Full Abstraction and Expressive Completeness for FP*

Joseph Y. Halpern
Edward L. Wimmers

IBM Almaden Research Center
San Jose, CA 95120
email: halpern@ibm.com, wimmers@ibm.com

Abstract: We consider issues related to the expressive power of the programming language FP. In particular, we consider whether a number of variants of FP are fully abstract and expressively complete. For example, we show that a version of FP with only one-sided sequences behave similarly to PCF in that the addition of parallel *or* is sufficient to make it fully abstract.

However, the addition of parallel *or* to FP (with its two-sided infinite sequences) is *not* sufficient to achieve full abstraction. By considering these and other variants, we obtain a better understanding of what is required of a language and semantics in order to guarantee full abstraction and expressive completeness.

*This is an expanded version of a paper that appears in the Proceedings of the Second IEEE Symposium on Logic in Computer Science, 1987. It is essentially identical to the version that appears in *Information and Computation* **118**:2, 1995, pp. 246-271.

1 Introduction

In a previous paper [HWW90] we considered the question: "What does it mean to have enough rewrite rules?" in the context of the functional programming language FP84 (henceforth called FP), which extends the FP originally defined by Backus [Ba78]. We suggested a number of criteria by which to judge this question, and showed that with respect to all of these criteria, there indeed were enough rewrite rules in FP. The next obvious question, and the one we address in this paper, is "Do we have enough primitive functions?"

In one sense the answer is "yes". We do know that FP has enough primitive functions to allow us to *encode* all recursive functions on the integers (cf. [Wi81]), but we want more, since we want to manipulate the objects in our domain, not just the integers. We consider here two other well-known criteria from the literature that deal with the question of having enough primitive functions. The first of these is called *full abstraction*. For a language defined by rewrite rules, we say that two terms are *behaviorally equivalent* if, roughly speaking, they rewrite to the same normal form in all contexts. A denotational semantics is called *fully abstract* with respect to a language if behavioral equivalence of terms in the language coincides with denotational equivalence. Full abstraction is of interest because it allows one to prove the operational equivalence (or inequivalence) of two expressions simply by proving the equality (or lack of it) of their meanings. Thus purely semantic reasoning allows us to deduce operational facts. The second (generally more powerful) criterion of language expressiveness is *expressive completeness*. A language is *expressively complete* if every "computable" element in the semantic domain is definable in the language.

In a seminal paper [Pl77], Plotkin examines these criteria in the context of the programming language PCF (which is based on LCF, Scott's *logic of computable functions* [Mi73]). Plotkin shows that what is perhaps the most natural denotational semantics for PCF is not fully abstract. However, he shows that if we add a parallel conditional to PCF, the semantic domain is fully abstract with respect to the resulting language. Moreover, he shows that by further augmenting PCF with a recursive "existential" operator, the resulting language is expressively complete with respect to the domain.

When we consider full abstraction and expressive completeness for FP, a number of issues arise that are not present in PCF. We hope that by understanding these issues better, we can eventually construct a general theory that relates the operational behavior of a programming language and its denotational semantics.

To understand the subtleties that arise in FP, we must review a little of its syntax and semantics. In FP we can define sequences of elements and manipulate them. We allow operations at both ends of a sequence. For example, we can append an element to the left-hand end of a sequence or to the right-hand end and remove the first element or last element of a sequence. FP is also powerful enough to allow us to define infinite sequences of elements. The fact that we can manipulate sequences from either end allows us to define sequences that are infinite to the right (such as $\langle\langle a, a, a, \ldots\rangle\rangle$) and infinite to the left (such as $\langle\langle ..., a, a, a\rangle\rangle$).

To deal with this semantically, we allow a $*$ to appear in sequences. Intuitively, an expression like $\langle\langle a, *\rangle\rangle$ is an *approximation* to any sequence whose first element is a (where we say that x approximates y if $x \leq_I y$ for some appropriate partial order $\leq_I$ on the elements of the domain). Thus $\langle\langle a, *\rangle\rangle$ approximates all of the following: $\langle\langle a\rangle\rangle$, $\langle\langle a, \perp\rangle\rangle$, $\langle\langle a, a, a\rangle\rangle$, $\langle\langle a, *, a\rangle\rangle$ (where, as usual, $\perp$ is a bottom element which approximates all other elements). Similarly, $\langle\langle *, a\rangle\rangle$ is an approximation to any sequence whose last element is a. It is easy to see that both $\langle\langle a\rangle\rangle$ and $\langle\langle a, *, a\rangle\rangle$ are upper bounds for the set $\{\langle\langle a, *\rangle\rangle, \langle\langle *, a\rangle\rangle\}$, and in fact they are minimal upper bounds for this set. However, this set has no *sup* (least upper bound), since $\langle\langle a\rangle\rangle$ and $\langle\langle a, *, a\rangle\rangle$ are incomparable. Thus, the semantic domain for FP is not *consistently complete*: it is not the case that two elements with an upper bound necessarily have a least upper bound. It can be shown that we lose consistent completeness exactly because the * may appear anywhere in a sequence.

This lack of consistent completeness has surprising implications for the expressive power of FP. For example, consider the apply-to-all function *apall*, which, when given a function f and a sequence $\langle\langle a_1, ..., a_n\rangle\rangle$, applies the function f to all the elements in the sequence, producing $\langle\langle f : a_1, ..., f : a_n\rangle\rangle$. (This is much like the LISP **mapcar**.) We can show that *apall* is not definable in FP. Intuitively, the problem is due to the fact that * may appear in the middle of a sequence in FP, so we do not know *a priori* when to apply f to the left-hand elements of the sequence, and when to apply it to the right-hand elements of the sequence.

On the other hand, consider FP1, the variant of FP which allows operations only on the left-hand elements of sequences. While the difference between FP and FP1 seems innocuous, it does have some surprising consequences. For one thing, since we can append only to the left-hand end of sequences, the * can occur only at the right-hand end of a sequence. As a consequence, we can show that the semantic domain for FP1 *is* consistently complete. Moreover, we can show that apply-to-all is definable in FP1; intuitively, this is because we just have to work from the left-hand end of the sequence.

These differences between FP1 and FP persist when we consider full abstraction and expressive completeness. For FP1 we can prove results analogous to those of Plotkin, as strengthened by Curien [Cu86] and Abramsky [Ab90]. We can show that FP1 is not fully abstract, but by adding a parallel *or* we get full abstraction, and by further adding an existential operator we also get expressive completeness. Our proofs are based on those of Plotkin, although the proof for FP1 presents new complexities since FP1 has greater structure than PCF, and, unlike PCF, is an untyped language. However, the straightforward analogue of these results for FP provably fails. In particular, adding the parallel *or* operator to FP does not achieve full abstraction.

These results lead us to look for extensions to FP with four properties that we can express informally as: natural syntax, natural rewrite rules, enough rewrite rules, and reasonable expressive power. While it is hard to make precise exactly what natural syntax and semantics is, a minimal criterion is that the set of expressions be recursive and that the set of rewrite rules be recursively enumerable. We can judge whether the language has enough rewrite rules by seeing if we have *observable completeness* or *strong completeness* in the sense of [HWW90] (these notions are reviewed in Section 2). Finally, reasonable expressive power for us will be either full abstraction or expressive completeness.

It seems that consistent completeness in the semantic domain together with a certain amount of parallelism in the language (such as a parallel conditional) is sufficient for full abstraction and expressive completeness. While we do not yet have a general theory in which to make this statement precise, we can offer evidence for the claim. Consider NFP, the nondeterministic variant of FP that results by adding **union** to FP, where **union**:$\langle \mathbf{x}, \mathbf{y} \rangle$ rewrites (nondeterministically) to either $\mathbf{x}$ or $\mathbf{y}$. We can easily modify our semantic domain to handle nondeterminism, and the resulting domain is consistently complete. Since elements in the domain are actually sets and the ordering relation is just the subset relation, the sup of any two elements is always their union. In NFP, the nondeterminism gives us almost enough parallelism to prove full abstraction and expressive completeness. However, there is a precise sense in which we still cannot distinguish between the elements $\langle\!\langle \bot, * \rangle\!\rangle$ and $\langle\!\langle *, \bot \rangle\!\rangle$. Once we add operators into the language that let us distinguish between these two elements, we can prove full abstraction and expressive completeness.

Proving full abstraction for NFP turns out to be much easier than proving full abstraction for FP1. The reason is that taking sups is so much easier in NFP. The sup of two elements is simply their union. Indeed, being able to take sups seems to be a more fundamental condition than consistent completeness when trying to prove full abstraction and expressive completeness. The insights gained in studying NFP enable us to construct extensions of FP that are fully abstract and expressively complete, and yet have the same domain as FP (and so, in particular, have a semantic domain which is not consistently complete). These extensions, however, make heavy use of a particular way of encoding finite elements in the semantic domain as *observable* elements (those obtained by starting with atoms and closing off under the sequence operator).

The rest of the paper is organized as follows. In the next section we review the syntax and semantics of FP and some results regarding completeness of rewrite rules from [HWW90]. In Section 3 we prove that apply-to-all is not definable in FP, although it is definable in FP1. In Section 4 we define and discuss the notions of full abstraction and expressive completeness. In Section 5 we show that FP and FP1 are not fully abstract. In Section 6 we show how to extend FP1 to get a language that is fully abstract and expressively complete. In Section 7 we show that by adding a nondeterministic **union** construct to FP as well as constructs that distinguish $\langle\!\langle \bot, * \rangle\!\rangle$ and $\langle\!\langle *, \bot \rangle\!\rangle$, the resulting language is fully abstract and expressively complete. Adding **union** to FP forces us to extend the semantic domain to accommodate nondeterminism. In Section 8 we examine fully abstract and expressively complete extensions of FP that leave the semantic domain unchanged. We conclude in Section 9 with some discussion of the general issues involved in obtaining fully abstract and expressively complete languages and domains.

2 A review of FP syntax and semantics

FP has a very simple syntax. We assume the existence of a finite set $\mathcal{A}$ of *atoms*,[1] which includes **T** and **F** (intuitively, *true* and *false*). We also assume the existence of a set $\mathcal{F}$ of primitive function symbols which is disjoint from $\mathcal{A}$. For now we take the set $\mathcal{F}$ to be {**al**, **first**, **tl**, **ar**, **last**, **tr**, **null**, **cond**, **comp**, **cons**, **K**, **id**, **apply**, **eqatom**, **isatom**, **isseq**, **isfunc**}. Intuitively, **al** (which stands for *append left*), appends an element to the left end of a sequence; **tl** takes the tail of a sequence by removing the first element, and **first** returns the first element of a sequence; **ar**, **tr**, and **last** are the corresponding operators that work on the right-hand end of a sequence; **null** tests whether a sequence is empty; **cond** is a conditional; **comp** is the composition function; **cons** takes a sequence of functions $\langle \mathbf{f}_1, ..., \mathbf{f}_n \rangle$ as an argument and returns the function that, when applied to $\mathbf{x}$, returns the sequence $\langle \mathbf{f}_1{:}\mathbf{x}, ..., \mathbf{f}_n{:}\mathbf{x} \rangle$; **K** is the familiar combinator from combinatory calculus that converts $\mathbf{x}$ to the constant function that always returns $\mathbf{x}$; **id** is the identity function; **apply**, when given a pair of elements, applies the first to the second (so that we have the rewrite rule $\mathbf{apply}{:}\langle \mathbf{f}, \mathbf{x} \rangle \rightarrow \mathbf{f}{:}\mathbf{x}$); **eqatom** tests for equality between atoms (and is undefined if either of its arguments is not an atom); finally, **isatom** (resp., **isseq**, **isfunc**) tests whether its argument is an atom (resp., sequence, function). We remark that the variant of FP used in [HWW90] did not have **eqatom**, **isatom**, **isseq**, and **isfunc**, but there is no difficulty including them in the language and defining appropriate rewrite rules for them.

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

1. The formation of sequences: if $\mathbf{x}_1, ..., \mathbf{x}_n$ are FP expressions (for $n \geq 0$), then $\langle \mathbf{x}_1, ..., \mathbf{x}_n \rangle$ is also an FP expression.
2. Application of expressions: if $\mathbf{x}_1$ and $\mathbf{x}_2$ are FP expressions, then $(\mathbf{x}_1{:}\mathbf{x}_2)$ is also an FP expression.

∎

We get variants of FP by adding or removing elements from the set $\mathcal{F}$ of primitive function symbols. In particular, FP1 is obtained by removing **ar**, **tr** and **last**.

We provide a denotational semantics for FP as usual by defining a domain D^* and a mapping μ from FP expressions to D^*. The desired domain can be constructed using general techniques that may be found in [Gu87]. We use instead the construction of

[1] We take the set of atoms to be finite here for notational convenience. It actually suffices if there exists an expression whose meaning is a (possibly infinite) sequence s such that the set of members of s is exactly the set of atoms. Such an expression exists in the case that the atoms are finitely generated. For example, we could include the positive integers in our set of atoms as long as we had a successor function **S**, with the rewrite rule $\mathbf{S}{:}\mathbf{n} \rightarrow \mathbf{n+1}$.

[HWW90], which is more specifically geared to FP, and thus easier to explain. Since all the details of the construction are contained in [HWW90], we only provide a sketch here.

We start with a number of standard domain-theoretic definitions.

