &\rightarrow_a P_{m-1}(y):z
\end{aligned}$$

Bottom rules:

$$\begin{aligned}
\Omega(x) &\rightarrow_a \Omega(y) \text{ if } x \rightarrow y \\
\langle x_1, \dots, *(z_1, \dots, z_i, \dots, z_k), \dots, x_n \rangle &\rightarrow_a \langle x_1, \dots, *(z_1, \dots, z'_i, \dots, z_k), \dots, x_n \rangle \text{ if } z_i \rightarrow z'_i
\end{aligned}$$

Figure 2: Lowering rules, *P*-rules, and bottom rules for FP' .

the *C* construct to the language. Also note that the definition of τ for *al* and *ar* is necessary to ensure that the type does not increase when this rule is applied. $\square$

The following proposition assures us that an FP' expression can always be rewritten to a well-formed expression in at most finitely many steps. Recall that $|x|$ denotes the length of an FP expression viewed as a string of symbols. We extend this notation to FP' expressions

of the first lowering rule and any lowering or P -rule that explicitly mentions a C , which may lower type. Thus, a problem may arise only if we apply one of these rules that may lower type to some subexpression of x'_1 . But it is also easy to show by induction on structure that if $x'_1 \rightarrow_a y'_1$ as a result of using one of these rules, then either $\tau(x'_1) = \tau(y'_1)$ or $\mu'(x'_1)$ is not a function item, which means that $\mu'(y'_1)$ is not a function item either, so that $\mu'(y'_1) = \perp$.

□

The following proposition assures us that in a precise sense, FP' rewriting simulates FP rewriting.

Proposition 5.7.

1. If x' and y' are FP' expressions such that $x' \rightarrow_a^* y'$, then $\rho(x') \rightarrow^* \rho(y')$.
2. If x is an FP redex and x' is a well-formed FP' expression such that $\rho(x') = x$ and $\mu'(x') \neq \perp$, then the FP -style rule corresponding to the FP rule for which x is a redex applies to x'' , where x' has one of the forms x'' , $P_m(x'')$, $C_{p,q}(x'')$, or $C_{p,q}(P_m(x''))$.
3. If x and y are FP expressions such that $x \rightarrow y$ and x' is an FP' expression such that $\rho(x') = x$, then there exists an FP' expression y' such that $\rho(y') = y$ and $x' \rightarrow_a^* y'$ using exactly one FP -style or bottom rule.

Proof. It suffices to show that part (1) holds if $x' \rightarrow_a y'$ is an instance of an FP' rewrite rule. If $x' \rightarrow_a y'$ via a lowering rule or a P -rule, then it is easy to see that $\rho(y') = \rho(x')$, and we can just take $y = x$. For bottom rules the result is also immediate from the definition. If $x' \rightarrow_a y'$ via an FP -style rewrite rule, then it is easy to see that $\rho(x') \rightarrow \rho(y')$ via the corresponding FP rewrite rule.

For part (2), we check all the rewrite rules on a case-by-case basis. For example, suppose $\rho(x') = \text{comp} : \langle f, g \rangle : y$ and $\mu'(x') \neq \perp$. *A priori*, it is possible that x' is of the form $\text{comp}_k : P_m(\langle f, g \rangle : y)$, so the comp FP -style rule does not apply. However, if x' were of this form, then it would not be well-formed.

For part (3), it suffices to show the result in case $x \rightarrow y$ is an instance of an FP rewrite rule. If $\mu(x') = \perp$, then $x' = \Omega(x)$ or $x \rightarrow_a \Omega(x)$ via a lowering rule, and $\Omega(x) \rightarrow_a \Omega(y)$ via a bottom rule. And if $\mu(x') \neq \perp$, then we can rewrite x' to a well-formed expression by Proposition 5.6. If x'' is the resulting expression, then we clearly have $\rho(x'') = \rho(x')$. The result is now immediate from part (2). □

FP' is not strongly normalizable or even weakly normalizable. Consider, for example, $\Omega(x)$, where x is any FP expression that is not weakly normalizable. Since $x \rightarrow y$ iff $\Omega(x) \rightarrow_a \Omega(y)$, we must have that $\Omega(x)$ is not weakly normalizable. However, it turns out that this is essentially the only counterexample to the strong normalizability of FP' . Proposition 5.6 already shows that lowering rules and P -rules by themselves cannot be applied infinitely often to an FP' expression. In the remainder of this section, we will show that

involve two arguments. Consider an expression such as $\text{apply}_4: \langle P_2(\text{comp}_7: \langle f, g \rangle), a_1 \rangle$. The apply redex is the only one contained in the expression, and it has degree 4. But after rewriting this and applying a P -rule, we end up with the expression $P_1(\text{comp}_7: \langle f, g \rangle : a_1)$, which has a redex of degree 7!

It is possible, with some effort, to get a proof in the same spirit as the typed λ -calculus weak normalization proof which shows that every FP' expression can be reduced to one in essential normal form. We do not present this proof here. Instead, we will prove a stronger result, showing that every FP' expression is *essentially strongly normalizable* where:

Definition 5.9. An FP' expression is *essentially strongly normalizable* (e.s.n.) iff every reduction sequence for it has at most finitely many FP -style rules.

Remark: We cannot hope to show that every reduction sequence leads to an expression in e.n.f. To see this, suppose that x is an FP expression which is not weakly normalizable. Then starting with an expression of the form $\text{first}_n: \langle a, \Omega(x) \rangle$, we can keep applying bottom rules to the expression $\Omega(x)$ without ever applying the obvious FP -style rule. This gives us an infinite reduction sequence that does not lead to an expression in e.n.f.

Theorem 5.10. Every FP' expression is *essentially strongly normalizable*.

Proof. Not surprisingly, we prove our result by adapting techniques used to show that typed λ -calculus is strongly normalizable. The original proof of strong normalizability for typed λ -calculus is due to Prawitz [Pr71]; our proof uses ideas similar to those in Tait [Ta67, Ta75]. (Actually our treatment follows the proof that the (first-order) λ -calculus is strongly normalizable as given in [FLO83], which in turn is based on the work of Tait.) However, just as in the case of weak normalizability, adapting the proof to FP requires some effort, mainly due to the greater structure of FP rewrite rules. The details can be found in Appendix C. $\square$

Corollary 5.11. For all FP' expressions x' , there exists an FP' expression y' such that $x' \rightarrow_a^* y'$ and y' is in essential normal form.

Proof. We first show that for every FP' expression z'_0 there exists an FP' expression z'_1 such that $z'_0 \rightarrow_a^* z'_1$ and either z'_1 is in e.n.f. or the reduction used at least one FP -style rule. By Proposition 5.6, $z'_0 \rightarrow_a^* z''$ for some well-formed z'' . If z'' is in e.n.f., then the claim follows by taking $z'_1 = z''$. If z'' is not in e.n.f., then z'' contains an FP -style redex (since it has a non-bottom rule redex and no P -rule or lowering rule redexes) and so $z'' \rightarrow_a z'_1$ by an FP -style rule and again the claim follows. Since Theorem 5.10 implies that no rewrite sequence has infinitely many reductions involving FP -style rules, it easily follows from the claim that we can eventually rewrite every FP' expression x' to an expression y' in e.n.f. $\square$

Definition 6.1. A *rewrite strategy* R is a function $R : (\text{FP expressions}) \times \mathbb{N} \rightarrow (\text{FP expressions})$ such that $R(x, 0) = x$ for all FP expressions x , and for all n , either $R(x, n) \rightarrow R(x, n+1)$ or $R(x, n) = R(x, n+1)$. Thus $R(x, n)$ describes the n^{th} step of rewriting when starting with x according to strategy R . We write $x \rightarrow_R^* y$ iff for some n we have $R(x, n) = y$.

We say R is *history independent* iff, intuitively, it performs the same rewriting when it reaches an expression y starting from x as it does starting from y . More formally, R is history independent iff for all x and n , if $R(x, n) = y$ and $R(y, 1) = y'$, then $R(x, n+1) = y'$. Note that if R is history independent, then R is completely determined once we give $R(x, 1)$ for all FP expressions x .

R is said to *stop prematurely* if for some x and n , we have $R(x, n)$ not in normal form, but $R(x, n') = R(x, n)$ for all $n' \geq n$. If x is strongly normalizable, then it is easy to see that any rewrite strategy that does not stop prematurely will reduce x to normal form.

In obvious analogy to observable completeness, we say a strategy R is *O-safe* iff for any FP expression x such that $\mu(x)$ is an observable item, $x \rightarrow_R^* \mu(x)^e$. We also want an analogue to strong completeness. It would seem that the obvious definition is to say that R is *S-safe* iff $x \rightarrow_R^* y$ for some transparent expression y . Unfortunately, with this definition we do not get an analogue to Proposition 3.6; S-safety would not imply O-safety. For example, suppose we have an expression x such that $\mu(x) = \langle a \rangle$. For an O-safe strategy R , we have $x \rightarrow_R^* \langle a \rangle$. But if we define S-safe as suggested above, then if R is an "S-safe" strategy, we can conclude $x \rightarrow_R^* \langle y \rangle$ where $\mu(y) = a$, and that $y \rightarrow_R^* a$. However, we have no way of concluding that $x \rightarrow_R^* \langle a \rangle$, since, for example, R may stop prematurely at $\langle y \rangle$.

Intuitively, the problem is that although transparency guarantees that $x \rightarrow_R^* \langle y \rangle$, there is no guarantee that y itself is transparent. These observations motivate the following definition.

Definition 6.2.

1. An FP expression x is *1-transparent* iff it is transparent.
2. An FP expression x is *(n+1)-transparent* iff x is transparent and, if x is of the form $\langle x_1, \dots, x_k \rangle$, then x_i is n -transparent for $i = 1, \dots, k$, and if x is of the form $y : z$ and $\mu(x) \neq \perp$, then y, z are both n -transparent.