Definition 2.2: Let D be a set partially ordered by $\leq$. A subset X of D is *compatible* iff there is an element x in D which is an upper bound of X. X is *downward closed* iff $x \in X$ and $x' \leq x$ implies $x' \in X$. X is *directed* iff for all $x_1, x_2 \in X$, there exists $x_3 \in X$ such that $x_1 \leq x_3$ and $x_2 \leq x_3$. $(D, \leq)$ is *complete* iff every directed subset of D has a least upper bound in D. The least upper bound of a directed set X is denoted by $\bigsqcup X$. x is a *finite element of* D iff whenever $x \leq \bigsqcup Y$ 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 $\{y \leq x \,|\, y \text{ is a finite element of } D\}$ is directed and $x = \bigsqcup\{y \leq x \,|\, y \text{ is a finite element of } D\}$. ∎

We construct D^*, the domain for FP, in stages. We first construct a sequence of domains $D_0, D_1, ...,$ where D_0 consists of $\perp$ and, for each atom $\mathbf{a} \in \mathcal{A}$, a corresponding *atomic item* a, and D_{n+1} consists of D_n together with all the continuous functions from D_n to itself (these are called *function items*), and all sequences of the form $\langle\!\langle x_1, ..., x_k \rangle\!\rangle$ and $\langle\!\langle y_1, \ldots, y_\ell, *, z_1, \ldots, z_m \rangle\!\rangle$, where $\{x_1, \ldots, x_k, y_1, \ldots, y_\ell, z_1, \ldots, z_m\} \subseteq D_n$, $k \leq 2n$, and $\max(\ell, m) \leq n$. The latter type of element is called a *sequence item*; if it contains no $*$, it is further called a *sequence of definite length*. Our restriction on the length of sequences guarantees that each set D_n is finite. There is a straightforward way to extend a function item $f \in D_n$ so that it is defined on domains D_m for $m > n$; see [HWW90] for details. If $f \in D_n$ and $x \in D_m$, we write $f : x$ to denote the result of applying f to x. Of course, if $x \in D_{n-1}$, then $f : x = f(x)$.

Let D be the union of the D_n's. The ordering $\leq_I$ on D is defined as follows, where, as we mentioned above, $*$ approximates any finite list of elements.

- $\perp \leq_I x$ holds for all x
- for any atomic item a, $a \leq_I x$ holds iff $a = x$
- for any function item f, $f \leq_I g$ holds iff g is a function item and $f : x \leq_I g : x$ for all $x \in D$
- for any sequence item $\langle\!\langle x_1, ..., x_n \rangle\!\rangle$ of definite length, $\langle\!\langle x_1, ..., x_n \rangle\!\rangle \leq_I y$ iff $y = \langle\!\langle y_1, ..., y_n \rangle\!\rangle$ and $x_i \leq_I y_i$ for all $i = 1, ..., n$
- for any sequence item $\langle\!\langle x_1, ..., x_m, *, x'_1, ..., x'_n \rangle\!\rangle$, $\langle\!\langle x_1, ..., x_m, *, x'_1, ..., x'_n \rangle\!\rangle \leq_I y$ iff $y = \langle\!\langle y_1, ..., y_m, z_1, ..., z_k, y'_1, ..., y'_n \rangle\!\rangle$ where at most one of the z_i's is $*$ and $x_i \leq_i y_i$ for all $i = 1, ..., m$ and $x'_i \leq_i y'_i$ for all $i = 1, ..., m$

It turns out that D is not quite appropriate for our semantic domain, principally because it is not complete. For example, $\{\langle\!\langle a, * \rangle\!\rangle, \langle\!\langle a, a, * \rangle\!\rangle, \langle\!\langle a, a, a, * \rangle\!\rangle, \ldots\}$ is a directed set in D which has no least upper bound in D. We take D^* to be the *completion* of D, that is, the domain that results by adding limits of sequences to D (just as a real number can be viewed as the set of rationals less than it). The elements of D^* are the non-empty, downward-closed, directed subsets of D. The elements of D^* are ordered by the subset relation, which makes D^* into a complete partial order (thus every directed subset of D^* has a unique least upper bound). We call an element of D^* an atomic item if it is of the form $\{\perp, a\}$, where a is an atomic item of D; we call an element of D^* a sequence item if it is a set all of whose elements (other than $\perp$) are sequence items in D; finally, we call an element of D^* a function item if it is a set all of whose elements (other than $\perp$) are function items in D.

We can embed D into D^* by identifying an element d of D with the subset $\{d' \in D | d' \leq_I d\}$. Furthermore, it can be shown (see Proposition 2.3 below) that an element of D^* is finite iff it is in the image of this embedding. It is now straightforward to give each FP expression a meaning in D^*. The meaning of an atom $\mathbf{a}$ is the atomic item a; the meaning of a sequence $\langle \mathbf{x}_1, \ldots, \mathbf{x}_n \rangle$ is the sequence item $\langle\!\langle x_1, \ldots, x_n \rangle\!\rangle$, where $\mu(\mathbf{x}_i) = x_i$. We map the primitive function symbols into functions in D^* that have the obvious meanings, and application corresponds to function application in D^*. Again, further details can be found in [HWW90]. The construction of the semantic domain for FP1, which we call $D1^*$, is identical except that we restrict sequence items so that the $*$ can only occur at the right-hand end of a sequence. The semantic function mapping FP1 expressions to $D1^*$ is denoted μ_1.

The domains D^* and $D1^*$ have a number of important properties. We describe the properties of D^* below; those for $D1^*$ are analogous.

Proposition 2.3: [HWW90]

(a) D^* is a cpo (complete partial order).

(b) x is a finite element of D^* iff $x \in D$.

(c) D^* is algebraic.

(d) D^* is isomorphic to $\mathcal{A} + Seqs(D^*) + (D^* \to D^*)$, where $Seqs(D^*)$ consists of all sequences (possibly infinite to the right and/or left) of elements in D^* (and is described in more detail below).

Since it is the presence of two-sided infinite sequences that is the cause of the lack of consistent completeness in D^*, we describe the $Seqs$ operation in a little more detail below. Given a domain X with an ordering $\leq_X$, we want $Seqs(X)$ to consist of all sequences of elements in X, including sequences that may be infinite to the right and/or left. Given a natural number n, let $[n]$ be an abbreviation for the set $\{0, \ldots, n-1\}$. For convenience, we take $[0]$ to denote the empty set, and $[\omega]$ to denote the infinite set of all the natural numbers. Formally, $s \in Seqs(X)$ iff one of the following holds:

- $s = f$ where f is a function from $[n]$ into X for some $n \in N$. (This case corresponds to a finite sequence of definite length; if $n = 0$, we identify the function with empty domain with the empty sequence.)

- $s = (f, g)$ where where f is a function from $[i]$ into X and g is a function from $[j]$ in X, for some $i, j \in N \cup \{\omega\}$. (This case corresponds to the sequences of indefinite length. The first function gives all the elements to the left of the $*$ and the second function gives all the elements to the right of the $*$ in reverse order. If $i = 0$, then there are no elements to the left of the $*$, while if $j = 0$, there are no elements to the right of the $*$.)

The ordering $\leq_{seq}$ on $Seqs(X)$ is defined as follows:

- $f_1 \leq_{seq} (f_2, g_2)$ never holds

- $(f_1, g_1) \leq_{seq} f_2$ holds iff the following all hold:
  - $dom(f_1) = [i]$, $dom(g_1) = [j]$, $dom(f_2) = [k]$ where $i + j \leq k$ and $i, j, k \in N$
  - $f_1(\ell) \leq_X f_2(\ell)$ for all $\ell < i$
  - $g_1(\ell) \leq_X f_2(k - \ell - 1)$ for all $\ell < j$

- $f_1 \leq_{seq} f_2$ holds iff $dom(f_1) = dom(f_2)$ and $f_1(\ell) \leq_X f_2(\ell)$ for all $\ell \in dom(f_1)$

- $(f_1, g_1) \leq_{seq} (f_2, g_2)$ holds iff $dom(f_1) \subseteq dom(f_2)$, $f_1(\ell) \leq_X f_2(\ell)$ for all $\ell \in dom(f_1)$, $dom(g_1) \subseteq dom(g_2)$, and $g_1(\ell) \leq_X g_2(\ell)$ for all $\ell \in dom(g_1)$

Because of the ordering on sequence items, D^* is not consistently complete. For example, the set $\{\langle\bot, *\rangle, \langle *, \bot\rangle\}$ has no least upper bound, but it does have two minimal upper bounds: $\langle\bot\rangle$ and $\langle\bot, *, \bot\rangle$. In fact, if a finite subset X of D^* consisting of finite elements has any upper bounds, then there is a finite set X' of minimal upper bounds of X such that every upper bound of X is greater than one of the minimal upper bounds in X'. In fact, the domain D^* satisfies a more general property called *SPF* (see [Gu87] or [Ab91]).

Computation in FP proceeds by using rewrite rules to transform expressions. The rewrite rules are given in Appendix A. In the rules for **eqatom**, **isatom**, **isseq**, and **isfunc**, we say that $\mathbf{x}$ *looks like a sequence* if $\mathbf{x}$ is of the form $\langle \mathbf{x}_1, ..., \mathbf{x}_n \rangle$, **al**:$\langle \mathbf{y}, \mathbf{z} \rangle$, or **ar**:$\langle \mathbf{y}, \mathbf{z} \rangle$ and $\mathbf{x}$ *looks like a function* if $\mathbf{x}$ is either a primitive function symbol or of the form **K**:$\mathbf{y}$, **comp**:$\langle \mathbf{f}_1, \mathbf{f}_2 \rangle$, or **cons**:$\mathbf{z}$, where $\mathbf{z}$ looks like a sequence. Note that we use $\mathbf{x} \to \mathbf{y}$ to mean that "$\mathbf{x}$ can be rewritten to $\mathbf{y}$ in one step"; we take $\to^*$ to be the reflexive, transitive closure of the $\to$ relation. The set of rewrite rules for FP1 consists of all the rewrite rules for FP that do not mention **tr**, **ar**, or **last**.

We want the rewrite rules to preserve the meanings of expressions. Fortunately, we have carefully defined our semantics so that the rewrite rules are sound. We have:

Theorem 2.4: [HWW90] *If* $\mathbf{x} \to^* \mathbf{y}$*, then* $\mu(\mathbf{x}) = \mu(\mathbf{y})$.

Besides soundness, we also want to know that we have in some sense *enough* rewrite rules. This issue is investigated in detail in [HWW90], so we just briefly review the results here that we need.

We call an FP expression *observable* if it is either an atom or a sequence $\langle \mathbf{x}_1, ..., \mathbf{x}_n \rangle$, where the $\mathbf{x}_i$ are observable. To each observable expression there corresponds an *observable item*, which is obtained by replacing $\langle, \rangle$, and boldface wherever they occur by $\langle\langle, \rangle\rangle$, and italics. Given an observable item x, let x^e be the observable expression to which it corresponds.

Theorem 2.5: [HWW90] *If* $\mathbf{x}$ *is an expression such that* $\mu(\mathbf{x})$ *is an observable item, then* $\mathbf{x} \to^* \mu(\mathbf{x})^e$.

For example, if we have an expression $\mathbf{x}$ such that $\mu(\mathbf{x}) = \langle\langle a, \langle\langle b \rangle\rangle \rangle\rangle$, then it follows from Theorem 2.5 that $\mathbf{x} \to^* \langle \mathbf{a}, \langle \mathbf{b} \rangle \rangle$. This result is extended in [HWW90]. Intuitively, we want to show that if we have an expression whose meaning is an atom, then we can rewrite it to that atom, if its meaning is a sequence, then we can rewrite it to an expression that looks like a sequence, and if its meaning is a function, then we can rewrite it to an expression that looks like a function. These intuitions motivate the following definition.

Definition 2.6: An FP expression $\mathbf{x}$ is *transparent* iff

1. if $\mu(\mathbf{x}) = a$ for some atomic item a, then $\mathbf{x} = \mathbf{a}$,
2. if $\mu(\mathbf{x})$ is a sequence item, then $\mathbf{x}$ looks like a sequence; if $\mu(\mathbf{x})$ is a sequence item of definite length, then $\mathbf{x}$ is of the form $\langle \mathbf{y}_1, ..., \mathbf{y}_n \rangle$, and
3. if $\mu(\mathbf{x})$ is a function item, then $\mathbf{x}$ looks like a function.

∎

Note that if $\mu(\mathbf{x}) = \perp$, then $\mathbf{x}$ is vacuously transparent and that once an expression is transparent, further rewriting cannot destroy transparency.

The proof of the following two results can be found in [HWW90]. Recall a term is in *normal form* iff no rewrite rules apply.

Theorem 2.7: [HWW90] *If* $\mathbf{x}$ *is an FP expression in normal form then* $\mathbf{x}$ *is transparent.*

Definition 2.8: A set of rewrite rules is *strongly complete* if for all FP expressions $\mathbf{x}$ there is a transparent expression $\mathbf{y}$ such that $\mathbf{x} \to^* \mathbf{y}$. ∎

Theorem 2.9: [HWW90] *The FP rewrite rules are sound and strongly complete.*

We refer to Theorem 2.9 as the *Adequacy Theorem.*

3 The undefinability of apply-to-all

We begin our investigation of the question "Are there enough primitive functions in FP?" by considering apply-to-all. This is a function we would like to have in practice, and it is definable in the version of FP originally considered by Backus [Ba78]. We show that apply-to-all is not definable in FP, but is definable in FP1. The proof will show the subtle difficulties caused by the presence of two-sided sequences in FP.

Let *apall* be the function on D^* such that

$$apall : f : \langle\!\langle x_1, \ldots, x_n \rangle\!\rangle = \langle\!\langle f : x_1, \ldots, f : x_n \rangle\!\rangle, \text{ where we take } f : * = *.$$

We take *apall1* to be the function in $D1^*$ with precisely the same definition. It is not hard to give an FP1 expression **apall1** such that $\mu_1(\textbf{apall1}) = apall1$. We define **apall1** so that

$$\textbf{apall1} = \lambda\textbf{f}, \textbf{x}.(\textbf{cond} : \langle \textbf{null}{:}\textbf{x}, \langle\,\rangle, \textbf{al}{:}\langle \textbf{f}{:}(\textbf{first}{:}\textbf{x}), \textbf{apall1}{:}\textbf{f}{:}(\textbf{tl}{:}\textbf{x})\rangle\rangle).$$

Using standard techniques for expressing λ abstraction and fixpoints in FP (cf. [HWW90]) we can find an FP1 expression with this property.

Note that it is not the case that $\mu(\textbf{apall1}) = apall$. Intuitively, since **apall1** works on a sequence starting with the left end, we can show, for example, that $\mu(\textbf{apall1}{:}\textbf{id}) : \langle\!\langle *, a \rangle\!\rangle = \langle\!\langle \bot, * \rangle\!\rangle$ rather than $\langle\!\langle *, a \rangle\!\rangle$. As we now show, this is in fact an unavoidable problem; there is no FP expression whose meaning is *apall*. We in fact prove a somewhat stronger result that will be useful in our later investigation of full abstraction. We show that *apall* is not definable in FP extended by a parallel *or*.

Let *or* be the finite function item such that $or : x = \bot$ unless x is a sequence of length 2, and

$$or : \langle\!\langle a, b \rangle\!\rangle = \begin{cases} T & \text{if } a = T \text{ or } b = T \\ F & \text{if } a = b = F \\ \bot & \text{otherwise} \end{cases}$$

Let FPO be the result of extending FP by adding a new primitive function symbol **or** and then closing off under application and the formation of sequences, and extending μ so that $\mu(\textbf{or}) = or$. We take FPO1 to be the analogous extension of FP1.

Theorem 3.1: *There is no FPO expression whose meaning is apall.*

Proof: The key step is to show that the finite function $apallKT$, which intuitively captures the meaning of $\textbf{apall} : (\textbf{K} : \textbf{T})$ on four very small sequences, is not definable in FPO. More formally, we define $apallKT$ as follows:

if $\langle\!\langle \bot \rangle\!\rangle \leq_I x$, then $apallKT : x = \langle\!\langle T \rangle\!\rangle$ otherwise

if $\langle\!\langle \bot, *, \bot \rangle\!\rangle \leq_I x$, then $apallKT : x = \langle\!\langle T, *, T \rangle\!\rangle$ otherwise if $\langle\!\langle \bot, * \rangle\!\rangle \leq_I x$, then $apallKT : x = \langle\!\langle T, * \rangle\!\rangle$ otherwise

if $\langle\!\langle *, \bot \rangle\!\rangle \leq_I x$, then $apallKT : x = \langle\!\langle *, T \rangle\!\rangle$ otherwise

$apallKT : x = \bot$

In Appendix B we prove

Lemma 3.2: *There is no expression* $\mathbf{g}$ *in FPO such that* $apallKT \leq_I \mu(\mathbf{g})$.

It easily follows that **apall** is not definable in FPO, since if $\mathbf{g}$ is an expression in FPO such that $\mu(\mathbf{g}) = apall$, then we must have $apallKT \leq_I \mu(\mathbf{g} : (\mathbf{K} : \mathbf{T}))$. ∎

FPO has an even more fundamental lack of expressive power than that characterized by Theorem 3.1. It is impossible to distinguish $\langle\!\langle \bot, * \rangle\!\rangle$ from $\langle\!\langle *, \bot \rangle\!\rangle$ in FP. More precisely, let *ef* and *el* (for "exists a first element" and "exists a last element") be functions on D^* such that:

$$ef : x = \begin{cases} T & \text{if } \langle\!\langle \bot, * \rangle\!\rangle \leq_I x \\ \bot & \text{otherwise} \end{cases}$$
$$el : x = \begin{cases} T & \text{if } \langle\!\langle *, \bot \rangle\!\rangle \leq_I x \\ \bot & \text{otherwise.} \end{cases}$$

Notice the *ef* and *el* are definable if we augment FP by a primitive function symbol **apallKT** such that $\mu(\mathbf{apallKT}) = apallKT$. For example, we have $ef = \mu(\lambda\mathbf{x}.(\mathbf{first} : \mathbf{apallKT} : \mathbf{x}))$. Thus, *ef* and *el* are in some sense more fundamental that $apallKT$ (or *apall*). As we now show, neither one is definable in FPO.

Theorem 3.3: *There is no FPO expression whose meaning is ef or el.*

Proof: See Appendix B. ∎

4 Full abstraction and expressive completeness

In this section we formally define the notions of full abstraction and expressive completeness. We want to define these notions, not just for FP, but for a number of variants and extensions of FP. To do this carefully, we need to provide a few more details about the variants we consider. As we mentioned earlier, each variant F of FP that we consider is obtained by adding or removing elements from the set $\mathcal{F}$ of primitive function symbols. Thus, the set of atoms is the same for all variants, as is the set of observables. For each variant F, there is a corresponding semantic domain E^*, and a semantic function ν mapping expressions in F to elements of E^*. All the semantic domains agree on the set of atomic items. We assume that all the atomic items in E^* are finite, as is every *step function* determined by every pair x, y of finite elements, where the step function determined by x, y, denoted $[x \to y]$, is the function f such that when applied to any element $x' \geq x$ yields y as the answer and when applied to anything else yields $\perp$. For the remainder of this section, we assume that F, E^*, and ν satisfy the properties described above.

As usual, we say a *context* is an expression with a single "hole" in it. If $\mathbf{C}$ is a context, we write $\mathbf{C}[\mathbf{x}]$ for the result of filling the hole in context $\mathbf{C}$ by $\mathbf{x}$. For example, in FP, if $\mathbf{C}$ is the context **al**:$\langle$—$, \langle\mathbf{a}\rangle\rangle$, then $\mathbf{C}[\mathbf{b}]$ is **al**:$\langle\mathbf{b}, \langle\mathbf{a}\rangle\rangle$.

When we talk about full abstraction, it is really with respect to a given domain and language. In the standard definition of full abstraction (for example, the one given in [Pl77]), the language is typed. In particular, there is a type **prog** that is the type of programs. In this case, we say two terms are *behaviorally equivalent*, and write $\mathbf{x} \approx \mathbf{y}$, if in all contexts $\mathbf{C}$ of type **prog** the expressions $\mathbf{C}[\mathbf{x}]$ and $\mathbf{C}[\mathbf{y}]$ have the same operational behavior. (That is, $\mathbf{C}[\mathbf{x}]$ rewrites to a normal form iff $\mathbf{C}[\mathbf{y}]$ rewrites to a normal form, and if one rewrites to a normal form, they both rewrite to the same one.) A fully abstract language is one where $\mathbf{x} \approx \mathbf{y}$ iff $\nu(\mathbf{x}) = \nu(\mathbf{y})$, where ν is the semantic function for the language. Thus, full abstraction says that two objects are semantically equal iff they are behaviorally equivalent.

Now FP and its variants are untyped, so there is no exact analogue to expressions of type **prog**. However, we can get an analogue by considering expressions whose meaning is an atomic item. More formally, we define full abstraction for all of the variants F of FP considered in this paper as follows.

Definition 4.1: Given a context $\mathbf{C}$ and an expression $\mathbf{x}$, we say $\mathbf{C}$ is an *atomic context* for $\mathbf{x}$ if $\nu(\mathbf{C}[\mathbf{x}])$ is an atomic item. We say two F expressions $\mathbf{x}$ and $\mathbf{y}$ are *behaviorally equivalent*, and again write $\mathbf{x} \approx \mathbf{y}$, if (a) for all contexts $\mathbf{C}$, $\mathbf{C}$ is an atomic context for $\mathbf{x}$ iff $\mathbf{C}$ is an atomic context for $\mathbf{y}$, and (b) if $\mathbf{C}$ is an atomic context for $\mathbf{x}$, then $\nu(\mathbf{C}[\mathbf{x}]) = \nu(\mathbf{C}[\mathbf{y}])$. Finally, we say that *F is fully abstract with respect to E^** if for all F expressions $\mathbf{x}$ and $\mathbf{y}$, we have $\mathbf{x} \approx \mathbf{y}$ iff $\nu(\mathbf{x}) = \nu(\mathbf{y})$. ∎

Note that by Theorem 2.5, for FP we have that if $\mu(\mathbf{C}[\mathbf{x}]) = a$ for some atomic item a, then $\mathbf{C}[\mathbf{x}] \to^* \mathbf{a}$. Thus, if $\mathbf{x} \approx \mathbf{y}$, then in all atomic contexts both $\mathbf{x}$ and $\mathbf{y}$ rewrite to the same normal form. Since analogues to Theorem 2.5 hold for all the variants of FP we consider, the same remark holds for them as well.

The intuition behind the notion of expressive completeness is that all computable elements are definable. But what is a computable element? Following Plotkin [Pl77], we take the computable elements to be the least upper bounds of recursively enumerable sets of finite elements. To make this precise, we need a few more definitions.

We first need some way of encoding the finite elements of E^* as integers. Suppose we have a function *code* mapping finite elements in E^* to observable items; we call *code*(d) the *code* of d. The details of *code* do not matter for the discussion that follows. (We define *code* carefully on the finite elements in D^* and $D1^*$ in Appendix C; we define *code* on the variants of FP as we discuss them in the appendix.) All that matters is that the encoding is such that we can easily write expressions in F that test whether an observable item encodes bottom (i.e., we can write a function $\mathbf{f}$ such that $\mu(\mathbf{f}){:}x = T$ if $x = code(\bot)$ and $\mu(\mathbf{f}){:}x = F$ if $x \neq \bot$ and $x \neq code(\bot)$). Similarly, we can write expressions in F to test whether an observable item encodes an atomic item, a finite sequence item of definite length, a finite sequence item of indefinite length, a function item, etc.

We next define a function taking observables to integers. Recall that we assumed that there were only finitely many atomic items. For convenience, suppose they are $a_1, ..., a_N$. Let p_i denote the i^{th} prime. Let enc map observable items to integers as follows:

$$\begin{aligned} &enc(a_i) = p_{i+1} \\ &enc(\langle\!\langle x_1, ..., x_n \rangle\!\rangle) = 2p_2^{enc(x_1)} ... \, p_{n+1}^{enc(x_n)}. \end{aligned}$$

Definition 4.2: A sequence $x_1, x_2, ...$ of finite elements in E^* is said to be *recursive* if there is a recursive function f on the integers such that $f(i) = enc(code(x_i))$. An element of E^* is said to be *computable* if it is the sup of an increasing recursive sequence of finite elements of E^*. We say that F is *expressively complete with respect to* E^* if for every computable element x in E^*, there is an expression $\mathbf{x}$ in F such that $\nu(\mathbf{x}) = x$. We say that F is *f.e. (finite element) complete with respect to* E^* if for every finite element of E^*, there is an expression $\mathbf{x}$ in F such that $\nu(\mathbf{x}) = x$. ▮

Clearly expressive completeness implies f.e. completeness. F.e. completeness in turn implies full abstraction for algebraic domains under some reasonable assumptions. Since we have only defined full abstraction for variants of FP, we cannot make this statement formal, but we do prove it for variants of FP.

Proposition 4.3: *If F is a variant of FP and F is f.e. complete with respect to* E^*, *then F is fully abstract with respect to* E^*.

Proof: If $\nu(\mathbf{x}) \neq \nu(\mathbf{y})$, then since E^* is algebraic, without loss of generality there is some finite element x_0 such that $x_0 \leq_I \nu(\mathbf{x})$ and $x_0 \not\leq_I \nu(\mathbf{y})$. Now consider the step function $f = [x_0 \rightarrow a_0]$ where a_0 is an atomic item in E^*. Our assumptions on variants of FP imply that f is finite. Since F is f.e. complete with respect to E^*, there is an F expression $\mathbf{f}$ such that $\nu(\mathbf{f}) = f$. Consider the context $\mathbf{C} = \mathbf{f}{:}$—. We have $\nu(\mathbf{C}[\mathbf{x}]) = a_0$ and $\nu(\mathbf{C}[\mathbf{y}]) = \bot$. Thus $\mathbf{x} \not\approx \mathbf{y}$. ▮

In general, we will prove full abstraction by showing f.e. completeness. We remark that Plotkin also proves f.e. completeness in the course of proving full abstraction for PCF. There is a question of which of full abstraction or f.e. completeness is the more important notion in an algebraic domain. The answer depends somewhat on one's point of view. If the language is viewed as fixed, then arguably full abstraction is the more important notion, since it ensures that the domain does not have any "extraneous" finite elements. If the domain is viewed as fixed, then f.e. completeness is arguably more important, since it means that the language is sufficiently powerful to express all the finite elements of the domain. In this paper, we use prove f.e. completeness for our positive results, and lack of full abstraction in our negative results, thus proving the stronger result in each case.

We remark that for *extensional* models of PCF, full abstraction and f.e. completeness are known to be equivalent [Mi77,St90]. There are non-extensional algebraic models of PCF that are f.e. complete but not fully abstract.

5 FP and FP1 are not fully abstract

Our goal in this section is to prove that neither FP nor FP1 is fully abstract. We actually prove that FPA is not fully abstract, where FPA is the extension of FP to include the primitive function symbol **apall**. The result for FP and FP1 follows readily.