Definition 6.3. R is *S-safe for* x iff for all n , there exists an n -transparent y such that $x \rightarrow_R^* y$. R is *S-safe* iff R is S-safe for all FP expressions x .

An argument by induction on the depth of nesting of brackets, analogous to Proposition 3.6, can be used to show that

Proposition 6.4. Every S-safe rewrite strategy is also O-safe.

We remark that it follows by an easy induction on n that an expression in normal form is n -transparent for all n (using Theorem 4.3), and that for all x , there is some n -transparent y such that $x \rightarrow_a^* y$ (using Theorem 5.13). Thus, we can conclude that there does exist a (not

necessarily recursive) S-safe strategy. In the remainder of this section we will show that there are also some natural (and recursive) S-safe strategies.

Observe that the two rewrite strategies perhaps most often considered in the literature, *leftmost-outermost* and *leftmost-innermost* (cf. [MNV73]), are *not* S-safe; in fact, they are not even O-safe. Consider the leftmost-outermost strategy, LO . LO is history independent and $LO(x, 1)$ is the result of applying the rewrite rule corresponding to the redex which starts lexically leftmost in x . To see that LO is not O-safe, let x be the expression $tl:(al: <fix:id, id: <a > >)$. It is easy to check that the meaning of this expression is the observable $\langle a \rangle$. It is also easy to see that for all n , $LO(x, n)$ is of the form $tl:(al: <y, id: <a > >)$, where y is the result of rewriting $fix:id$. Therefore the leftmost-outermost strategy does not rewrite this expression to $\langle a \rangle$, and hence is not O-safe.

The leftmost-innermost strategy chooses the leftmost redex which does not have another redex as a subexpression. The same example used above also shows that leftmost-innermost is not O-safe either.

It is interesting to note that we can modify the leftmost-outermost strategy to get a natural rewrite strategy which is not S-safe but is O-safe.

Definition 6.5. The *modified leftmost strategy* ML is history independent. Thus, in order to describe it, we need only give $ML(x, 1)$ for each FP expression x . If x is in normal form, then clearly $ML(x, 1) = x$. Otherwise we compute $ML(x, 1)$ by taking the first rule in the list below that applies:

1. if x is itself a redex, then $ML(x, 1)$ is (the unique) y such that $x \rightarrow y$ is an instance of a rewrite rule
2. if $x = <x_1, \dots, x_n>$, then $ML(x, 1) = <x_1, \dots, x_{i-1}, ML(x_i, 1), x_{i+1}, \dots, x_n>$ where i is such that x_i is not in normal form and x_j is in normal form for all $j > i$
3. if $x = y:z$ and y is not in normal form, then $ML(x, 1) = ML(y, 1):z$
4. if $x = al: <x_1, x_2>$ and x_2 is not in normal form, then $ML(x, 1) = al: <x_1, ML(x_2, 1)>$
5. otherwise, x must be of the form $y:z$ with z not in normal form, and we take $ML(x, 1) = y:ML(z, 1)$.

Note that rule 4 is the only one that distinguishes modified leftmost and leftmost-outermost. It is also interesting to note that al and ar are not treated symmetrically in this rule.

It is easy to see that the modified leftmost strategy is not S-safe. Consider the expression $\langle fix:id, id:a \rangle$. The modified leftmost strategy will spend all its time working on $fix:id$, and never get around to rewriting $id:a$ to a . However, we have

Proposition 6.6. The modified leftmost strategy is O-safe.

Proof. See Appendix D. $\square$

if some expression in the tracking sequence is in C.N.F. .

Lemma 6.8. *If R is a rewrite strategy, x is an FP expression, and x' is a well-formed FP' expression such that $p(x') = x$, then there is a tracking sequence for (R, x) starting with x' .*

Proof. Immediate from Proposition 5.7 and Proposition 5.6. $\square$

Theorem 6.9. *The following are equivalent:*

1. R is S-safe for x .
2. Every tracking sequence for (R, x) reaches e.n.f.

Proof. The proof is quite technical, and is presented in Appendix E. $\square$

We next define *fair* strategies, a large class of strategies that are provably S-safe. Intuitively, a fair strategy is one in which every redex either “disappears” or is eventually rewritten. In order to make this precise, we must first describe what it means for a given redex (called the *parent* redex) to have a *child*. Suppose $x \rightarrow y$ and x' is a subexpression of x which is a redex. Then a subexpression y' of y is a child of x' iff it is a redex that “corresponds” to x' in y ; in particular, both x' and y' are instances of the left-hand sides of the same rewrite rule and occupy the same “position” in x and y , respectively. Rather than giving a formal definition of child here, we give a more easily understood informal definition, supplemented by examples.

A child can occur in one of three ways:

1. The parent redex is disjoint from the rewritten redex. For example, if we take $x = \langle \text{id:a}, \text{id:b} \rangle$, $y = \langle \text{id:a}, b \rangle$, and $x' = \text{id:a}$, then x' is disjoint from the rewritten redex id:b , and the occurrence of id:a in y is a child of x' .
2. The rewritten redex is a *subredex* of (i.e. subexpression of the instantiation of a schema variable in the rewrite rule schema of) the parent redex. For example, if we take $x = x' = \text{first} : \langle \text{id:a} \rangle$ and $y = y' = \text{first} : \langle a \rangle$, then the rewritten redex id:a is a subredex of x' and y' is a child of x' .
3. The parent redex is a subredex of the rewritten redex. For example, if $x = \text{cons} : \langle z_1, z_2 \rangle : (\text{id:a})$, $y = \langle z_1 : (\text{id:a}), z_2 : (\text{id:a}) \rangle$, and $x' = \text{id:a}$, then the parent redex x' is a subredex of x . In this case x' has two children in y ; i.e., the two occurrences of id:a .

Note that the rewritten redex never has any children and that in cases 1 and 2, the parent always has exactly one child. Also note, for example, that in the reduction

order. Suppose there are k such redexes. We then rewrite the first redex, then all the descendants of the second redex in the resulting expression, all the descendants of the third redex in that resulting expression, and so on up to the k^{th} redex. If this takes n' steps, then $n_{i+1} = n_i + n'$.

- **cons-last:** The $(i+1)^{\text{st}}$ phase consists of two subphases. In the first subphase we rewrite all the non-cons redexes (i.e., all those not of the form $\text{cons}:x:y$) until there are no more, while in the second subphase we rewrite all the cons redexes until there are no more (at this point, of course, we will in general have non-cons redexes appearing again). We omit the formal definition here.
- **comp-last:** CM is analogous to CN , except that we first rewrite the non-comp redexes, and then rewrite the comp redexes.
- **apply-last:** Again, AP is analogous to CN , except that we first rewrite the non-apply redexes.

It should be clear from the definitions that BF is indeed a fair strategy. The only problem in showing that CN is a fair strategy is that, *a priori*, it is not clear that n_{i+1} is finite; it may be the case that the $(i+1)^{\text{st}}$ phase as defined never terminates. However, from Theorem 4.2, we know that $\text{FP}\{\text{cons}\}$ is strongly normalizable, so it is easy to see that the first subphase terminates in a finite number of steps. And since $\text{FP}\{\text{comp}\}$ is strongly normalizable, it follows that the second subphase also terminates in a finite number of steps. Similar arguments can be used to show that CM and AP are fair strategies. Note that any hybrid of these strategies (for example, one where we alternate between CM and CN in consecutive phases) will also be fair. Also note that a shuffles in one of the above phases infinitely often into another strategy (such as leftmost-outermost) will be fair.

7. Limiting Completeness

The notion of strong completeness we have defined has a strong syntactic component, since it requires that an expression with a certain meaning can be rewritten to another expression with a prescribed syntactic form. We would like to be assured that our rewriting also produces arbitrarily good semantic approximations. To understand this point, consider an expression x whose meaning is the limit of $\langle a, *, b \rangle$, $\langle a, a, *, b, b \rangle$, $\langle a, a, a, *, b, b, b \rangle$, ... Thus, the meaning of x is a double-sided infinite sequence with a 's added from the left and b 's added from the right. Strong completeness assures us that x can be rewritten to an expression of the form $\text{al}: \langle a, x_1 \rangle$ or $\text{ar}: \langle x_1, b \rangle$. Furthermore, it also assures us that in either case, x_1 can be rewritten to an expression of the form $\text{al}: \langle a, x_2 \rangle$ or $\text{ar}: \langle x_2, b \rangle$. We can continue this argument indefinitely. However, it is consistent with the definition of strong completeness that we can only rewrite x to an expression of the form $\text{al}: \langle a, \text{al}: \langle a, \text{al}: \dots \rangle \rangle$, without ever getting information about the right-hand sequence of b 's in $\mu(x)$. This seems somehow unsatisfactory. To handle this phenomenon, we are led to the notion of *limiting completeness*, which is the FP analogue to the notion with the same name defined by Wadsworth for λ -calculus (cf. [Wa78]). Of course, once we have defined limiting completeness, we will

rewriting strategy R is *L-safe* for x iff for all $d \in \mu(x)$, there exists n such that $d \in \mu_r(R(x, n))$.
 R is *L-safe* iff it is L-safe for all FP expressions x .

Remark: There is one intuition behind the notion of limiting completeness in the λ -calculus which is *not* captured by our definition of μ_r . To explain our point, we need a little notation. If M is a λ -calculus expression, let M_r be the result of replacing all redexes in M by Ω and then applying the rewrite rules $\Omega x \rightarrow \Omega$ and $\lambda x. \Omega \rightarrow \Omega$ until there are no more redexes of this form. If $M \rightarrow^* N$, then it is easy to see that M_r and N_r have the same structure and N_r is more defined than M_r ; i.e., N_r looks just like M_r except that some Ω 's in M_r may be replaced by other, more completely defined expressions. This is *not* the case in FP because of the greater structure of the rewrite rules. For example, consider the expression $x = \text{al} : \langle a, \text{id} : \langle a \rangle \rangle$. The only redex here is $\text{id} : \langle a \rangle$; thus $\mu_r(x) = \text{al} : \langle a, \perp \rangle = \langle a, * \rangle$. Had we taken the route in FP of replacing redexes by Ω , we would get $x_r = \text{al} : \langle a, \Omega \rangle$. However, x rewrites to $y = \text{al} : \langle a, \langle a \rangle \rangle$, which is a redex, so $\mu_r(y) = \perp$ and $y_r = \Omega$. Thus y_r is *less* defined than x_r . Moreover, y rewrites to $z = \langle a, a \rangle$, and clearly $z_r = z$. Thus z_r has a different structure from x_r , although $x \rightarrow^* z$. The root cause of this situation is that, due to the greater structure of FP rewrite rules, it is possible that $x \rightarrow y$ but $\mu_r(y) <_I \mu_r(x)$. In order to avoid this situation, we must not only replace redexes by Ω , but also *potential redexes* (expressions that rewrite to redexes). This leads us to define a semantic function μ_{pr} , which is just like μ_r except that it maps potential redexes to $\perp$ (the *pr* stands for *potential redex*). We investigate μ_{pr} further in Appendix G. While μ_r and μ_{pr} are distinct, similar definitions in the λ -calculus would not result in distinct notions because of the simpler structure of the β -rule.

Note that limiting completeness is just what we need to show that if the meaning of x is a double-sided infinite sequence with a 's added from the left and b 's added from the right, as in the example at the beginning of this section, we can rewrite x to an expression with arbitrarily many a 's and arbitrarily many b 's.

Theorem 7.4. *FP has the limiting completeness property.*

Proof. Given an FP expression x and $d \in \mu(x)$, by Proposition 5.2, we can find an FP' expression x' such that $\rho(x') = x$ and $d \leq_I \mu'(x)$. By Theorem 5.10, we can find y' in e.n.f. such that $x' \rightarrow_a^* y'$. Since $\rightarrow_a$ rewriting is sound, we have $\mu'(y') = \mu'(x')$, so that $d \leq_I \mu'(y')$. Since y' is in e.n.f., by part 3 of Lemma 7.1, we know that $\mu'(y') \in \mu_r(\rho(y'))$, so that $d \in \mu_r(\rho(y'))$. Finally, by Proposition 5.7, $x \rightarrow^* \rho(y')$. $\square$

Theorem 7.5. *L-safety implies O-safety.*

Proof. Suppose R is L-safe and $\mu(x)$ is observable. Thus $\mu(x) = \{x\}^d$ for some $x \in D$ and since R is L-safe, there must be some y such that $x \rightarrow_R^* y$ and $x \in \mu_r(y)$. An easy induction on the number of brackets ($\langle \rangle$'s) in x , using Proposition 7.2, can now be used to show that $y = x^e$. Thus $x \rightarrow^* x^e$, and R is O-safe. $\square$

8. Conclusions

We have answered the two questions posed in the introduction. We have given a denotational semantics for FP where distinct observable expressions get different meanings, shown that the rewrite rules are sound for that semantics, and shown that they are complete in three different senses. For each of our notions of completeness, we have defined a corresponding notion of a “good” rewrite strategy. We have characterized and compared these notions, and given natural examples of good strategies. While we have proved these results in the context of FP, we believe that our techniques will readily apply to other languages.

Throughout this paper we have not discussed, nor have we needed to discuss, the Church-Rosser property for FP. (CR would say that for all FP expressions x , y_1 , and y_2 , if $x \rightarrow^* y_1$ and $x \rightarrow^* y_2$, then there exists z such that $y_1 \rightarrow^* z$ and $y_2 \rightarrow^* z$.) We conjecture that FP with the particular set of rewrite rules we have described is in fact CR, but we have not yet proved this. But even if we had proved CR, it is not clear how interesting it would be.

To understand this point, first note that CR for expressions whose meaning is observable follows immediately from our results on observable completeness. Indeed, observable completeness assures us that CR holds for observables in the following strong sense: We can add arbitrarily many new (sound) rules to the system and still be assured that expressions with observable meanings are CR. This is because the addition of rules necessarily preserves observable completeness (as well as the other properties we have considered), and observable completeness implies CR for expressions with observable meanings.

Our feeling is that any basic property of the language should be *robust*, in the sense that it is preserved under the addition of sound rewrite rules. But consider what happens when we add the following two (sound) rules to our system:

$$\text{comp: } \langle \text{comp: } \langle f, g \rangle, \text{comp: } \langle h, i \rangle \rangle \rightarrow \text{comp: } \langle f, \text{comp: } \langle g, \text{comp: } \langle h, i \rangle \rangle \rangle$$

$$\text{comp: } \langle \text{comp: } \langle f, g \rangle, \text{comp: } \langle h, i \rangle \rangle \rightarrow \text{comp: } \langle \text{comp: } \langle \text{comp: } \langle f, g \rangle, h \rangle, i \rangle$$

(Note that the soundness of these rules follows from the associativity of composition; it is perhaps more easily seen if we rewrite the rules using infix notation as $(f \circ g) \circ (h \circ i) \rightarrow f \circ (g \circ (h \circ i))$ and $(f \circ g) \circ (h \circ i) \rightarrow ((f \circ g) \circ h) \circ i$ respectively.) It is easy to see that if f, g, h , and i are replaced by primitive function symbols, then both of the above right hand sides are themselves in normal form. Thus, even if CR held for the system using the rewrite rules described in Figure 1, it would not be a robust property.

While the language we have presented is universal, it is by no means sufficient as a useful programming language. There are many more functions to be added, some defined in terms of the primitives given and some as new primitives, and there are useful extensions that could be made to some of our functions. Although the addition of new primitives will necessarily make our proofs longer (there being more cases to consider and rules to check), we believe that the structure of the proofs is such that it will be relatively easy to make modifications

9. Appendix A: An Alternative Syntax

In the initial stages of extending Backus' FP78 to include infinite sequences, a syntactic design decision had to be made: what syntactic object would be used to represent the result of appending something to the left of a non-sequence? E.g., to what would $al: \langle a, b \rangle$ rewrite? Of course, whatever was chosen had to have the correct properties; e.g., first of it had to yield a , tl of it had to yield $\perp$, and appending c to the right of it had to satisfy all of the properties of the semantic item $\langle a, *, c \rangle$.

Rather than burden the syntax with a special construction, it was decided simply to make $al: \langle a, b \rangle$ a normal form, i.e. an expression to which no rewrite rules apply, and to design the rewrite rules for $first$, tl , $last$, and tr accordingly. This design decision was responsible for the unusual structure of some of the rules and for a small amount of the technical complexity in the statements (and proofs) of the results of this paper.

An alternative approach is to include another syntactic construct σ , which may appear only inside of sequences. Formally, if $x_1, \dots, x_n, y$ are expressions, then

$$\begin{aligned}
 &\langle x_1, \dots, x_i, \sigma(\langle y_1, \dots, y_j \rangle), z_1, \dots, z_k \rangle \rightarrow \langle x_1, \dots, x_i, y_1, \dots, y_j, z_1, \dots, z_k \rangle \\
 &al: \langle x, y \rangle \rightarrow \langle x, \sigma(y) \rangle \quad \text{if } x, y \neq \sigma(z) \\
 &ar: \langle x, y \rangle \rightarrow \langle \sigma(x), y \rangle \quad \text{if } x, y \neq \sigma(z) \\
 &first: \langle x_1, \dots, x_n \rangle \rightarrow x_1 \quad \text{if } n \geq 1 \text{ and } x_1 \neq \sigma(y) \\
 &last: \langle x_1, \dots, x_n \rangle \rightarrow x_n \quad \text{if } n \geq 1 \text{ and } x_n \neq \sigma(y) \\
 &tl: \langle x_1, \dots, x_n \rangle \rightarrow \langle x_2, \dots, x_n \rangle \quad \text{if } n \geq 1, x_1 \neq \sigma(y), \text{ and } \langle x_2, \dots, x_n \rangle \neq \langle \sigma(z) \rangle \\
 &tl: \langle \sigma(x_1), x_2, \dots, x_n \rangle \rightarrow \langle \sigma(tl: \langle \sigma(x_1), x_2 \rangle), x_3, \dots, x_n \rangle \quad \text{if } n \geq 3 \\
 &tr: \langle x_1, \dots, x_n \rangle \rightarrow \langle x_1, \dots, x_{n-1} \rangle \quad \text{if } n \geq 1, x_n \neq \sigma(y), \text{ and } \langle x_1, \dots, x_{n-1} \rangle \neq \langle \sigma(z) \rangle \\
 &tr: \langle x_1, \dots, x_{n-1}, \sigma(x_n) \rangle \rightarrow \langle x_1, \dots, x_{n-2}, \sigma(tr: \langle x_{n-1}, \sigma(x_n) \rangle) \rangle \quad \text{if } n \geq 3 \\
 &null: \langle \rangle \rightarrow T \\
 &null: a \rightarrow F \quad (\text{for any atom } a) \\
 &null: \langle x_1, \dots, x_n \rangle \rightarrow F \quad \text{if } n \geq 1 \\
 &(cons: \langle f_1, \dots, f_n \rangle):x \rightarrow \langle f_1:x, \dots, f_n:x \rangle \quad \text{if no } f_i = \sigma(g) \\
 &(cons: \langle f_1, \dots, f_m, \sigma(g), f_{m+1}, \dots, f_n \rangle):x \rightarrow \langle f_1:x, \dots, f_m:x, \sigma(cons:g:x), f_{m+1}:x, \dots, f_n:x \rangle \\
 &cond: \langle T, x, y \rangle \rightarrow x \quad \text{if } x, y \neq \sigma(z) \\
 &cond: \langle F, x, y \rangle \rightarrow y \quad \text{if } x, y \neq \sigma(z) \\
 &(comp: \langle f, g \rangle):x \rightarrow f:(g:x) \quad \text{if } f, g \neq \sigma(z) \\
 &apply: \langle f, x \rangle \rightarrow f:x \quad \text{if } f, x \neq \sigma(z) \\
 &(K:x):y \rightarrow x \\
 &id:x \rightarrow x
 \end{aligned}$$

Figure 3: The FP rewrite rules for the alternative syntax.