The fact that FPA is not fully abstract follows from two lemmas. For the first, let FPAO be FP extended to include both **apall** and **or**, as well as the primitive function symbols **ef** and **el** introduced in Section 3. We extend μ in the obvious way to FPAO. In Appendix A, we provide sound rewrite rules for **apall**, **or**, **ef**, and **el**; the techniques of [HWW90] show that these rules are strongly complete for FPAO.

Lemma 5.1: *If* $\mathbf{x}$ *is an FPAO (resp. FPA, FPO) expression in normal form such that* $\mu(\mathbf{x})$ *is a function item and* $\mu(\mathbf{x}{:}\mathbf{y}') = \bot$ *for all FPAO (resp. FPA, FPO) expressions* $\mathbf{y}'$*, then* $\mathbf{x} \approx \mathbf{K}{:}\Omega$*.*

Proof: Define a new meaning function $\mu_{\mathbf{x}}$ that acts just like μ except that $\mu_{\mathbf{x}}(\mathbf{x}) = \mu(\mathbf{K}{:}\Omega)$. More formally, $\mu_{\mathbf{x}}(\mathbf{f}) = \mu(\mathbf{f})$ if $\mathbf{f}$ is an atom or primitive function symbol,

$$\mu_{\mathbf{x}}(\langle \mathbf{y}_1, ..., \mathbf{y}_n \rangle) = \langle\!\langle \mu_{\mathbf{x}}(\mathbf{y}_1), ..., \mu_{\mathbf{x}}(\mathbf{y}_n) \rangle\!\rangle,$$

$$\mu_{\mathbf{x}}(\mathbf{y}{:}\mathbf{z}) = \begin{cases} \mu(\mathbf{K}{:}\Omega) & \text{if } \mathbf{y}{:}\mathbf{z} = \mathbf{x} \\ \mu_{\mathbf{x}}(\mathbf{y}) : \mu_{\mathbf{x}}(\mathbf{z}) & \text{otherwise.} \end{cases}$$

The function $\mu_{\mathbf{x}}$ has three important properties:

1. $\mu_{\mathbf{x}}(\mathbf{y}) \leq_I \mu(\mathbf{y})$ for all FPAO (resp. FPA, FPO) expressions $\mathbf{y}$,
2. $\mu_{\mathbf{x}}(\mathbf{C}[\mathbf{x}]) \leq_I \mu(\mathbf{C}[\mathbf{K}{:}\Omega])$ for all contexts $\mathbf{C}$,
3. if $\mathbf{y} \to^* \mathbf{z}$, then $\mu_{\mathbf{x}}(\mathbf{z}) \leq_I \mu_{\mathbf{x}}(\mathbf{y})$.

Properties (1) and (2) can be checked by a straightforward induction on the structure of $\mathbf{y}$ and the structure of $\mathbf{C}$ respectively. To see that property (3) holds, note that, as usual, straightforward inductions on the length of the reduction and the structure of $\mathbf{y}$ show that it suffices to consider the case where $\mathbf{y} \to \mathbf{z}$, and this is an instance of a rewrite rule. Again we must consider the rewrite rules on a case-by-case basis. For example, if $\mathbf{y}$ is of the form $\mathbf{apply}{:}\langle \mathbf{f}, \mathbf{y}' \rangle$ and $\mathbf{z}$ is of the form $\mathbf{f}{:}\mathbf{y}'$, then we know $\mathbf{y} \neq \mathbf{x}$ (since $\mathbf{x}$

is in normal form and $\mathbf{y}$ is not), so it is easy to see that $\mu_{\mathbf{x}}(\mathbf{y}) = \mu(\mathbf{apply}) : \langle\langle \mu_{\mathbf{x}}(\mathbf{f}), \mu_{\mathbf{x}}(\mathbf{y}')\rangle\rangle = \mu_{\mathbf{x}}(\mathbf{f}) : \mu_{\mathbf{x}}(\mathbf{y}')$. Thus $\mu_{\mathbf{x}}(\mathbf{z}) \leq_I \mu_{\mathbf{x}}(\mathbf{y})$. (Note that if $\mathbf{x} = \mathbf{z}$ then we might have $\mu_{\mathbf{x}}(\mathbf{z}) <_I \mu_{\mathbf{x}}(\mathbf{y})$.) Similar arguments work in all cases except when $\mathbf{y}$ is of the form $\mathbf{f}{:}\mathbf{y}''{:}\mathbf{y}'$ and $\mathbf{f}$ is **comp**, **cons**, or $\mathbf{K}$. We consider the case where $\mathbf{y}$ is of the form $\mathbf{comp}{:}\langle \mathbf{f}_1, \mathbf{f}_2\rangle{:}\mathbf{y}'$ here; the other cases are similar and left to the reader. If $\mathbf{x} \neq \mathbf{comp}{:}\langle \mathbf{f}_1, \mathbf{f}_2\rangle$, then similar arguments to that used in the case of **apply** easily show that $\mu_{\mathbf{x}}(\mathbf{z}) \leq_I \mu_{\mathbf{x}}(\mathbf{y})$. If $\mathbf{x} = \mathbf{comp}{:}\langle \mathbf{f}_1, \mathbf{f}_2\rangle$, then $\mu(\mathbf{y}) = \perp$ by our assumption that $\mu(\mathbf{x}{:}\mathbf{y}') = \perp$ for all FPAO (resp. FPA, FPO) expressions $\mathbf{y}'$. Thus, since FPAO (resp. FPA, FPO) rewriting is sound, we must have $\mu(\mathbf{z}) = \perp$. Since $\mu_{\mathbf{x}}(\mathbf{z}) \leq_I \mu(\mathbf{z})$, we also have $\mu_{\mathbf{x}}(\mathbf{z}) = \perp$, so $\mu_{\mathbf{x}}(\mathbf{z}) \leq_I \mu_{\mathbf{x}}(\mathbf{y})$.

We are now ready to prove that $\mathbf{x} \approx \mathbf{K}{:}\Omega$. If $\mu(\mathbf{C}[\mathbf{K}{:}\Omega]) = b$ for some atom b, then $\mu(\mathbf{C}[\mathbf{x}]) = b$ since $\mu(\mathbf{C}[\mathbf{K}{:}\Omega]) \leq_I \mu(\mathbf{C}[\mathbf{x}])$. Conversely, if $\mu(\mathbf{C}[\mathbf{x}]) = b$ for some atom $\mathbf{b}$, then (by Adequacy for FPAO (resp. FPA, FPO)), $\mathbf{C}[\mathbf{x}] \rightarrow^* \mathbf{b}$ Thus, $b = \mu_{\mathbf{x}}(\mathbf{b}) \leq_I \mu_{\mathbf{x}}(\mathbf{C}[\mathbf{x}]) \leq_I \mu(\mathbf{C}[\mathbf{K}{:}\Omega])$. Therefore, $\mu(\mathbf{C}[\mathbf{K}{:}\Omega]) = b = \mu(\mathbf{C}[\mathbf{x}])$. ∎

The second lemma is an analogue of a result proved by Plotkin for PCF [Pl77], and shows that parallel *or* is not definable in FPA. The proof requires rather extensive machinery, and so is deferred to Appendix B.

Lemma 5.2: *There is no FPA expression* $\mathbf{g}$ *such that* $or \leq_I \mu(\mathbf{g})$.

Theorem 5.3: *Neither FPA nor FP is fully abstract with respect to* D^**, and FP1 is not fully abstract with respect to* $D1^*$.

Proof: Let $\mathbf{f}$ be an FP1 expression that tests to see if its argument is greater than *or*. That is, $\mathbf{f}{:}\mathbf{g} \rightarrow^* \mathbf{T}$ iff $\mu(\mathbf{g}){:}\langle\langle T, \perp\rangle\rangle = T$, $\mu(\mathbf{g}) : \langle\langle \perp, T\rangle\rangle = T$, and $\mu(\mathbf{g}) : \langle\langle F, F\rangle\rangle = F$. We can take $\mathbf{f}$ to be the expression

$$\lambda \mathbf{g}.(\mathbf{cond}{:}\langle \mathbf{g}{:}\langle \mathbf{T}, \Omega\rangle, \mathbf{cond}{:}\langle \mathbf{g}{:}\langle \Omega, \mathbf{T}\rangle, \mathbf{cond}{:}\langle \mathbf{g}{:}\langle \mathbf{F}, \mathbf{F}\rangle, \Omega, \mathbf{T}\rangle, \Omega\rangle, \Omega\rangle.$$

We leave it to the reader to check that when the lambda is eliminated $\mathbf{f}$ is in normal form. By Lemma 5.2, there is no FPA expression (and hence no FP or FP1 expression) $\mathbf{g}$ such that $or \leq_I \mu(\mathbf{g})$, so we must have $\mu(\mathbf{f}{:}\mathbf{g}) = \perp$ for every expression $\mathbf{g}$ in FPA, FP, or FP1. Thus, by Lemma 5.1, $\mathbf{f} \approx \mathbf{K}{:}\Omega$. Since $\mu(\mathbf{f}) \neq \mu(\mathbf{K}{:}\Omega)$ (and similarly $\mu_1(\mathbf{f}) \neq \mu_1(\mathbf{K}{:}\Omega)$), it follows that neither FPA nor FP is fully abstract with respect to D^*, and FP1 is not fully abstract with respect to $D1^*$. ∎

In the next few sections, we examine how we can extend FP and FP1 in order to get full abstraction.

6 Extending FP1 to get full abstraction and expressive completeness

In [Pl77], Plotkin shows that by adding a parallel conditional facility to PCF we get full abstraction. He also observed that using parallel conditional, parallel *or* is definable.

Curien [Cu86] and Abramsky [Ab90] later showed that in order to get full abstraction for PCF, it suffices to add parallel *or*. Stoughton [St91] showed that parallel if and or are each definable in terms of the other. Here we show that we can get full abstraction for FP1 by adding parallel *or*; however, we cannot get full abstraction for FP in this way! We then go on to show that using techniques similar to Plotkin's, we can further extend FP1 to get expressive completeness.

Given a finite element x in $D1^*$, let $\mathbf{code}(x)$ represent $code(x)^e$, the observable expression corresponding to the observable item $code(x)$.

Theorem 6.1: *There is an FPO1 expression* $\mathbf{E}_1$ *such that for all finite elements* x *in* $D1^*$*, we have* $\mu_1(\mathbf{E}_1{:}\mathbf{code}(x)) = x$*.*

Proof: See Appendix C. ∎

Corollary 6.2: *FPO1 is f.e. complete with respect to* $D1^*$*.*

By applying Proposition 4.3, we have

Corollary 6.3: *FPO1 is fully abstract with respect to* $D1^*$*.*

As mentioned above, the analogue to this result does *not* hold for FP. Even though adding parallel *or* to FP1 is sufficient to make it fully abstract, the addition of parallel or to FP (whose domain is not consistently complete) does not suffice to make FP fully abstract.

Theorem 6.4: *FPO is not fully abstract with respect to* D^**.*

Proof: Our first step is to define a function **geapallKT** with the property that $\mu(\mathbf{geapallKT}) : g = T$ if $apallKT \leq_I g$, where $apallKT$ is the function defined in Section 3, and $\mu(\mathbf{geapallKT}) : g = \bot$ if $apallKT \not\leq_I g$. In order to do this, **geapallKT** must check if its argument satisfies the following five properties:

- $\langle\!\langle T, * \rangle\!\rangle \leq_I g : \langle\!\langle \bot, * \rangle\!\rangle$,
- $\langle\!\langle *, T \rangle\!\rangle \leq_I g : \langle\!\langle *, \bot \rangle\!\rangle$,
- $\langle\!\langle T, * \rangle\!\rangle \leq_I tr : (g : \langle\!\langle \bot, *, \bot \rangle\!\rangle)$,
- $\langle\!\langle *, T \rangle\!\rangle \leq_I tl : (g : \langle\!\langle \bot, *, \bot \rangle\!\rangle)$,
- $\langle\!\langle T \rangle\!\rangle \leq_I g : \langle\!\langle \bot \rangle\!\rangle)$.

If the argument g does satisfy all these properties, then $\mu(\mathbf{geapallKT}) : g$ returns T; otherwise it returns $\bot$.

The following definition of **geapallKT** does the trick:

$\lambda\mathbf{g}.($

$\quad\mathbf{cond} : \langle\mathbf{first} : (\mathbf{g} : (\mathbf{al} : \langle\Omega, \Omega\rangle)),$

$\quad\quad\mathbf{cond} : \langle\mathbf{last} : (\mathbf{g} : (\mathbf{ar} : \langle\Omega, \Omega\rangle)),$

$\quad\quad\quad\mathbf{cond} : \langle\mathbf{first} : (\mathbf{tr} : (\mathbf{g} : (\mathbf{al} : \langle\Omega, \mathbf{ar} : \langle\Omega, \Omega\rangle\rangle))),$

$\quad\quad\quad\quad\mathbf{cond} : \langle\mathbf{last} : (\mathbf{tl} : (\mathbf{g} : (\mathbf{al} : \langle\Omega, \mathbf{ar} : \langle\Omega, \Omega\rangle\rangle))),$

$\quad\quad\quad\quad\quad\mathbf{cond} : \langle\mathbf{null} : (\mathbf{tl} : (\mathbf{g} : \langle\Omega\rangle)), \mathbf{T}, \Omega\rangle, \Omega\rangle, \Omega\rangle, \Omega\rangle, \Omega\rangle)$

We leave it to the reader to check that when the lambda is eliminated, the expression is indeed in normal form. From Lemma 3.2, we know that there is no FPO definable function $\mathbf{g}$ such that $apallKT \leq_I \mu(\mathbf{g})$. Thus, we must have $\mu(\mathbf{geapallKT} : \mathbf{g}) = \perp$ for all FPO expressions $\mathbf{g}$. From Lemma 5.1, it follows that $\mathbf{geapallKT} \approx \mathbf{K} : \Omega$. But $\mu(\mathbf{geapallKT}) \neq \mu(\mathbf{K} : \Omega)$ since $\mu(\mathbf{geapallKT}) : apallKT = T$. Therefore, FPO is not fully abstract. ∎

We remark that we show in Lemma B.5 below that the same result holds even if we add to FPO the function symbols **ef** and **el** (whose meaning is given in Section 3).