These rules apply only if the left-hand side is well-formed.

$$\begin{aligned}
& \text{al}_p: \langle x, \langle y_1, \dots, y_n \rangle \rangle \rightarrow_a \langle x, y_1, \dots, y_n \rangle \\
& \text{al}_p: \langle x, \Omega(\langle y_1, \dots, y_n \rangle) \rangle \rightarrow_a \langle x, * (y_1, \dots, y_n) \rangle \\
& \text{ar}_p: \langle \langle y_1, \dots, y_n \rangle, x \rangle \rightarrow_a \langle y_1, \dots, y_n, x \rangle \\
& \text{ar}_p: \langle \Omega(\langle y_1, \dots, y_n \rangle), x \rangle \rightarrow_a \langle * (y_1, \dots, y_n), x \rangle \\
& \text{first}_p: \langle x_1, \dots, x_n \rangle \rightarrow_a x_1 \quad (n \geq 1) \\
& \text{first}_p: (\text{al}_r: \langle x, y \rangle) \rightarrow_a x \\
& \text{first}_p: \bar{C}_{q,0}(\text{ar}_r: \langle \text{al}_s: \langle x, y \rangle, z \rangle) \rightarrow_a x \\
& \text{first}_p: \bar{C}_{q,0}(\text{ar}_r: \langle \bar{C}_{s,0}(\text{ar}_t: \langle x, y \rangle), z \rangle) \rightarrow_a \text{first}_p: (\text{ar}_t: \langle x, y \rangle) \\
& \text{last}_p: \langle x_1, \dots, x_n \rangle \rightarrow_a x_n \quad (n \geq 1) \\
& \text{last}_p: (\text{ar}_q: \langle x, y \rangle) \rightarrow_a y \\
& \text{last}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, \text{ar}_s: \langle y, z \rangle \rangle) \rightarrow_a z \\
& \text{last}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, \bar{C}_{0,s}(\text{al}_t: \langle y, z \rangle) \rangle) \rightarrow_a \text{last}_p: (\text{al}_t: \langle y, z \rangle) \\
& \text{tl}_p: \langle x_1, \dots, x_n \rangle \rightarrow_a \langle x_2, \dots, x_n \rangle \quad (n \geq 1, x_1 \neq * (y_1, \dots, y_k)) \\
& \text{tl}_p: \langle * (y_1, \dots, y_k), x_2, \dots, x_n \rangle \rightarrow_a \langle * (y_2, \dots, y_k, \rho(x_2)), x_3, \dots, x_n \rangle \quad (n \geq 3, k \geq 1) \\
& \text{tl}_p: \langle * (), x_2, \dots, x_n \rangle \rightarrow_a \langle * (), x_3, \dots, x_n \rangle, \quad (n \geq 3) \\
& \text{tl}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, \bar{C}_{0,s}(\text{al}_t: \langle y, z \rangle) \rangle) \rightarrow_a \bar{C}_{0,q-1}(\bar{C}_{0,s}(\text{al}_t: \langle y, z \rangle)) \\
& \text{tl}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, \bar{C}_{s,0}(\text{ar}_t: \langle y, z \rangle) \rangle) \rightarrow_a \bar{C}_{0,q-1}(\bar{C}_{s,0}(\text{ar}_t: \langle y, z \rangle)) \\
& \text{tl}_p: \bar{C}_{q,0}(\text{ar}_r: \langle \bar{C}_{0,s}(\text{al}_t: \langle x, y \rangle), z \rangle) \rightarrow_a \bar{C}_{q-1,0}(\bar{C}_{0,s}(\text{ar}_t: \langle y, z \rangle)) \\
& \text{tl}_p: \bar{C}_{q,0}(\text{ar}_r: \langle \bar{C}_{s,0}(\text{ar}_t: \langle x, y \rangle), z \rangle) \rightarrow_a \bar{C}_{q-1,0}(\bar{C}_{s-1,1}(\text{ar}_t: \langle \text{tl}_p: (\text{ar}_t: \langle x, y \rangle), z \rangle)) \\
& \text{tr}_p: \langle x_1, \dots, x_n \rangle \rightarrow_a \langle x_1, \dots, x_{n-1} \rangle \quad (n \geq 1, x_n \neq * (y_1, \dots, y_k)) \\
& \text{tr}_p: \langle x_1, \dots, x_{n-1}, * (y_1, \dots, y_k) \rangle \rightarrow_a \langle x_1, \dots, x_{n-2}, * (\rho(x_{n-1}), y_1, \dots, y_{k-1}) \rangle \quad (n \geq 3, k \geq 1) \\
& \text{tr}_p: \langle x_1, \dots, x_{n-1}, * () \rangle \rightarrow_a \langle x_1, \dots, x_{n-2}, * () \rangle, \quad (n \geq 3) \\
& \text{tr}_p: \bar{C}_{q,0}(\text{ar}_r: \langle \bar{C}_{0,s}(\text{al}_t: \langle x, y \rangle), z \rangle) \rightarrow_a \bar{C}_{q-1,0}(\bar{C}_{0,s}(\text{al}_t: \langle x, y \rangle)) \\
& \text{tr}_p: \bar{C}_{q,0}(\text{ar}_r: \langle \bar{C}_{s,0}(\text{ar}_t: \langle x, y \rangle), z \rangle) \rightarrow_a \bar{C}_{q-1,0}(\bar{C}_{s,0}(\text{ar}_t: \langle x, y \rangle)) \\
& \text{tr}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, \bar{C}_{s,0}(\text{ar}_t: \langle y, z \rangle) \rangle) \rightarrow_a \bar{C}_{0,q-1}(\bar{C}_{s,0}(\text{al}_t: \langle x, y \rangle)) \\
& \text{tr}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, \bar{C}_{0,s}(\text{al}_t: \langle y, z \rangle) \rangle) \rightarrow_a \bar{C}_{0,q-1}(\bar{C}_{1,s-1}(\text{al}_t: \langle x, \text{tr}_p: (\text{al}_t: \langle y, z \rangle) \rangle)) \\
& \text{null}_p: \langle \rangle \rightarrow_a T \\
& \text{null}_p: a \rightarrow_a F \quad (\text{for any atom } a) \\
& \text{null}_p: \langle x_1, \dots, x_n \rangle \rightarrow_a F \quad (n \geq 1) \\
& \text{null}_p: \bar{C}_{0,q}(\text{al}_r: \langle x, y \rangle) \rightarrow_a F \\
& \text{null}_p: \bar{C}_{q,0}(\text{ar}_r: \langle x, y \rangle) \rightarrow_a F \\
& (\text{cons}_p: \langle f_1, \dots, f_n \rangle): x \rightarrow_a P_{p-2}(\langle f_1: x, \dots, f_n: x \rangle), \quad (f_i \neq * (g_1, \dots, g_k)) \\
& (\text{cons}_p: \langle f_1, \dots, * (g_1, \dots, g_k), \dots, f_n \rangle): x \rightarrow_a P_{p-2}(\langle f_1: x, \dots, * (g_1: \rho(x), \dots, g_k: \rho(x)), \dots, f_n: x \rangle), \quad n \geq 1 \\
& (\text{cons}_p: \bar{C}_{0,q}(\text{al}_r: \langle f, g \rangle)): x \rightarrow_a P_{p-2}(\bar{C}_{0,q}(\text{al}_r: \langle f: x, \text{cons}_p: g: x \rangle)) \\
& (\text{cons}_p: \bar{C}_{q,0}(\text{ar}_r: \langle f, g \rangle)): x \rightarrow_a P_{p-2}(\bar{C}_{q,0}(\text{ar}_r: \langle \text{cons}_p: f: x, g: x \rangle)) \\
& \text{cond}_p: \langle T, x, y \rangle \rightarrow_a x \\
& \text{cond}_p: \langle F, x, y \rangle \rightarrow_a y \\
& (\text{comp}_p: \langle f, g \rangle): x \rightarrow_a f: (g: x) \\
& \text{apply}_p: \langle f, x \rangle \rightarrow_a f: x \\
& (K_p: x): y \rightarrow_a P_{p-2}(x) \\
& \text{id}_p: x \rightarrow_a x
\end{aligned}$$

Figure 4: The FP-style rewrite rules

for $i \geq N$. Let ρ' be defined in the same way as ρ except that $\rho'(\Omega(t)) = \Omega$. Thus ρ' maps FP' expressions to FP expressions augmented with Ω . It is now easy to check that $|\rho'(x_i)| > |\rho'(x_{i+1})|$ for $i \geq N$. This gives us the desired contradiction. (We remark that it is not necessarily the case that $|\rho(x_i)| > |\rho(x_{i+1})|$ for $i \geq N$ since a bottom rule of the form $\Omega(t) \rightarrow_a \Omega(t')$ might have been used in rewriting $x_i \rightarrow_b^* y_i$.) $\square$

Definition 11.4. x is an *immediate component* of z iff one of the following holds:

1. $z = \langle z_1, \dots, z_i, * (t_1, \dots, t_k), z_{i+1}, \dots, z_n \rangle$ and $x = z_j$ for some $j = 1, \dots, n$
2. $z = \langle z_1, \dots, z_n \rangle$ and $x = z_j$ for some $j = 1, \dots, n$
3. $z = \bar{C}_{0,q}(ar: \langle z_1, z_2 \rangle)$ and $x = z_1$ or $x = z_2$ (recall from Appendix B that $\bar{C}_{0,q}(x)$ is represents either $C_{0,q}(x)$ or just x .)
4. $z = \bar{C}_{q,0}(ar: \langle z_1, z_2 \rangle)$ and $x = z_1$ or $x = z_2$.

x is a *component* of y iff x is an immediate component of z for some well-typed z such that $y \rightarrow_b^* z$. Note that if $\mu'(y)$ is not a sequence item, then y has no components.