In order to get expressive completeness for PCF, Plotkin adds a parallel conditional and an "existential" operator $\exists$ such that $\nu(\exists)(f) = F$ if $f(\perp) = F$ and $\nu(\exists)(f) = T$ if $f(n) = T$ for some natural number n. We use observable elements rather than natural numbers to get the same effect in our domain. Thus, we take FP1* to be FPO1 augmented with the primitive function symbol **exists**, where

$$\mu_1(\mathbf{exists}) : f = \begin{cases} F & \text{if } f : \perp = F \\ T & \text{if } f : x = T \text{ for some observable item } x \\ \perp & \text{otherwise.} \end{cases}$$

Using **exists**, we can define an FP1* expression $\mathbf{sup}_1$ that takes least upper bounds of sequences of elements.

Theorem 6.5: *There is an expression* $\mathbf{sup}_1$ *in FP1* such that if* $\langle z_1, \ldots, z_k\rangle$ *is an increasing sequence of finite elements in* $D1^*$*, then*

$$\mu_1(\mathbf{sup}_1) : \langle\langle code(z_1), \ldots, code(z_k), *\rangle\rangle = z_k.$$

Proof: See Appendix C. ∎

Corollary 6.6: *FP1* is expressively complete with respect to* $D1^*$*.*

Proof: Let $x = \sup\{x_1, x_2, x_3, \ldots\}$ where $x_1, x_2, \ldots$ is a recursive sequence of finite elements in $D1^*$ and $x_i \leq_I x_{i+1}$. We must show that there is an expression whose meaning is x. Using well-understood techniques it can be shown that recursive sequences can be encoded in FP1*; thus, there is an expression $\mathbf{e}_0$ such that $\mu_1(\mathbf{e}_0) = \langle\!\langle \mathit{code}(x_1), \mathit{code}(x_2), \ldots \rangle\!\rangle$. By the continuity of $\mu_1(\mathbf{sup_1})$, we have that

$$\begin{aligned} & \mu_1(\mathbf{sup_1} : \mathbf{e}_0) \\ = \; & \mu_1(\mathbf{sup_1}) : \mu_1(\mathbf{e}_0) \\ = \; & \bigsqcup\{\mu_1(\mathbf{sup_1}) : \langle\!\langle \mathit{code}(x_1), \ldots, \mathit{code}(x_n), * \rangle\!\rangle | n = 1, 2, 3 \ldots\} \\ = \; & \bigsqcup\{x_1, x_2, x_3, \ldots\} \\ = \; & x, \end{aligned}$$

as desired. ▮

7 Full abstraction and expressive completeness for nondeterministic FP

In the previous section we showed that by adding a certain parallel facilities to FP1 we were able to get full abstraction and expressive completeness. However, the obvious analogues of these extensions for FP do not give full abstraction or expressive completeness. One major difference between FP and FP1 is that the domain for FP1 is consistently complete.

To investigate the effect of consistent completeness, we force the domain to be consistently complete by adding unions to the domain. Formally, we define a domain ND^* in a manner analogous to the way we defined D^*. We construct a sequence of finite domains $ND_0, ND_1, ND_2, \ldots$. We take $ND_0 = D_0$, and let ND_{n+1} consists of ND_n together with all the continuous functions from ND_n to ND_n, sequence items of the appropriate form and length of elements in ND_n, and sets of elements in ND_n. Let $ND = \cup_n ND_n$. We can define an ordering $\leq_{ND}$ on ND in a straightforward way; we omit details here. Let ND^* consist of all of non-empty, downward-closed subsets (with respect to $\leq_{ND}$) of ND. Notice that, unlike D^*, the sets in ND^* are not necessarily directed. By doing this, we in effect permit unions, thereby ensuring consistent completeness.

We extend FP to NFP$^-$ by adding one more primitive function symbol **union**. We extend μ to μ_N by defining $\mu_N(\mathbf{union}) : \langle\!\langle x_1, ..., x_n \rangle\!\rangle = x_1 \cup ... \cup x_n$. ND^* is easily seen to be consistently complete; the sup of any two elements of ND^* is just their union.

NFP$^-$ does allow a great deal of parallel evaluation. In particular, the parallel *or* function *or* is easily seen to be definable in NFP$^-$, since we can now work from both ends of a sequence of length two to test for **T**. The following definition does the trick (the first test checks that the sequence has length two):

$$\begin{aligned}&\lambda \mathbf{x}.\mathbf{cond} : \langle \mathbf{null} : (\mathbf{tl} : (\mathbf{tl} : \mathbf{x})),\\ &\quad \mathbf{union} : \langle \mathbf{cond} : \langle \mathbf{first} : \mathbf{x}, \mathbf{T}, \mathbf{cond} : \langle \mathbf{last} : \mathbf{x}, \mathbf{T}, \mathbf{F}\rangle,\\ &\qquad \mathbf{cond} : \langle \mathbf{last} : \mathbf{x}, \mathbf{T}, \mathbf{cond} : \langle \mathbf{first} : \mathbf{x}, \mathbf{T}, \mathbf{F}\rangle\rangle,\\ &\qquad \Omega\rangle\end{aligned}$$

However, we can easily extend the techniques of Theorem 3.3 to show that neither **ef** nor **el** is definable in NFP$^-$. It follows that **apallKT** and **apall** are not definable in NFP$^-$ either. By combining the ideas of Theorem 3.3 and Theorem 5.3, we can then show that NFP$^-$ is not fully abstract. However, the inability to define **ef** and **el** is all that keeps NFP$^-$ from being fully abstract and expressively complete. Let NFP be the result of adding **ef** and **el** to NFP$^-$.

As we show in Appendix C, we can extend the function *code* to *ND* in a straightforward way, so that $code(x)$ is an observable item for each element $x \in ND$. Again, we take $\mathbf{code}(x)$ to be the observable element corresponding to the observable item $code(x)$.

Theorem 7.1: *There is an NFP expression* $\mathbf{E}_N$ *such that for all for all finite elements* x *in* ND*, *we have* $\mu_N(\mathbf{E}_N{:}\mathbf{code}(x)) = x$.

Proof: See Appendix C. ∎

Corollary 7.2: *NFP is f.e. complete with respect to* ND*.

By applying Proposition 4.3, we immediately get

Corollary 7.3: *NFP is fully abstract with respect to* ND*.

We now turn our attention to expressive completeness. As for FP1*, the key step in proving expressive completeness for NFP is the ability to take the sup of an increasing recursive sequence of finite elements. Using **union**, we can do this in a straightforward way in NFP.

Theorem 7.4: *There is an expression* $\mathbf{sup}_N$ *in NFP such that if* $\langle z_1, \ldots, z_k\rangle$ *is an increasing sequence of finite elements in* ND*, *then*

$$\mu_1(\mathbf{sup}_N) : \langle\!\langle code(z_1), \ldots, code(z_k), *\rangle\!\rangle = z_k.$$

Proof: See Appendix C. ∎

Now, just as in the case of FP1*, we get

Corollary 7.5: *NFP is expressively complete with respect to* ND*.

We also note that if we extend FP1 by adding **union** and extend $D1^*$ by allowing unions, the resulting language NFP1 is fully abstract and expressively complete with respect to the domain. We do not need to explicitly add **ef** or **el** to the language: The problem of distinguishing $\langle\!\langle *, \bot\rangle\!\rangle$ from $\langle\!\langle \bot, *\rangle\!\rangle$ does not arise in NFP1 since $\langle\!\langle *, \bot\rangle\!\rangle$ is not in the domain.

8 Extending FP to get full abstraction and expressive completeness

In the previous section we showed that by extending FP with nondeterminism we could get a language that was fully abstract and expressively complete. However, in extending to nondeterminism, we also extended the domain D^* to ND^*, which is consistently complete. The reader may wonder at this point if we can get full abstraction or expressive completeness for an extension of FP that still has semantic domain D^*. Using the ideas from the proofs in the previous sections, we show that this can be done.

Our proofs of full abstraction for FPO1 and NFP both depend on constructing expressions $\mathbf{E}_1$ and $\mathbf{E}_N$, respectively, which map $code(x)$ to x for a finite item x. We cannot construct such an expression in FP. Thus, in order to get full abstraction, we simply extend FP by adding with a new primitive function symbol $\mathbf{E}$ with this property.

Let FPE be the result of augmenting FP with **ef**, **el**, and the further primitive function symbols $\mathbf{E}$. We define μ on $\mathbf{E}$ so that $\mu(\mathbf{E}) : y = x$ if $y = code(x)$. (We give the precise definition of $\mu(\mathbf{E})$ in Appendix C, after we have given more details of the function *code*.) Of course, the definition of $\mathbf{E}$ guarantees that FPE is f.e. complete, hence fully abstract.

The proofs of expressive completeness for FP1* and NFP show that the crucial factor in getting complete expressiveness is the ability to take sups of increasing recursive sequences of finite elements. Let FPE* be the result of extending FPE with the primitive function symbol **sup**. Further extend μ so that if $x_1, \ldots, x_k$ is an increasing sequence of finite elements in D^*, then $\mu(\mathbf{sup}) : \langle\langle code(x_1), ..., code(x_k), *\rangle\rangle = x_k$. (Again, we give the precise definition of $\mu(\mathbf{sup})$ in Appendix C.) Arguments similar to those used for FP1* can now be used to show that FPE* is expressively complete.

We do not want to claim that FPE and FPE* are natural extensions of FP. However, they both have recursive syntax, r.e. rewrite rules (see Appendix C for details of the rewrite rules), and a finite set of primitive functions. Moreover, these extensions do indicate exactly what is required of a language in order to get full abstraction and expressive completeness.

9 Conclusions

We have investigated the question of full abstraction and expressive completeness for FP and a number of its variants. By considering these issues for a number of related domains, we feel we have obtained a better understanding of what is required of a language and semantics in order to guarantee full abstraction and expressive completeness. We hope that the results of this paper will lead to a framework for a more general theory of constructing good programming languages, i.e., ones with natural syntax, natural rewrite rules, enough rewrite rules, and reasonable expressive power.

A Rewrite Rules

In this appendix, we provide the rewrite rules for FP and a number of extensions of FP we have considered in this paper. For FP as we have defined it, the rewrite rules are:

$\mathbf{al}{:}\langle \mathrm{x}, \langle \mathbf{y}_1, \ldots, \mathbf{y}_n\rangle\rangle \to \langle \mathrm{x}, \mathbf{y}_1, \ldots, \mathbf{y}_n\rangle$
$\mathbf{ar}{:}\langle\langle \mathbf{y}_1, \ldots, \mathbf{y}_n\rangle, \mathrm{x}\rangle \to \langle \mathbf{y}_1, \ldots, \mathbf{y}_n, \mathrm{x}\rangle$
$\mathbf{first}{:}\langle \mathrm{x}_1, \ldots, \mathrm{x}_n\rangle \to \mathrm{x}_1 \ \ (n \geq 1)$
$\mathbf{first}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle) \to \mathrm{x}$
$\mathbf{first}{:}(\mathbf{ar}{:}\langle \mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle, \mathrm{z}\rangle) \to \mathrm{x}$
$\mathbf{first}{:}(\mathbf{ar}{:}\langle \mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle, \mathrm{z}\rangle) \to \ \mathbf{first}{:}(\mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle)$
$\mathbf{last}{:}\langle \mathrm{x}_1, \ldots, \mathrm{x}_n\rangle \to \mathrm{x}_n \ \ (n \geq 1)$
$\mathbf{last}{:}(\mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle) \to \mathbf{y}$
$\mathbf{last}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{ar}{:}\langle \mathbf{y}, \mathrm{z}\rangle\rangle) \to \mathrm{z}$
$\mathbf{last}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{al}{:}\langle \mathbf{y}, \mathrm{z}\rangle\rangle) \to \mathbf{last}{:}(\mathbf{al}{:}\langle \mathbf{y}, \mathrm{z}\rangle)$
$\mathbf{tl}{:}\langle \mathrm{x}_1, \ldots, \mathrm{x}_n\rangle \to \langle \mathrm{x}_2, \ldots, \mathrm{x}_n\rangle \ \ (n \geq 1)$
$\mathbf{tl}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{al}{:}\langle \mathbf{y}, \mathrm{z}\rangle\rangle) \to \mathbf{al}{:}\langle \mathbf{y}, \mathrm{z}\rangle$
$\mathbf{tl}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{ar}{:}\langle \mathbf{y}, \mathrm{z}\rangle\rangle) \to \mathbf{ar}{:}\langle \mathbf{y}, \mathrm{z}\rangle$
$\mathbf{tl}{:}(\mathbf{ar}{:}\langle \mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle, \mathrm{z}\rangle) \to \mathbf{ar}{:}\langle \mathbf{y}, \mathrm{z}\rangle$
$\mathbf{tl}{:}(\mathbf{ar}{:}\langle \mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle, \mathrm{z}\rangle) \to \ \mathbf{ar}{:}\langle \mathbf{tl}{:}(\mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle), \mathrm{z}\rangle$
$\mathbf{tr}{:}\langle \mathrm{x}_1, \ldots, \mathrm{x}_n\rangle \to \langle \mathrm{x}_1, \ldots, \mathrm{x}_{n-1}\rangle \ \ (n \geq 1)$
$\mathbf{tr}{:}(\mathbf{ar}{:}\langle \mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle, \mathrm{z}\rangle) \to \mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle$
$\mathbf{tr}{:}(\mathbf{ar}{:}\langle \mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle, \mathrm{z}\rangle) \to \mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle$
$\mathbf{tr}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{ar}{:}\langle \mathbf{y}, \mathrm{z}\rangle\rangle) \to \mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle$
$\mathbf{tr}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{al}{:}\langle \mathbf{y}, \mathrm{z}\rangle\rangle) \to \ \mathbf{al}{:}\langle \mathrm{x}, \mathbf{tr}{:}(\mathbf{al}{:}\langle \mathbf{y}, \mathrm{z}\rangle)\rangle$
$\mathbf{null}{:}\langle\ \rangle \to \mathbf{T}$
$\mathbf{null}{:}\mathbf{a} \to \mathbf{F}$ (for any atom $\mathbf{a}$)
$\mathbf{null}{:}\langle \mathrm{x}_1, \ldots, \mathrm{x}_n\rangle \to \mathbf{F} \ \ (n \geq 1)$
$\mathbf{null}{:}(\mathbf{al}{:}\langle \mathrm{x}, \mathbf{y}\rangle) \to \mathbf{F}$
$\mathbf{null}{:}(\mathbf{ar}{:}\langle \mathrm{x}, \mathbf{y}\rangle) \to \mathbf{F}$
$\mathbf{cons}{:}\langle \mathbf{f}_1, \ldots, \mathbf{f}_n\rangle{:}\mathrm{x} \to \langle \mathbf{f}_1{:}\mathrm{x}, \ldots, \mathbf{f}_n{:}\mathrm{x}\rangle$
$\mathbf{cons}{:}(\mathbf{al}{:}\langle \mathbf{f}, \mathbf{g}\rangle){:}\mathrm{x} \to \mathbf{al}{:}\langle \mathbf{f}{:}\mathrm{x}, \mathbf{cons}{:}\mathbf{g}{:}\mathrm{x}\rangle$
$\mathbf{cons}{:}(\mathbf{ar}{:}\langle \mathbf{f}, \mathbf{g}\rangle){:}\mathrm{x} \to \mathbf{ar}{:}\langle \mathbf{cons}{:}\mathbf{f}{:}\mathrm{x}, \mathbf{g}{:}\mathrm{x}\rangle$
$\mathbf{cond}{:}\langle \mathbf{T}, \mathrm{x}, \mathbf{y}\rangle \to \mathrm{x}$
$\mathbf{cond}{:}\langle \mathbf{F}, \mathrm{x}, \mathbf{y}\rangle \to \mathbf{y}$
$\mathbf{comp}{:}\langle \mathbf{f}, \mathbf{g}\rangle{:}\mathrm{x} \to \mathbf{f}{:}(\mathbf{g}{:}\mathrm{x})$
$\mathbf{apply}{:}\langle \mathbf{f}, \mathrm{x}\rangle \to \mathbf{f}{:}\mathrm{x}$
$\mathbf{K}{:}\mathrm{x}{:}\mathbf{y} \to \mathrm{x}$
$\mathbf{id}{:}\mathrm{x} \to \mathrm{x}$
$\mathbf{eqatom}{:}\langle \mathrm{x}, \mathrm{x}\rangle \to \mathbf{T}$ if x looks like an atom
$\mathbf{eqatom}{:}\langle \mathrm{x}, \mathbf{y}\rangle \to \mathbf{F}$ if both x and $\mathbf{y}$ look like atoms and $\mathrm{x} \neq \mathbf{y}$
$\mathbf{isatom}{:}\mathrm{x} \to \mathbf{T}$ if x looks like an atom

$\mathbf{isatom{:}x} \to \mathbf{F}$ if $\mathbf{x}$ looks like a sequence or function
$\mathbf{isseq{:}x} \to \mathbf{T}$ if $\mathbf{x}$ looks like a sequence
$\mathbf{isseq{:}x} \to \mathbf{F}$ if $\mathbf{x}$ looks like an atom or function
$\mathbf{isfunc{:}x} \to \mathbf{T}$ if $\mathbf{x}$ looks like a function
$\mathbf{isfunc{:}x} \to \mathbf{F}$ if $\mathbf{x}$ looks like an atom or sequence

It is worth observing that the rule $\mathbf{tl}{:}(\mathbf{al}{:}\langle \mathbf{x}, \mathbf{y}_0\rangle)) \to \mathbf{y}_0$ is sound only in the case that $\mathbf{y}_0$ is a sequence. Thus, in order to preserve soundness, we require that $\mathbf{y}_0$ look like a sequence in the rules.

The languages FPA, FPO, and FPAO defined in Section 5 have the extra primitive function symbols **apall**, **or**, **ef**, and **el**. The following rewrite rules characterize these functions:

$\mathbf{apall} : \mathbf{f} : \langle \mathbf{x}_1, \ldots, \mathbf{x}_n\rangle \to \langle \mathbf{f} : \mathbf{x}_1, \ldots, \mathbf{f} : \mathbf{x}_n\rangle$
$\mathbf{apall} : \mathbf{f} : (\mathbf{al} : \langle \mathbf{x}, \mathbf{y}\rangle) \to \mathbf{al} : \langle \mathbf{f} : \mathbf{x}, \mathbf{apall} : \mathbf{f} : \mathbf{y}\rangle$
$\mathbf{apall} : \mathbf{f} : (\mathbf{ar} : \langle \mathbf{x}, \mathbf{y}\rangle) \to \mathbf{ar} : \langle \mathbf{apall} : \mathbf{f} : \mathbf{x}, \mathbf{f} : \mathbf{y}\rangle$
$\mathbf{or} : \langle \mathbf{T}, \mathbf{x}\rangle \to \mathbf{T}$
$\mathbf{or} : \langle \mathbf{x}, \mathbf{T}\rangle \to \mathbf{T}$
$\mathbf{or} : \langle \mathbf{F}, \mathbf{F}\rangle \to \mathbf{F}$
$\mathbf{ef}{:}\langle \mathbf{x}_1, ..., \mathbf{x}_n\rangle \to \mathbf{T},\ n \geq 1$
$\mathbf{ef}{:}(\mathbf{al}{:}\langle \mathbf{x}, \mathbf{y}\rangle) \to \mathbf{T}$
$\mathbf{ef}{:}(\mathbf{ar}{:}\langle \mathbf{al}{:}\langle \mathbf{x}, \mathbf{y}\rangle, \mathbf{z}\rangle) \to \mathbf{T}$
$\mathbf{ef}{:}(\mathbf{ar}{:}\langle \mathbf{ar}{:}\langle \mathbf{x}, \mathbf{y}\rangle, \mathbf{z}\rangle) \to \mathbf{ef}{:}(\mathbf{ar}{:}\langle \mathbf{x}, \mathbf{y}\rangle)$
$\mathbf{el}{:}\langle \mathbf{x}_1, ..., \mathbf{x}_n\rangle \to \mathbf{T},\ n \geq 1$
$\mathbf{el}{:}\langle \mathbf{x}_1, ..., \mathbf{x}_n\rangle \to \mathbf{T},\ n \geq 1$
$\mathbf{el}{:}(\mathbf{ar}{:}\langle \mathbf{x}, \mathbf{y}\rangle) \to \mathbf{T}$
$\mathbf{el}{:}(\mathbf{al}{:}\langle \mathbf{x}, \mathbf{ar}{:}\langle \mathbf{y}, \mathbf{z}\rangle\rangle) \to \mathbf{T}$
$\mathbf{el}{:}(\mathbf{al}{:}\langle \mathbf{x}, \mathbf{al}{:}\langle \mathbf{y}, \mathbf{z}\rangle\rangle) \to \mathbf{el}{:}(\mathbf{al}{:}\langle \mathbf{y}, \mathbf{z}\rangle)$.

Using the techniques of [HWW90], we can show that these rules characterize these primitive functions **apall** and **or**, in that when we combine the rules for **apall** (resp. **or**; **apall**, **or**, **ef**, **el**) with the rewrite rules for FP, we get an analogue of Theorem 2.9 (Adequacy) for FPA (resp. FPO, FPAO).

The language FP1* of Section 5 had the additional function symbol **exists**. For this, we have the following rules:

$\mathbf{exists{:}f} \to \mathbf{or} : \langle \mathbf{f} : \mathbf{t}, \mathbf{exists} : \mathbf{f}\rangle$, where $\mathbf{t}$ is any observable expression
$\mathbf{exists{:}f} \to \mathbf{F}$ if $\mathbf{f}{:}\Omega \to^* \mathbf{F}$.

Note that the second rewrite rule for **exists** is r.e. rather than recursive; the instances of this rule form an r.e. (but not recursive) set, since the problem of deciding if $\mathbf{f}{:}\Omega \to^* \mathbf{F}$ is r.e. We could replace this rule by the recursive rule $\mathbf{exists{:}f} \to \mathbf{cond}{:}\langle \mathbf{f}{:}\Omega, \mathbf{T}, \mathbf{F}\rangle$.

The latter rule is unsound (for example, if $\mu_1(\mathbf{f}) : \perp = \perp$ but $\mu_1(\mathbf{f}) : a = T$, then $\mu_1(\mathbf{exists}{:}\mathbf{f}) = T$ while $\mu_1(\mathbf{cond}{:}\langle \mathbf{f}{:}\Omega, \mathbf{T}, \mathbf{F}\rangle) = \perp$). However, it can only lower the meaning of an expression; adding this rule results in a set of rewrite rules that is *weakly sound*, in that if $\mathbf{x} \to^* \mathbf{y}$ with respect to these rules, then $\mu_1(\mathbf{y}) \leq_I \mu_1(\mathbf{x})$. Either choice of rule (when combined with rewrite rules for FP1, which are identical to those for FP, without the rules for **ar**, **tr**, and **last**) gives an Adequacy Theorem for FP1*. In the latter case, where the rewrite rules are only weakly sound, we have to generalize the notion of strong completeness to get an appropriate Adequacy Theorem: for strong completeness, we now require that for every expression $\mathbf{x}$ there is a transparent expression $\mathbf{y}$ such that $\mathbf{x} \to^* \mathbf{y}$ and $\mu(\mathbf{y}) = \mu(\mathbf{x})$. The requirement that $\mu(\mathbf{y}) = \mu(\mathbf{x})$ is redundant for sound rewrite rules, but not for weakly sound rules. (Notice that without this requirement, by simply adding a rule such as $\mathbf{exists} : \mathbf{f} \to \Omega$, we would get a system that is weakly sound and strongly complete.)

B Proofs of Lemma 3.2, Theorem 3.3, and Lemma 5.2

We recommend that the reader read this section after having read the Sections 3–5, since we develop here simultaneously all the results required to prove Lemma 3.2, Lemma 5.2, and Theorem 3.3. In these proofs, we use the rewrite rules for FPAO given in Appendix A, as well as the fact that the Adequacy Theorem (the analogue of Theorem 2.9) holds for FPAO (with these rewrite rules). As we remarked earlier, this can be proved by a straightforward extension of the techniques of [HWW90]; we omit details here.

Let FPX be the extension of FPAO to include **oneof**, $\Omega_1(\mathbf{x})$, and $\Omega_2(\mathbf{x})$. As usual, we close off under application and sequence formation. FPX has three meaning functions μ_1, μ_2, and μ_3. For expressions $\mathbf{y}$ not involving **oneof**, $\Omega_1(\mathbf{x})$, and $\Omega_2(\mathbf{x})$, $\mu_i(\mathbf{y})$, $i = 1, 2, 3$, agrees with the earlier definitions given. For the new expressions, we have:

$\mu_i(\Omega_i(\mathbf{x})) = \perp$ for $i = 1, 2$
$\mu_i(\Omega_{3-i}(\mathbf{x})) = \mu_i(\mathbf{x})$ for $i = 1, 2$
$\mu_3(\Omega_i(\mathbf{x})) = \perp$ for $i = 1, 2$
$\mu_1(\mathbf{oneof}) = \langle\langle \perp, * \rangle\rangle$
$\mu_2(\mathbf{oneof}) = \langle\langle *, \perp \rangle\rangle$
$\mu_3(\mathbf{oneof}) = \perp$

The key new symbol here is **oneof**. We use the different meanings given to **oneof** by μ_1 and μ_2 to illustrate our point that in FP we cannot distinguish between $\langle\langle \perp, * \rangle\rangle$ and $\langle\langle *, \perp \rangle\rangle$.

Rewrite rules for **apall**, **or**, **ef**, and **el** were given in Appendix A. We have the following additional rules to take care of the new primitive functions in FPX (where i ranges over $1, 2$):

$\mathbf{x} \to \Omega_i(\mathbf{x})$ if $\mu_i(\mathbf{x}) = \perp$

$\Omega_i(\Omega_{3-i}(\mathbf{x})) \to \Omega$
$\Omega_i(\mathbf{C}[\Omega_i(\mathbf{x})]) \to \Omega_i(\mathbf{C}[\mathbf{x}])$
$\mathbf{tl} : (\mathbf{al} : \langle \mathbf{x}, \mathbf{oneof} \rangle) \to \mathbf{oneof}$
$\mathbf{tl} : (\mathbf{ar} : \langle \mathbf{oneof}, \mathbf{y} \rangle) \to \mathbf{ar} : \langle \Omega, \mathbf{y} \rangle$
$\mathbf{tr} : (\mathbf{ar} : \langle \mathbf{oneof}, \mathbf{y} \rangle) \to \mathbf{oneof}$
$\mathbf{tr} : (\mathbf{al} : \langle \mathbf{x}, \mathbf{oneof} \rangle) \to \mathbf{al} : \langle \mathbf{x}, \Omega \rangle$
$\mathbf{null} : \mathbf{oneof} \to \mathbf{F}$
$\mathbf{ef} : \mathbf{oneof} \to \Omega_2(\mathbf{T})$
$\mathbf{el} : \mathbf{oneof} \to \Omega_1(\mathbf{T})$
$\mathbf{cons} : \mathbf{oneof} : \mathbf{y} \to \mathbf{oneof}$
$\mathbf{or} : \langle \Omega_i(\mathbf{T}), \Omega_{3-i}(\mathbf{T}) \rangle \to \mathbf{T}$

Notice that if the language contains $\mathbf{oneof}, \mathbf{ef}$, and $\mathbf{el}$, then it must contain Ω_1, Ω_2 to allow the above rewrite rules.

The next lemma guarantees the soundness of FPX rewriting.

Lemma B.1:

(a) If $\mathbf{x}_0$ *is an FPX expression,* $i \in \{1,2\}$*, and* $\mathbf{x}_0 \to^* \mathbf{x}_1$*, then* $\mu_i(\mathbf{x}_0) = \mu_i(\mathbf{x}_1)$*.*

(b) If $\mathbf{x}_0$ *is an FPX*$-\{\mathbf{oneof}, \mathbf{or}\}$ *expression and* $\mathbf{x}_0 \to^* \mathbf{x}_1$*, then* $\mu_3(\mathbf{x}_0) = \mu_3(\mathbf{x}_1)$*.*

Proof: Each part follows easily by checking all the rewrite rules. The only interesting case is the use of rule $\mathbf{x} \to \Omega_i(\mathbf{x})$ if $\mu_i(\mathbf{x}) = \bot$ in part 2. Notice that in this case, $\mu_3(\mathbf{x}) \leq_I \mu_i(\mathbf{x}) = \bot$. Therefore, it follows that $\mu_3(\mathbf{x}) = \bot = \mu_3(\Omega_i(\mathbf{x}))$. ∎

Note that part (b) of the preceding theorem does not apply to the full language FPX. Some restriction is crucial, since, for example, $\mathbf{null} : \mathbf{oneof} \to \mathbf{F}$ is not sound with respect to μ_3.

To take advantage in FPX of the Adequacy Theorem for FPAO, we need a way of mapping FPX expressions down to FPAO so that rewriting in FPX can be, in a precise sense, *tracked* in FPAO. The relation $\rhd_i$ (for $i \in \{1,2\}$) between FPX and FPAO expressions captures the properties of tracking that we need. The basic idea that the expressions in FPAO that are tracking the expressions in FPX should have the same meaning and the FPAO expressions should not mention $\mathbf{oneof}$, Ω_1, or Ω_2.

Definition B.2: For $i \in \{1,2\}$, define $\mathbf{x} \rhd_i \mathbf{y}$ for FPX expressions $\mathbf{x}$ and FPAO expressions $\mathbf{y}$ by induction on the structure of $\mathbf{y}$ ordered as follows:

- if $\mathbf{y}$ is an atom or primitive function symbol, then $\mathbf{x} \rhd_i \mathbf{y}$ iff $\mathbf{x} = \mathbf{y}$ or $\mathbf{x} = \Omega_{3-i}(\mathbf{y})$
- if $\mathbf{y} = \langle \mathbf{y}_1, \ldots, \mathbf{y}_n \rangle$, then $\mathbf{x} \rhd_i \mathbf{y}$ iff $\mathbf{x} = \mathbf{x}'$ or $\mathbf{x} = \Omega_{3-i}(\mathbf{x}')$, and at least one of the following holds:

- $\mathbf{x}' = \langle \mathbf{x}_1, \ldots, \mathbf{x}_n \rangle$ where $\mathbf{x}_j \ \triangleright_i \mathbf{y}_j$ for $j = 1, \ldots, n$
- $n \geq 1$ and $\mathbf{x}' = \mathbf{al} : \langle \mathbf{x}_1, \mathbf{x}_0 \rangle$ where $\mathbf{x}_1 \ \triangleright_i \mathbf{y}_1$ and $\mathbf{x}_0 \ \triangleright_i \langle \mathbf{y}_2, \ldots, \mathbf{y}_n \rangle$
- $n \geq 1$ and $\mathbf{x}' = \mathbf{ar} : \langle \mathbf{x}_0, \mathbf{x}_n \rangle$ where $\mathbf{x}_0 \ \triangleright_i \langle \mathbf{y}_1, \ldots, \mathbf{y}_{n-1} \rangle$ and $\mathbf{x}_n \ \triangleright_i \mathbf{y}_n$

- if $\mu(\mathbf{y}) = \perp$, then $\mathbf{x} \ \triangleright_i \mathbf{y}$ iff $\mu_i(\mathbf{x}) = \perp$
- if $\mathbf{y} = \mathbf{al} : \langle \mathbf{y}_1, \mathbf{y}_2 \rangle$, then $\mathbf{x} \ \triangleright_i \mathbf{y}$ iff $\mathbf{x} = \mathbf{x}'$ or $\mathbf{x} = \Omega_{3-i}(\mathbf{x}')$, and at least one of the following holds:
  - $\mathbf{x}' = \mathbf{al} : \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ where $\mathbf{x}_j \ \triangleright_i \mathbf{y}_j$ for $j = 1, 2$
  - $\mu(\mathbf{y}_1) = \perp = \mu(\mathbf{y}_2)$ and $i = 1$ and $\mathbf{x}' = \mathbf{oneof}$
- if $\mathbf{y} = \mathbf{ar} : \langle \mathbf{y}_1, \mathbf{y}_2 \rangle$, then $\mathbf{x} \ \triangleright_i \mathbf{y}$ iff $\mathbf{x} = \mathbf{x}'$ or $\mathbf{x} = \Omega_{3-i}(\mathbf{x}')$, and at least one of the following holds:
  - $\mathbf{x}' = \mathbf{ar} : \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ where $\mathbf{x}_j \ \triangleright_i \mathbf{y}_j$ for $j = 1, 2$
  - $\mu(\mathbf{y}_1) = \perp = \mu(\mathbf{y}_2)$ and $i = 2$ and $\mathbf{x}' = \mathbf{oneof}$
- if $\mathbf{y} = \mathbf{y}_1 : \mathbf{y}_2$, then $\mathbf{x} \ \triangleright_i \mathbf{y}$ iff $\mathbf{x} = \mathbf{x}_1 : \mathbf{x}_2$ or $\mathbf{x} = \Omega_{3-i}(\mathbf{x}_1 : \mathbf{x}_2)$, and $\mathbf{x}_j \ \triangleright_i \mathbf{y}_j$ for $j = 1, 2$

∎

Although we do not use this fact here, it easily follows from the definition of $\triangleright_i$ that $\triangleright_i$ is sound, in the sense of the following proposition:

Proposition B.3: *If $i \in \{1, 2\}$ and $\mathbf{x} \ \triangleright_i \mathbf{y}$, then $\mu_i(\mathbf{x}) = \mu(\mathbf{y})$.*

The next lemma guarantees that there are enough rewrite rules in FPX in the sense that if an FPAO expression $\mathbf{y}_0$ can be rewritten to $\mathbf{y}_1$, then an FPX expression $\mathbf{x}_0$ that corresponds to $\mathbf{y}_0$ can be rewritten to an expression $\mathbf{x}_1$ that corresponds to $\mathbf{y}_1$. This will enable us to take advantage of the Adequacy Theorem for FPAO.

Lemma B.4: *If $i \in \{1, 2\}$ and $\mathbf{x}_0 \ \triangleright_i \mathbf{y}_0 \to^* \mathbf{y}_1$ and $\mathbf{x}_0$ is an FPX expression that does not have both* **apall** *and* **oneof** *as subexpressions, then there exists an FPX expression $\mathbf{x}_1$ such that $\mathbf{x}_0 \to^* \mathbf{x}_1 \ \triangleright_i \mathbf{y}_1$.*

Proof: It is easy to reduce the result to the case where $\mathbf{y}_0 \to \mathbf{y}_1$ is an instance of an FPAO rewrite rule. We consider two illustrative cases here, leaving the remainder to the reader.

First suppose $i = 1$, $\mathbf{x}_0 = \mathbf{tl} : (\mathbf{ar} : \langle \mathbf{oneof}, \mathbf{y}_0 \rangle)$, $\mathbf{y}_0 = \mathbf{tl} : (\mathbf{ar} : \langle \mathbf{al} : \langle \Omega, \Omega \rangle, \mathbf{y}_0 \rangle)$, and $\mathbf{y}_1 = \mathbf{ar} : \langle \Omega, \mathbf{y}_0 \rangle$. Since $\mathbf{x}_0 \to \mathbf{y}_1$, we can take $\mathbf{x}_1 = \mathbf{y}_1$ in this case.

Now suppose that $i = 1$, $\mathbf{x}_0 = \mathbf{tr} : (\mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \mathbf{al} : \langle \mathbf{y}_3, \Omega_2(\langle \mathbf{y}_4 \rangle) \rangle \rangle \rangle)$, $\mathbf{y}_0 = \mathbf{tr} : (\langle \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4 \rangle)$, and $\mathbf{y}_1 = \langle \mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3 \rangle$. Then we can rewrite $\mathbf{x}_0$ as follows:

$$
\begin{aligned}
&\mathbf{tr} : (\mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \mathbf{al} : \langle \mathbf{y}_3, \Omega_2(\langle \mathbf{y}_4 \rangle)\rangle\rangle\rangle)\\
&\rightarrow \mathbf{al} : \langle \mathbf{y}_1, \mathbf{tr} : (\mathbf{al} : \langle \mathbf{y}_2, \mathbf{al} : \langle \mathbf{y}_3, \Omega_2(\langle \mathbf{y}_4 \rangle)\rangle\rangle)\rangle\\
&\rightarrow \mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \mathbf{tr} : (\mathbf{al} : \langle \mathbf{y}_3, \Omega_2(\langle \mathbf{y}_4 \rangle)\rangle)\rangle\rangle\\
&\rightarrow \mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \Omega_2(\mathbf{tr} : (\mathbf{al} : \langle \mathbf{y}_3, \Omega_2(\langle \mathbf{y}_4 \rangle)\rangle))\rangle\rangle\\
&\rightarrow \mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \Omega_2(\mathbf{tr} : (\mathbf{al} : \langle \mathbf{y}_3, \langle \mathbf{y}_4 \rangle\rangle))\rangle\rangle\\
&\rightarrow \mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \Omega_2(\mathbf{tr} : \langle \mathbf{y}_3, \mathbf{y}_4 \rangle)\rangle\rangle\\
&\rightarrow \mathbf{al} : \langle \mathbf{y}_1, \mathbf{al} : \langle \mathbf{y}_2, \Omega_2(\langle \mathbf{y}_3 \rangle)\rangle\rangle
\end{aligned}
$$

This last expression serves as the desired $\mathbf{x}_1$. ∎

As the following example shows, the previous lemma would not hold if we allowed $\mathbf{x}_0$ to be an arbitrary FPX expression: $\mathbf{apall} : (\mathbf{K} : \mathbf{T}) : \mathbf{oneof} \rhd_1 \mathbf{apall} : (\mathbf{K} : \mathbf{T}) : (\mathbf{al} : \langle \Omega, \Omega \rangle) \rightarrow \mathbf{al} : \langle \mathbf{K} : \mathbf{T} : \Omega, \mathbf{apall} : (\mathbf{K} : \mathbf{T}) : \Omega \rangle$.

We now have the machinery required to prove Lemma 3.2, Lemma 5.2, and Theorem 3.3. For convenience, we restate these results before proving them.

Lemma 3.2: *There is no expression* $\mathbf{g}$ *in FPO such that* $apallKT \leq_I \mu(\mathbf{g})$.

In fact, we prove the following strengthened form of Lemma 3.2.

Lemma B.5: *There is no expression* $\mathbf{g}$ *in FPO* $\cup\{\mathbf{ef}, \mathbf{el}\}$ *such that* $apallKT \leq_I \mu(\mathbf{g})$.