Lemma 11.5. *If x is a component of y , then $x <_{se} y$ and $\tau(x) \leq \tau(y)$. Furthermore, if y is e.s.n., then x is e.s.n. also.*

Proof. Suppose the hypotheses of the lemma hold. Then there exists a well-formed z such that $y \rightarrow_b^* z$ and x is an immediate component of z . The expression z must have one of the four forms listed in Definition 11.4. We will consider only the third case and leave the other cases to the reader. In this case, $z = \bar{C}_{0,q}(ar: \langle z_1, z_2 \rangle)$ and x is either z_1 or z_2 . Clearly $\tau(x) \leq \tau(z) \leq \tau(y)$ and x is a very proper subexpression of z . Hence, $x <_{se} y$ and $\tau(x) \leq \tau(y)$, as desired. If y is e.s.n., then x is e.s.n. by Proposition 11.3. $\square$

We are now ready to define stability.

Definition 11.6. An FP' expression x is *stable* iff it is e.s.n. with respect to $\rightarrow_b$ and both of the following hold:

1. if $\mu'(x)$ is a sequence item, then every component of x is stable
2. if $\mu'(x)$ is a function item, then $x:y$ is stable for all stable y with $\tau(y) < \tau(x)$.

Note that the stability is well-defined, since the stability of x is defined in terms of expressions y such that either $\tau(y) < \tau(x)$ or $\tau(y) = \tau(x)$, $y <_{se} x$, with both x, y being e.s.n. (Here we use the fact that $<_{se}$ is well founded over the set of e.s.n. expressions.)

Lemma 11.7. *If x is stable, then every component of x is stable.*

Proof. If $\mu'(x)$ is a sequence item, then the result follows from the definition of stability. If $\mu'(x)$ is not a sequence item, then the result is vacuously true since x has no components. $\square$

Proposition 11.8. *If $x \rightarrow_b^* y$ and x is stable, then y is stable.*

Proof. We prove the result by induction on $\tau(x)$. Clearly, y is e.s.n. If $\mu'(y)$ is not a

assumption, z_{i-1} is of the form $Q(x_{i-1};y_{i-1})$. If z_i is of the form $Q(x_i;y_i)$ and $x_{i-1} \rightarrow_b^* x_i$ and $y_{i-1} \rightarrow_b^* y_i$, then (since clause (1) does not hold for z_i) it must be the case that z_i is well-typed and clause (2) holds for z_i . If this is not the case, then either (a) an FP-style rule or the last P -rule was applied to $x_{i-1};y_{i-1}$ in rewriting $z_{i-1} \rightarrow_b z_i$, or (b) the first lowering rule was used in rewriting z_{i-1} to z_i . We cannot have (a), since in this case $x_{i-1};y_{i-1}$ would have to be well-typed, contradicting our choice of i . If (b) holds, then it is easy to see that clause (3) applies to the sequence. $\square$

Definition 11.13. x is *semi-stable* if x is e.s.n. and all the well-typed y such that $x \rightarrow_b^* y$ are stable.

It is immediate from the definition that if x is semi-stable and $x \rightarrow_b^* x'$ then x' is semi-stable.

Proposition 11.14. *If x and y are semi-stable, then $x;y$ is semi-stable.*

Proof. Consider any rewriting sequence $z_1 \rightarrow_b z_2 \rightarrow_b \dots$, where $z_1 = x;y$. We first show that if z_k is well-typed, then z_k is stable. Suppose that z_k is well-typed. From Lemma 11.12, we know that there is an $i \leq k$ such that either (a) z_i is of the form $Q(x';y')$, where $x';y'$ is well-typed, $x \rightarrow_b^* x'$, and $y \rightarrow_b^* y'$ or (b) $\Omega(\rho(z_1)) \rightarrow_b^* z_i$. We now show that in either case, z_i is stable. If part (a) holds, then, since x and y are semi-stable by assumption, it follows that x' and y' are both semi-stable. Moreover, $\mu(x)$ must be a function item (otherwise $\mu(x';y') = \perp$, and $x';y'$ is not well-typed). Thus, by the definition of stability, for function items, it follows that $x';y'$ is stable. Now by Lemma 11.11, it follows that z_i is stable. If part (b) holds, by Lemma 11.10, $\Omega(\rho(z_1))$ is stable. The stability of z_i now follows from Proposition 11.8. The stability of z_k follows from one more application of Proposition 11.8. It follows that if $x;y \rightarrow_b^* z$ and z is well-typed, then z is stable. We now show that there can be only finitely many FP-style rules in this rewrite sequence. We know that one of the three clauses of Lemma 11.12 must apply. If it is clause (1), there can be only finitely many FP-style rules since both x and y are e.s.n. by assumption. If it is either clause (2) or (3), then by the arguments in the preceding paragraph, the expression z_i satisfying clause (2) or (3) is stable. It follows that there can be only finitely many FP-style rules in the entire sequence. This shows that $x;y$ is e.s.n. $\square$

Lemma 11.15. *x is stable iff x is semi-stable.*

Proof. The only-if direction follows from Proposition 11.8. We prove the if direction by induction on $\tau(x)$. If $\tau(x) = 0$, the result is trivial since x is e.s.n. and that's all that needs to be checked. So assume that $\tau(x) > 0$. If $\mu'(x)$ is not a sequence item or a function item, we are done since x is assumed to be e.s.n. If $\mu'(x)$ is a sequence item, then every component z of x is an immediate component of some well-typed y such that $x \rightarrow_b^* y$ and hence x is stable. If $\mu'(x)$ is a function item, let z be stable with $\tau(z) < \tau(x)$. We are done if we show that $x;z$ is stable. By the induction hypothesis, it suffices to show that $x;z$ is semi-stable. This follows from Proposition 11.14. $\square$

3. *there exists an i such that $y_i = R_i(\Omega(t))$.*

Proof. Since the proof is similar to the proof of Lemma 11.12, we confine ourselves to a few comments. If every $\rho(y_i)$ has the form $al:t$ for some t , then with the exception of the P -rules that move a P or C past al_i , P rules or lowering rules can occur only entirely inside of $Q_i(al_i)$, x_i , or R_i . It is easy to see that in all the cases, $x_i \rightarrow_c^* x_{i+1}$. Also note that the rules moving a P or C past al_i can be applied only finitely many times since each time such rule is applied to rewrite $P_m(x') \rightarrow_a x''$ or $C_{m,n}(x') \rightarrow_a x''$, then either the type of x'' is lower than the type of x' , or the length of $\mu'(x'')$ is less than the length of $\mu'(x')$. Let i be chosen so the $z_{i-1} \rightarrow_b z_i$ is the last occurrence of one of those rules. (If there are no such occurrences just take $i = 0$.) The proof for part 2 is similar except that i is chosen to be the unique index with $\rho(z_i) = al:\rho(x_i)$ and $\rho(z_{i+1}) = \langle \rho(t_0), \dots, \rho(t_n) \rangle$. $\square$

Lemma 11.21. *If x is stable, then $R(x)$, $al_m:x$, and $ar_n:x$ are stable for all m, n .*

Proof. This is proven simultaneously by induction on x under the $<_{se}$ relation. Assume that x is stable and that the lemma holds for all stable $x' <_{se} x$. It is easy to see by induction on the length of the reduction sequence that if $x \rightarrow_c^* x''$, then either $x \rightarrow_b^* x''$ or $x \rightarrow_b^* \langle x_1, x_2 \rangle$ and $\langle R_1(x_1), R_2(x_2) \rangle \rightarrow_b^* x''$ for some choice of x_1, x_2, R_1, R_2 . The induction hypothesis thus implies that if $x \rightarrow_c^* x''$, then x'' is stable.

To prove that $al_m:x$ is stable, we show that it is e.s.n. and that every component is stable. Consider a reduction sequence $al_m:x = y_0 \rightarrow_b y_1 \rightarrow_b y_2 \rightarrow_b \dots$ for $al_m:x$. If part 1 of Lemma 11.20 applies, then $y_i = R_i(Q_i(al_i):x_i)$ where $x \rightarrow_c^* x_i$. Hence x_i is stable and e.s.n. Therefore, by part 1 of Lemma 11.20, the entire reduction sequence contains only finitely many FP-style reductions. If parts 2 or 3 of Lemma 11.20 apply, a similar argument can be used. A similar argument using the cases of Lemma 11.20 can also be used to show that every component of $al_m:x$ is stable. The proof for $ar_m:x$ is similar, so is omitted.

Finally, we must show that $R(x)$ is stable. It suffices to show that $R(x)$ is e.s.n. and that every component of $R(x)$ is stable as well. We leave it to the reader to check that Lemma 11.19 implies that it suffices to consider the cases where x has one of the forms $al_i: \langle z_1, z_2 \rangle$, $ar_i: \langle z_1, z_2 \rangle$, or $\langle z_1, \dots, z_n \rangle$. If $x = al_i: \langle z_1, z_2 \rangle$, then since z_1, z_2 are components of x and hence stable and the inductive hypothesis applies to $\langle z_1, z_2 \rangle$, an argument using Lemma 11.20 similar to the one above can be used to prove the result. We leave the other cases to the reader. $\square$

Corollary 11.22. *If x is stable, then $C_{m,n}(x)$ is stable for all m, n .*

Corollary 11.23. *al_l and ar_l are stable for all l .*

Finally, we want to prove that every primitive function symbol is stable. It is useful to divide the function symbols into two groups. We say that the FP' function symbol f is of *type 1* if $\rho(f) \in \{\text{first, last, tl, tr, null, cond, apply, id}\}$ and of *type 2* if $\rho(f) \in \{\text{comp, cons, K}\}$. Notice

then parts 2a and 2b are vacuously satisfied and part 1a follows by assumption. If clause 4 occurs, then parts 1a and 2b are vacuously satisfied and part 2a follows by assumption. This completes the proof of parts 1a, 2a, and 2b.

We now prove part 1b. Any component of $f:x_1:\dots:x_n$ must be an immediate component of some w for which $f:x_1:\dots:x_n \rightarrow_b^* w$. Consider the reduction sequence going from $f:x_1:\dots:x_n$ to w . Again we apply Lemma 11.24. Note that clause 1 cannot apply to the reduction sequence since ρ of the leftmost symbol in any step in clause 1 must be $\rho(f)$ whereas the leftmost symbol in $\rho(w)$ must be one of $al, ar, <$. Since $\mu'(w) \neq \perp$ clause 2 cannot apply to the reduction sequence. Clause 4 cannot apply since f is assumed to be of type 1. Hence clause 3 must apply. In this case, $z:x_2:\dots:x_n \rightarrow_b^* w$ for some $z \in RHS(f, x_1)$ and hence x is a component of $z:x_2:\dots:x_n$ as desired. This completes the proof of part 1b. The proof of 2c is similar and is left to the reader. $\square$

Lemma 11.26.

1. If f is of type 1, then
 - a. f is stable iff $f:x_1$ is stable for all stable x_1 .
 - b. If x_1 is stable, then $f:x_1$ is stable iff every member of $RHS(f, x_1)$ is stable.
2. If f is of type 2, then
 - a. f is stable iff $f:x_1:x_2$ is stable for all stable x_1, x_2 .
 - b. If x_1, x_2 are stable, then $f:x_1:x_2$ is stable iff every member of $RHS(f, x_1, x_2)$ is stable.

Proof. Since the proofs are similar, we prove part 2 and leave part 1 to the reader. The only-if direction of part 2a follows from Corollary 11.16. To prove the if direction of part 2a, it suffices to show that $f:x_1$ is stable for all stable x_1 . By part 2b of Lemma 11.25, it follows that $f:x_1$ is e.s.n. Since f is of type 2, it must be the case that $\mu'(f:x_1)$ is a function item or $\perp$. Thus, in order to prove that $f:x_1$ is stable, it now suffices to show that $f:x_1:x_2$ is stable for all stable x_2 . Since this is true by assumption, the proof of part 2a is complete.

The only-if direction of part 2b follows from Proposition 11.8. To prove the if direction of part 2b, we first show that if $n \geq 2$ and $x_1, \dots, x_n$ are all stable, then $f:x_1:\dots:x_n$ is e.s.n. By part 2a of Lemma 11.25, it suffices to show that $z:x_3:\dots:x_n$ is e.s.n. for all $z \in RHS(f, x_1, x_2)$. Note that z is stable by assumption and hence $z:x_3:\dots:x_n$ is stable by Corollary 11.16. Thus, $f:x_1:\dots:x_n$ is e.s.n. We next prove simultaneously for all $n \geq 2$ by induction on $\tau(f:x_1:\dots:x_n)$ that $f:x_1:\dots:x_n$ is stable provided $x_1, \dots, x_n$ are all stable. If $\mu'(f:x_1:\dots:x_n)$ is not a function item or sequence item, then we are done since we already know that $f:x_1:\dots:x_n$ is e.s.n. If $\mu'(f:x_1:\dots:x_n)$ is a function item, then we are done since $f:x_1:\dots:x_n:x_{n+1}$ is stable for stable x_{n+1} by the induction hypothesis. In the case that $\mu'(f:x_1:\dots:x_n)$ is a sequence item, by part 2c of Lemma 11.25, every component of $f:x_1:\dots:x_n$ is a component of $z:x_3:\dots:x_n$ for some $z \in RHS(f, x_1, x_2)$. By assumption and Corollary 11.16, $z:x_3:\dots:x_n$ is stable. Thus, every component of $z:x_3:\dots:x_n$ is stable, and $f:x_1:\dots:x_n$ is stable, as desired. $\square$

12. Appendix D: Modified Leftmost is O-safe

In this appendix we show that the modified leftmost strategy is O-safe. We want to show that if $\mu(x)$ is an observable item, then for some n , $ML(x, n) = \mu(x)^e$. By Corollary 4.4, it suffices to show that ML reduces x to normal form. Since $\mu(x)$ is observable, we can find a well-formed FP' expression x' such that $\rho(x'_0) = x$ and $\{\mu'(x'_0)\}^d = \mu(x)$. Consider a tracking sequence (see Definition 6.7) $x'_0, x'_1, x'_2, \dots$ for x under the ML rewrite strategy. As noted, it suffices to show that $\rho(x'_n)$ is in normal form for some n . For this, it suffices to show that $x'_k \rightarrow_a^* x'_{k+1}$ uses an FP-style rule (as opposed to using a bottom rule) whenever $\rho(x'_k)$ is not in normal form. For then we have that either $\rho(x'_n)$ is in normal form for some n or else the tracking sequence has an infinite number of FP-style rules, contradicting Theorem 5.10. Since ML is history independent, it suffices to show the result for the first step, namely that $x'_0 \rightarrow_a^* x'_1$ uses an FP-style rule whenever $\rho(x'_0)$ is not in normal form and $ML(\rho(x'_0), 1) = \rho(x'_1)$, provided $\mu'(x'_0)$ is observable. We will find it more convenient to prove the contrapositive, namely that if $x'_0 \rightarrow_a x'_1$ using a bottom rule, then $\mu'(x'_0)$ is not observable. Before checking this, we need a bit of notation. Let x^{ml} be the FP expression obtained from the FP expression x by replacing the ML redex (if one exists) by $a:b$ where a and b are atoms. (There is nothing special about the choice of $a:b$; any expression whose meaning is $\perp$ would do.)

Lemma 12.1. *If $x'_0 \rightarrow_a x'_1$ by the use of a bottom rule and $\rho(x'_1) = ML(\rho(x'_0), 1)$, then $\mu'(x'_0) \in \mu((\rho(x'_0))^{ml})$.*

Proof. This is easily proved by induction on the structure of x'_0 . For example, assume that x'_0 has the form $y':z'$. Then $y':z'$ is not the redex for the $\rightarrow_a$ rule used to rewrite x'_0 to x'_1 since we are assuming that a bottom rule was used. Therefore, the $\rightarrow_a$ redex must be entirely contained inside of either y' or z' . Say, for example, that the redex is entirely contained in y' . Then we get $\mu'(x'_0) = \mu'(y':z') = \mu'(y') : \mu'(z') \in \mu((\rho(y'))^{ml}) : \mu(\rho(z')) = \mu((\rho(y'))^{ml}; \rho(z')) = \mu((\rho(y'):\rho(z'))^{ml}) = \mu((\rho(y':z'))^{ml}) = \mu((\rho(x'_0))^{ml})$. $\square$

We next want to show that if x is an FP expression not in normal form, then $\mu(x^{ml})$ is not observable. Once we show this, it follows from Lemma 12.1 that no bottom rule could have been used in a tracking sequence for (ML, x) if $\mu(x)$ is observable. Coupled with the previous observation, this says that for some n , it must be the case that $ML(x, n)$ is in normal form. Thus the fact that ML is O-safe follows from part 3 of the following lemma (parts 1 and 2 are necessary only as induction hypotheses).

Lemma 12.2. *If x is not in normal form, then*

1. *If $\mu(x^{ml})$ is a sequence item, then x has one of the forms $al:\langle x_1, x_2 \rangle$, $ar:\langle x_1, x_2 \rangle$, or $\langle x_1, \dots, x_k \rangle$; if $\mu(x^{ml})$ is of definite length, then in fact x is of the form $\langle x_1, \dots, x_k \rangle$.*
2. *If $\mu(x^{ml})$ is a function item, then x has the form $f:y_1$ where f is comp, cons, or K.*
3. *$\mu(x^{ml})$ is not observable.*

induction hypothesis). Thus, by the the induction hypothesis, z must have one of the forms $al: \langle z_1, z_2 \rangle$ or $ar: \langle z_1, z_2 \rangle$. Since they are similar, we consider only the al case. z_2 cannot have the form $\langle \dots \rangle$ (otherwise $\mu(z^{ml})$ has definite length). z_2 cannot have one of the forms $al: \langle z_3, z_4 \rangle$ or $ar: \langle z_3, z_4 \rangle$ (otherwise x is a redex). If z_2 is in normal form, then by Theorem 4.3 $\mu(z_2)$ is not a sequence item and hence $\mu(x) = \perp$; thus, $\mu(x^{ml}) = \perp$ which is a contradiction. If z_2 is not in normal form, then by the induction hypothesis, $\mu(z_2^{nl})$ is not a sequence item and hence $\mu(x^{ml}) = \perp$ which is also a contradiction. Thus, $y \neq tl$.

Similar proofs work in all other cases. $\square$

13. Appendix E: Characterizing S-safety

Recall that we are trying to show that (1) R is S-safe for x iff (2) every tracking sequence for (R, x) reaches e.n.f. We first show that (1) implies (2).

Definition 13.1. A rewrite strategy R is *steady* with respect to a well-formed FP' expression x' iff R is S-safe for $\rho(x')$ and there exists a tracking sequence for $(R, \rho(x'))$ starting with x' in which no FP-style rewrite rules are used.

Lemma 13.2. If x' is a well-formed FP' expression and R is a rewrite strategy that is steady with respect to x' , then x' is in essential normal form.

Proof. Suppose x' is well-formed and R is steady w.r.t x' . We first show that if $\mu'(x') \neq \perp$, then $x = \rho(x')$ must be transparent. For suppose not. Then, since R is S-safe for x , there must be some $n > 0$ such that $R(x, n)$ is transparent and $R(x, n')$ is not transparent for $n' < n$. Let $x'_0, x'_1, \dots$ be any tracking sequence for (R, x) with $x' = x'_0$. We claim there must be an FP-style rewrite rule applied in reducing $x'_{n-1} \rightarrow_a^* x'_n$. Since $\rho(x'_{n-1}) \neq \rho(x'_n)$, either a bottom rule or an FP-style rule must be applied. If it is a bottom rule, then we leave it to the reader to check that we must have $\mu'(x_{n-1}) = \perp$. But this contradicts the assumption that $\mu'(x') \neq \perp$. Thus every tracking sequence for (R, x) starting with x' has an occurrence of at least one FP-style rule. But this contradicts the assumption that R is steady for x' .

The lemma now follows by induction on the structure of x' , using the observation above. If x' is an atom or a primitive function or of the form $\Omega(x)$, then it is clearly in e.n.f. If x' is of the form $\langle y'_1, \dots, y'_m \rangle$ and no y'_i has the form $*(...)$, let R'_i be the strategy defined on $\rho(y'_i)$, $i = 1, \dots, m$, by taking $R'_i(\rho(y'_i), n)$ to be the i^{th} component of $R(x, n)$. It is easy to see that R'_i is steady for y'_i , so by the inductive hypothesis, y'_i is in e.n.f. Hence $x' = \langle y'_1, \dots, y'_m \rangle$ is in e.n.f. as well. A similar proof works if one of the y'_i has the form $*(...)$. If x' has one of the forms $y':z'$, $P_m(y':z')$, $C_{p,q}(y':z')$, or $C_{p,q}(P_m(y':z'))$ (the only remaining cases, since x' is well formed), then we proceed by cases on the form of $\mu'(x')$. Since x' does not have the form $\Omega(x)$ and x' is well-formed, we have that $\mu'(x') \neq \perp$. Thus, by the observation above, we know that $\rho(x')$ must be transparent. This means that $\rho(x)$ must be of the form $al: \langle z, y \rangle$ or $ar: \langle y, z \rangle$, where $\mu(y)$ is not of definite length, or of the form $f:y$, where f is one of $comp$, $cons$, or K . We consider only the case that $\rho(x')$ is of the form $comp:y$ since all the

If $x' \rightarrow_a y'$ by an FP-style rule, then $Plen(x') \leq 2$ since x' is well-formed. There are two ways that $Plen$ can grow, one by the FP-style rule introducing a **P** or **C** and the other by the FP-style rule taking away some structure and thereby allowing two **P** or **C**'s to come together. The first occurs in many FP-style rules including the **K** rule $(K_m:x):y \rightarrow P_{m-2}(x)$. The second occurs only when the redex involves a **K** or **id**; a case analysis shows that in all cases we have $Plen(y') \leq 5$. $\square$

There are two related notions of having large subscripts that we consider. The first (and weaker) notion will be called partial support. This is essentially a syntactic notion; an expression x has partial support n if, roughly speaking, the subscripts on all the **P**'s and on all the primitive functions symbols that appear in x are at least n . (As we shall see, the situation is somewhat more complicated due to the presence of expressions of the form $C_{m,n}(x)$.) An expression x has total support n if, roughly speaking, in addition to having partial support n , the meaning of x is a good approximation to the meaning of $\rho(x)$, in that $\mu'(x)$ is at least as large as $P_n(\mu(\rho(x)))$. Actually, we want this property not just for x , but for all of its subexpressions. Formally, we define a function PS mapping FP' expressions to $\mathbb{N} \cup \{\infty\}$ by induction on structure:

1. $PS(a) = \infty$
2. $PS(f_n) = n$
3. $PS(\Omega(t)) = \sup\{n \mid P_n(\mu(x)) = \perp\}$
4. $PS(\langle x_1, \dots, x_m \rangle) = \min\{\infty, PS(x_1), \dots, PS(x_m)\}$
5. $PS(\langle x_1, \dots, x_i, *(t_1, \dots, t_j), y_1, \dots, y_k \rangle) = 0$
6. $PS(P_n(x)) = \min(n, PS(x))$
7. $PS(C_{m,n}(x)) = \begin{cases} \min(PS(x), \sup\{k \mid \mu'(C_{k,n}(x)) \leq \mu'(C_{m,n}(x))\}) & \text{if } \mu'(x) \text{ is a sequence} \\ \sup\{n \mid P_n(\mu(x)) = \perp\} & \text{otherwise} \end{cases}$
8. $PS(x_1:x_2) = \min(PS(x), PS(y))$.

We say that x has *partial support* n if $n \leq PS(x)$; we say that x has *total support* n if x has partial support n and $P_n(\mu(\rho(y))) \leq \mu'(y)$ for all subexpressions y of x . Note that if $n \leq n'$ and x has total (resp. partial) support n' , then it has total (resp. partial) support n as well.

We want to relate the partial support of expressions to their total support, and estimate how the partial and total support change as a result of applying rewrite rules. These goals motivate the next four lemmas.

Lemma 13.4. *For all FP expressions x and all integers m , there exists an integer n such that if $\rho(x') = x$ and x' has partial support n , then x' has total support m .*

Proof. This is proved by induction on the structure of x . The only interesting case is when x has the form $x = x_1:x_2$. Since $P_m(\mu(x_1:x_2)) \in \mu(x_1):\mu(x_2)$, we can find $x_1 \in \mu(x_1)$ and $x_2 \in \mu(x_2)$ such that $P_m(\mu(x_1:x_2)) = x_1:x_2$. Let $m' \geq m$ be large enough so that $x_1, x_2 \in D_{m'}$.

anywhere near n . This phenomenon is one of the principle reasons for the considering the notion of partial support.

Lemma 13.7. *For every finite FP reduction sequence $R(x, 0), \dots, R(x, k)$ and every $M \geq 0$, there exists N such that if $x'_0 \rightarrow_a^* \dots \rightarrow_a^* x'_k$ is a tracking sequence for the given FP reduction sequence and x'_0 has total support N , then x'_i has total support M for all $i = 0, \dots, k$.*

Proof. This is easily proved by induction on k (the length of the reduction sequence) once we have established the base case. So assume $k = 1$ and fix M . If $x'_0 = x'_1$ or $x'_0 \rightarrow_a x'_1$ by means of a bottom rule, just take $N = M$. The only remaining case is that there exists x''_1 such that $x'_0 \rightarrow_a x''_1$ by means of an FP-style rule and then $x''_1 \rightarrow_a^* x'_1$ by using only P -rules and lowering rules. Since, by Proposition 5.6, there can only be $2^{2^{|R(x,1)|}}$ steps involved in rewriting $x''_1 \rightarrow_a^* x'_1$, by Lemma 13.5, if x''_1 has total support $M_1 = M + |R(x, 1)| \times 2^{2^{|R(x,1)|}}$, then x'_1 has total support M . By Lemma 13.4, there exists N_1 such that if x''_1 has partial support N_1 , then it has total support M_1 . Take $N = N_1 + 2$. By Lemma 13.6, if x'_0 has total support N , then x''_1 has partial support N_1 . The result now follows. $\square$

Lemma 13.8. *There exists a function BIG such that for all positive integers n , rewrite strategies R , and FP expressions x , $BIG(n, R, x)$ is a positive integer with the following property: for every FP' expression x' such that (a) x' is well-formed, (b) $\rho(x') = x$, (c) x' has total support $BIG(n, R, x)$, and (d) some tracking sequence for (R, x) starting with x' reaches e.n.f., there exists m such that $R(x, m)$ is n -transparent.*

Proof. First we define $BIG(n, R, x) = 0$ if $\mu(x) = \perp$. The result is trivially true in this case. For the rest of the proof, we assume that $\mu(x) \neq \perp$. We define BIG by induction on n for all R and x . For the base case, choose k big enough so that $\mu(x) \cap D_k$ contains an element other than $\perp$ and so that if $\mu(x)$ is a sequence of definite length k' , then $k \geq k'$; define $BIG(1, R, x)$ to be any such k . Suppose the m^{th} term in some tracking sequence for (R, x) starting with x' is in e.n.f. If this m^{th} term is x'_m , then by Proposition 5.12, it follows that $\rho(x'_m) = R(x, m)$ is transparent.

Inductively, suppose we have defined $BIG(i, R, x)$ for all R and x and $i \leq n$. We show how to define $BIG(n+1, R, x)$. Given R and x , there are two cases to consider: either for all m , $R(x, m)$ is not transparent, or for some m , $R(x, m)$ is transparent. In the former case, we take $BIG(n+1, R, x) = BIG(1, R, x)$. Given a well-formed x' with support $BIG(1, R, x)$ and with $\rho(x') = x$, then no tracking sequence starting with x' ever reaches e.n.f (for otherwise, by the definition of $BIG(1, R, x)$, there would be some m such that $R(x, m)$ is transparent, contradicting our assumption), so that the result is vacuously true. In the latter case, let m be such that $R(x, m)$ is transparent and $R(x, m')$ is not transparent for $m' < m$. $R(x, m)$ is either an atom, a primitive function symbol, or of the form $\langle y_1, \dots, y_k \rangle$, $\text{al} : \langle y, z \rangle$, $\text{ar} : \langle y, z \rangle$, or $\text{f} : y$, for $\text{f} = \text{K}, \text{cons}$, or comp . If $R(x, m)$ is either an atom or a primitive function symbol,

It is useful to define a meaning function μ'_r on FP' expressions that is an analogue of μ_r . More formally, μ'_r is defined inductively just as μ' is. The only clause that is different is the clause that $\mu'_r(x_1:x_2) = \perp$ if $\rho(x_1:x_2)$ is an FP redex and otherwise $\mu'_r(x_1:x_2) = \mu'_r(x_1) \cdot \mu'_r(x_2)$. In order to show that (1) implies (2), we need a preliminary lemma.

Lemma 14.1. *For all FP expressions x and all integers m , there exists an integer n such that if $\rho(x') = x$ and x' has partial support n , then $P_m(\mu_r(x)) \leq_I \mu'_r(x')$.*

Proof. This is easily proved by induction on the structure of x in a manner similar to the proof of Lemma 13.4. $\square$

We can now show that (1) implies (2) in our characterization of L-safety. Suppose R is L-safe for x and $d \in \mu(x)$. Since R is L-safe for x , there exists k such that $d \in \mu_r(R(x, k))$. There exists an integer m such that $d \in D_m$. By Lemma 14.1, we can find an integer n such that if $\rho(x'_k) = R(x, k)$ and x'_k has total support (or even just partial support) n , then $d \leq_I P_m(\mu_r(R(x, k))) \leq_I \mu'_r(x'_k)$. By Lemma 13.7, we can find an integer N such that if x'_0 has total support N and $x'_0, \dots, x'_n$ is a $(\rightarrow_a)$ tracking sequence for $R(x, 0), \dots, R(x, k)$, then x'_k has total support n . By Lemma 13.9, we can find a well-formed expression x'_0 with total support N and with $\rho(x'_0) = x$. Let $x'_0, \dots, x'_k$ be a $(\rightarrow_a)$ tracking sequence for $R(x, 0), \dots, R(x, k)$. Since x'_k has total support n , it follows that $d \leq_I \mu'_r(x'_k)$. Let x''_k be formed from x'_k by replacing all subexpressions of x'_k of the form $y_1:y_2$ for which $\rho(y_1:y_2)$ is an FP redex by the new subexpression $\Omega(\rho(y_1:y_2))$. It is easy to see by induction on the structure of x'_k that $\mu'(x''_k) = \mu'_r(x'_k)$. Furthermore, note that $x'_k \rightarrow_b x''_k$ by means of the two step rewriting $y_1:y_2 \rightarrow_b P_0(y_1:y_2) \rightarrow_b \Omega(\rho(y_1:y_2))$. Let $x''_k \rightarrow_a^* x'''_k$ where x'''_k is well-formed and only lowering and P -rules are used to reduce x''_k to x'''_k . It is easy to see by construction that x'''_k is in e.n.f. This gives us that $d \leq_I \mu'_r(x'_k) = \mu'(x''_k) = \mu'(x'''_k)$ as desired. Finally note that the reduction sequence $x'_0, \dots, x'_{k-1}, x'''_k$ can be extended past k to a tracking sequence for the entire reduction sequence $R(x, 0), R(x, 1), \dots$ by applying Lemma 6.8 to the strategy $R'(R(x, k), j) = R(x, k + j)$.

In order to show that (2) implies (1), assume (2) holds and we are given $d \in \mu(x)$. We want to show that for some n , we have $d \in \mu_r(R(x, n))$. By assumption, we can find a $\rightarrow_b$ tracking sequence $x'_0, x'_1, \dots$ for (R, x) such that for some n , x'_n is in e.n.f. and $d \leq_I \mu'(x'_n)$. By part 3 of Lemma 7.1, we have that $d \leq_I \mu'(x'_n) \in \mu_r(R(x, n))$ which proves the result. $\square$

This characterization of L-safety was in terms in $\rightarrow_b$ tracking sequences. Does a similar characterization hold if we use $\rightarrow_a$ tracking sequences instead? As the following example shows, the answer is no.

Example 14.2. Let $x = \text{cons} : \langle \text{id} : \text{id}, \text{id} : \text{id} \rangle$, $y = \text{cons} : \langle \text{id}, \text{id} : \text{id} \rangle$, $z = \text{cons} : \langle \text{id} : \text{id}, \text{id} \rangle$, $w_0 = \text{cons} : \langle \text{comp} : \langle \text{comp}, \text{cons} : \langle K : \text{first}, \text{id} \rangle \rangle, \text{comp} : \langle \text{comp}, \text{cons} : \langle K : \text{last}, \text{id} \rangle \rangle \rangle : x$, $w_1 = \langle \text{comp} : \langle \text{first}, x \rangle, \text{comp} : \langle \text{last}, x \rangle \rangle$, and $w_2 = \langle \text{comp} : \langle \text{first}, y \rangle, \text{comp} : \langle \text{last}, z \rangle \rangle$. It is easy to check that $w_0 \rightarrow^* w_1 \rightarrow^* w_2$. Thus

$$3. \mu_{pr}(x) = \min \{ \mu_r(y) \mid x \rightarrow^* y \}.$$

Proof. Part 1 is immediate from the definitions. Part 2 follows easily by induction on the structure of x . Part 3 follows from parts 1 and 2. $\square$

Using μ_{pr} , we can now define our last notion of having “enough” rules:

Definition 15.2. A set of rewrite rules is *LL-complete* iff for all FP expressions x and all $d \in \mu(x)$ there is an FP expression y such that $x \rightarrow^* y$ and $d \in \mu_{pr}(y)$. R is *LL-safe for x* iff for all $d \in \mu(x)$, there exists n such that $d \in \mu_{pr}(R(x, n))$. R is *LL-safe* iff it is LL-safe for all FP expressions x .

By part 2 of Proposition 15.1 and Theorem 7.4 it follows that FP has the LL-completeness property. By part 1 of Proposition 15.1, it follows that if R is LL-safe for x then it is also L-safe for x , and so LL-safety implies L-safety. To see that the converse does not hold, let $x = \text{comp}: \langle \text{first}, \text{cons}(\text{al}: \langle K:a, \text{id}: \langle \rangle \rangle) \rangle$. It is easy to check that $\mu_r(x) = \mu(x) = \mu(K:a)$, while $\mu_{pr}(x):t = \perp$ for all t . So if R is a strategy that does nothing on x (i.e., $R(x, n) = x$ for all n), then R is L-safe for x but not LL-safe for x .

We would also like to show that S-safety implies LL-safety. To do this, we first give a characterization of LL-safety in terms of FP' tracking sequences. To understand the intuition behind this definition, recall that in Lemma 7.1 we showed that if x' is in e.n.f., then $\mu'(x') \in \mu_r(\rho(x'))$. It is not necessarily the case that if x' is in e.n.f. then $\mu'(x') \in \mu_{pr}(\rho(x'))$. For a counterexample, consider $x' = \text{al}_1: \langle a_1, \Omega(\text{id}: \langle a_1 \rangle) \rangle$. It is easy to see that x' is in e.n.f., $\mu'(x') = \langle a_1, * \rangle$, and $\mu_{pr}(\rho(x')) = \perp$.

This expression x' is exactly the same one we used to show that it is possible for an FP' expression to be in e.n.f. and yet be rewritten to another expression which is *not* in e.n.f. We define an FP' expression to be in *almost normal form (a.n.f.)* iff every expression it rewrites to (using $\rightarrow_a$ rewriting) is in e.n.f. One can show that any well-formed FP' expression x' in e.n.f. can be rewritten to a.n.f. using at most a finite number of FP-style rules (indeed, the number is at most the number of occurrences of al and ar). We can now show that if x' is an FP' expression in a.n.f., then $\mu'(x') \in \mu_{pr}(\rho(x'))$. Using a.n.f., we can now state our characterization for LL-safety.

Theorem 15.3. *Let R be a rewrite strategy. Then the following are equivalent:*

1. R is LL-safe for x .
2. For all $d \in \mu(x)$, there exists a tracking sequence $x'_0, x'_1, \dots$ for (R, x) such that for some n , x'_n is in a.n.f. and $d \leq_I \mu'(x'_n)$.

Proof. Just like that for Theorem 7.7, replacing e.n.f. by a.n.f. throughout. We omit details here. $\square$

References

- [Ba78] J.W. Backus, Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, *Communications of the ACM* **21:8** (August, 1978), pp. 613-641.
- [Bo82] C. Böhm, Combinatory foundation of functional programming, *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming*, (August, 1982), pp. 29-36.
- [BMS80] R. Burstall, D. MacQueen, and D. Sannella, HOPE: An experimental applicative language, *Proceedings of the 1980 LISP Conference* (August, 1980), pp. 136-143.
- [DM84] W. Dosch and B. Moller, Busy and lazy FP with infinite objects, *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming* (August, 1984), pp. 282-292.
- [FLO83] S. Fortune, D. Leivant, and M.J. O'Donnell, The expressiveness of simple and second-order type structures, *Journal of the ACM* **30:1** (January, 1983), pp. 151-185.
- [FW76] D. Friedman and D. Wise, Cons should not evaluate its arguments, *Proceedings of the Third International Colloquium on Automata, Languages, and Programming*, Springer-Verlag (1976), pp. 257-284.
- [GMP82] J. Goguen, J. Meseguer, and D. Plaisted, Programming with parameterized abstract objects in OBJ, in *Theory and Practice of Software Technology*, North-Holland (1982).
- [HW³85] J.Y. Halpern, J.H. Williams, E.L. Wimmers, and T.C. Winkler, Denotational semantics and rewrite rules for FP, *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages* (January, 1985), pp. 108-120.
- [HM76] P. Henderson and J.H. Morris, A lazy evaluator, *Proceedings of the Third ACM Symposium on Principles of Programming Languages* (January, 1976), pp. 95-103.
- [HO84] C.M. Hoffmann and M.J. O'Donnell, Implementation of an interpreter for abstract equations, *Proceedings of the Eleventh ACM Symposium on Principles of Programming Languages* (January, 1984), pp. 111-120.
- [MNV73] Z. Manna, S. Ness, and J. Vuillemin, Inductive methods for proving properties of programs, *Communications of the ACM* **16:8** (1973), pp. 491-502.
- [Me82] A.R. Meyer, What is a model of the λ -calculus?, *Information and Control* **52**, (1982) pp. 87-122.
- [Ni85] F. Nickl, A denotational semantics for Backus' FP with infinite objects, Technical Report of the Fakultät für Mathematik und Informatik, Universität Passau (1985).
- [O'D77] M.J. O'Donnell, *Computing in Systems Described by Equations*, Lecture Notes in Computer Science No. 58, Springer-Verlag (1977).
- [PI77] G.D. Plotkin, LCF considered as a programming language, *Theoretical Computer Science* **5** (1977), pp. 223-255.