Proof: Suppose there were an expression $\mathbf{g}$ in FPO $\cup\{\mathbf{ef}, \mathbf{el}\}$ such that $apallKT \leq_I \mu(\mathbf{g})$. Since $\langle\!\langle T \rangle\!\rangle \leq_I \mu(\mathbf{g}) : \langle\!\langle \bot \rangle\!\rangle$, we have that $\mu(\mathbf{g}) : \langle\!\langle \bot \rangle\!\rangle = \langle\!\langle T \rangle\!\rangle$. Since $\langle\!\langle T, * \rangle\!\rangle \leq_I \mu(\mathbf{g}) : \langle\!\langle \bot, * \rangle\!\rangle \leq_I \langle\!\langle T \rangle\!\rangle$, it follows that $\mu(\mathbf{g}) : \langle\!\langle \bot, * \rangle\!\rangle$ must be either $\langle\!\langle T \rangle\!\rangle$ or $\langle\!\langle T, * \rangle\!\rangle$. If $\mu(\mathbf{g}) : \langle\!\langle \bot, * \rangle\!\rangle = \langle\!\langle T \rangle\!\rangle$, then $\mu(\mathbf{g}) : \langle\!\langle \bot, *, \bot \rangle\!\rangle$ would be an upper bound to both $\langle\!\langle T \rangle\!\rangle$ and $\langle\!\langle T, *, T \rangle\!\rangle$ which is impossible. Therefore, $\mu(\mathbf{g}) : \langle\!\langle \bot, * \rangle\!\rangle = \langle\!\langle T, * \rangle\!\rangle$. Similarly, $\mu(\mathbf{g}) : \langle\!\langle *, \bot \rangle\!\rangle = \langle\!\langle *, T \rangle\!\rangle$.

Notice that $\mathbf{g} : \mathbf{oneof} \rhd_1 \mathbf{g} : (\mathbf{al} : \langle \Omega, \Omega \rangle)$. By Adequacy, $\mathbf{g} : (\mathbf{al} : \langle \Omega, \Omega \rangle) \rightarrow^* \mathbf{al} : \langle \mathbf{T}, \mathbf{y}_1 \rangle$ for some FP expression $\mathbf{y}_1$ such that $\mu(\mathbf{y}_1)$ is not a sequence item. By Lemma B.4, there is some FPX expression $\mathbf{x}$ such that $\mathbf{x} \rhd_1 \mathbf{al} : \langle \mathbf{T}, \mathbf{y}_1 \rangle$ and $\mathbf{g} : \mathbf{oneof} \rightarrow^* \mathbf{x}$. Therefore, by Lemma B.1, $\mu_2(\mathbf{x}) = \langle\!\langle *, T \rangle\!\rangle$. Furthermore, $\mathbf{x}$ must have a form such as $\mathbf{al} : \langle \mathbf{T}, \mathbf{x}_1 \rangle$, $\mathbf{al} : \langle \Omega_2(\mathbf{T}), \mathbf{x}_1 \rangle$, $\Omega_2(\mathbf{al} : \langle \mathbf{T}, \mathbf{x}_1 \rangle)$, or $\Omega_2(\mathbf{al} : \langle \Omega_2(\mathbf{T}), \mathbf{x}_1 \rangle)$. It is easy to see that in all such cases, $\mu_2(\mathbf{x}) \neq \langle\!\langle *, T \rangle\!\rangle$. This contradiction yields the result. ∎

Theorem 3.3: *There is no FPO expression whose meaning is ef or el.*

Proof: Since the proofs are similar, we show only that *ef* is not definable. Suppose there were an FPO expression $\mathbf{ef}_0$ such that $\mu(\mathbf{ef}_0) = ef$. Notice that $\mathbf{ef}_0 : \mathbf{oneof} \rhd_1 \mathbf{ef}_0 : (\mathbf{al} : \langle \Omega, \Omega \rangle)$. By Adequacy, $\mathbf{ef}_0 : (\mathbf{al} : \langle \Omega, \Omega \rangle) \rightarrow^* \mathbf{T}$. By Lemma B.4, there is some FPX expression $\mathbf{x}$ such that $\mathbf{x} \rhd_1 \mathbf{T}$ and $\mathbf{ef}_0 : \mathbf{oneof} \rightarrow^* \mathbf{x}$. From Lemma B.1 it follows that $\mu_2(\mathbf{x}) = \bot$. Since $\mathbf{x} \rhd_1 \mathbf{T}$, $\mathbf{x}$ is $\mathbf{T}$ or $\Omega_2(\mathbf{T})$. Since $\mu_2(\mathbf{x}) = \bot$, $\mathbf{x}$ can not be $\mathbf{T}$. Therefore, $\mathbf{ef}_0 : \mathbf{oneof} \rightarrow^* \Omega_2(\mathbf{T})$.

Call an FPX expression *FPO-like* iff it has no occurrences of $\mathbf{ef}$ or $\mathbf{apall}$ and every subexpression of the form $\Omega_2(\mathbf{y})$ has the property that $\mu_2(\mathbf{y}) = \bot$. Notice that the original expression $\mathbf{ef}_0$: **oneof** is FPO-like. It is easy to check that if $\mathbf{x}$ and $\mathbf{y}$ are FPX expressions such that $\mathbf{x} \rightarrow^* \mathbf{y}$, then $\mathbf{y}$ is FPO-like if $\mathbf{x}$ is FPO-like. This gives us a contradiction, since the final expression $\mathbf{ef}_0$: **oneof** $\rightarrow^* \Omega_2(\mathbf{T})$, and $\Omega(\mathbf{T})$ is not FPO-like. This contradiction yields the desired result. ∎

Lemma 5.2: *There is no FPA expression* $\mathbf{g}$ *such that* $or \leq_I \mu(\mathbf{g})$.

Proof: Suppose there were an FPA expression $\mathbf{g}$ such that $or \leq_I \mu(\mathbf{g})$. Notice that $\mathbf{g} : \langle \Omega_1(\mathbf{T}), \Omega_2(\mathbf{T}) \rangle \rhd_1 \mathbf{g} : \langle \Omega, \mathbf{T} \rangle$. By Adequacy, $\mathbf{g} : \langle \Omega, \mathbf{T} \rangle \rightarrow^* \mathbf{T}$. By Lemma B.4, there is some FPX expression $\mathbf{x}$ such that $\mathbf{g} : \langle \Omega_1(\mathbf{T}), \Omega_2(\mathbf{T}) \rangle \rightarrow^* \mathbf{x}$ and $\mathbf{x} \rhd_1 \mathbf{T}$. This implies that $\mathbf{x}$ is either $\mathbf{T}$ or $\Omega_2(\mathbf{T})$. From Lemma B.1 it follows that $\mu_2(\mathbf{x}) = T$ and this implies that $\mathbf{x}$ can not be $\Omega_2(\mathbf{T})$. Thus, $\mathbf{g} : \langle \Omega_1(\mathbf{T}), \Omega_2(\mathbf{T}) \rangle \rightarrow^* \mathbf{T}$. By Lemma B.1, $\mu_3(\mathbf{g} : \langle \Omega_1(\mathbf{T}), \Omega_2(\mathbf{T}) \rangle) = \mu_3(\mathbf{T}) = T$. Therefore it follows that $T = \mu_3(\mathbf{g}) : \mu_3(\langle \Omega_1(\mathbf{T}), \Omega_2(\mathbf{T}) \rangle)$ $= \mu(\mathbf{g}) : \langle\langle \perp, \perp \rangle\rangle \leq_I \mu(\mathbf{g}) : \langle\langle F, F \rangle\rangle = F$. This contradiction gives the desired result. ∎

C Proofs of Theorems 6.1, 6.5, 7.1, and 7.4, and rewrite rules for FPE and FPE*

C.1 Preliminaries

Our proofs that FPO1 is f.e. complete with respect to $D1^*$ and that FP1* is expressively complete with respect to $D1^*$ follows roughly the same lines as Plotkin's proof that LCF enriched with a parallel conditional is f.e. complete and that LCF enriched with an existential operator is expressively complete [Pl77]. However, there are greater complexities in the context of FP, since we are working in an untyped domain, and so cannot proceed by induction on types. Instead, we adopt an approach similar to Wadsworth [Wa78] and work with the finite elements in $D1^*$, and use induction on the *complexity* of these finite elements. In order to do so, we need to define an appropriate notion of complexity. We also need a way of representing the finite elements; in particular, we need a representation of the function items.

We proceed as follows. We simultaneously define a set *Rep* of *representations* of finite elements in D^*, a surjective function *emb* from *Rep* to the set of finite elements of D^*, and an injective function *code* from *Rep* to observable items. Since the image of the function *emb* is the set of finite elements of D^*, we can define an injective function *rep* from the finite elements of D^* to *Rep* such that $emb(rep(d)) = d$ for all finite elements of D^*. We call $rep(d)$ the *canonical representation* of d. The actual choice of canonical representation is not particularly significant, since throughout we work with all representations, not just the canonical representation; for definiteness, we choose $rep(d)$ to be the representation d' such that $enc(code(d'))$ is minimal in the set $\{enc(code(d'')) | emb(d'') = d\}$. (Recall that *enc* is the mapping from observable items to integers defined in Section 4.) Thus, we can view *code* as a function from finite items in D^* to observable items by identifying $code(d)$ with $code(rep(d))$ for each finite element d in D^*.

The set *Rep* is defined much like the set D. That is, we define a sequence of sets $\mathit{Rep}_0, \mathit{Rep}_1, \ldots$. For each element $r \in \mathit{Rep}_n$, we associate a (unique) finite element $\mathit{emb}(r) \in D^*$. This will enable us to compare elements in Rep_n using the ordering $\leq_I$. We take $\mathit{Rep}_0 = D_0$; thus, Rep_0 consists of all the atomic items and $\perp$. Suppose we have constructed $\mathit{Rep}_0, \ldots, \mathit{Rep}_n$. Rep_{n+1} is the result of starting with Rep_n and closing off under sequence formation and function formation. It is easy to define what it means to be closed under sequence formation: we require that if $\{x_1, \ldots, x_k, y_1, \ldots, y_\ell, z_1, \ldots, z_m\} \subseteq \mathit{Rep}_n$, $k \leq 2n$, and $\max(\ell, m) \leq n$, then $\langle\langle x_1, ..., x_k\rangle\rangle$ and $\langle\langle y_1, \ldots, y_\ell, *, z_1, \ldots, z_m\rangle\rangle$ are in Rep_{n+1}. We call such representations of this form *sequence representations*; as in the case of sequence items, if a sequence representation has no $*$, we say that it has definite length. The definition of *emb* on these sequence representations should be obvious. We now define what it means to be closed under function formation. Suppose that $x_1, \ldots, x_k, y_1, \ldots, y_k \in \mathit{Rep}_n$, $y_j \neq \perp$ for $j = 1, \ldots, k$, and for each $x \in D^*$, the set $\{\mathit{emb}(y_i) : \mathit{emb}(x_i) \leq_I x\}$ has a least upper bound. (Note that the least upper bound is finite if it exists.) In that case, $[x_1 \to y_1, \ldots, x_n \to y_n] \in \mathit{Rep}_{n+1}$. We call a representation of this form a *function representation.* If $g = [x_1 \to y_1, \ldots, x_n \to y_n]$, we define $\mathit{dom}(g) = \{x_1, \ldots, x_n\}$ and $\mathit{range}(g) = \{y_1, \ldots, y_n\}$, and denote y_i as $g(x_i)$. Moreover, we define $\mathit{emb}([x_1 \to y_1, \ldots, x_n \to y_n])$ to be the function $f \in D^*$ such that

$$f : z = \sqcup\{\mathit{emb}(y_i) : \mathit{emb}(x_i) \leq z, i = 1, \ldots, k\}.$$

We leave it to the reader to check that f is indeed finite. This completes the definition of *Rep*. It should be clear that the function *emb* maps *Rep* onto the finite elements in D^*. Indeed, as a consequence of the close similarity of Rep_n and D_n, it is easy to see that $\mathit{emb}(\mathit{Rep}_n)$ is the same as the image of the embedding of D_n into D^*. Using *emb*, we can define an ordering $\leq_r$ on *Rep*: for representations $x, y \in \mathit{Rep}$, we define $x \leq_r y$ iff $\mathit{emb}(x) \leq_I \mathit{emb}(y)$.

We can similarly define a subset *Rep1* of *Rep* that consists of the representations of finite elements in $D1^*$, simply by restricting sequence representations so that the $*$ can only occur at the right-hand end, and modifying the definition of function representation to use $D1^*$ instead of D^*. Clearly the range of $\mathit{emb}(\mathit{Rep1})$ consist of the finite elements of $D1^*$.

We now define *code* on *Rep* as follows:

$\mathit{code}(\perp) = \langle\langle\ \rangle\rangle$,

$\mathit{code}(a) = a$ for the atom a,

$\mathit{code}(\langle\langle x_1, ..., x_n\rangle\rangle) = \langle\langle F, \mathit{code}(x_1), ..., \mathit{code}(x_n)\rangle\rangle$ (where we take $\mathit{code}(*) = \langle\langle\langle\langle\ \rangle\rangle\rangle\rangle$)

$\mathit{code}([x_1 \to y_1, ..., x_n \to y_n]) = \langle\langle T, \mathit{code}(x_1), \mathit{code}(y_1), ..., \mathit{code}(x_n), \mathit{code}(y_n)\rangle\rangle$,

As we remarked earlier, we can now view *code* as a function on the finite elements of D^* by identifying $\mathit{code}(d)$ with $\mathit{code}(\mathit{rep}(d))$ for a finite element d in D^*. We shall make this identification below without further comment. This definition of *code* clearly satisfies the minimal requirements mentioned in Section 4. In particular, we can easily write FP1 functions that test whether an observable item encodes a representation for bottom, for an atomic item, for a finite sequence item of definite length, for a finite sequence item of indefinite length, or for a function item.

To carry out the proof, we need to introduce some technical machinery. We first define two notions of *complexity* on *Rep*, denoted $||x||_1$ and $||x||_2$ respectively. The first notion counts the number of arrows in each term in a function representation. the second essentially measures the length of the representation. These notions of complexity will allow us to prove some results by induction on complexity.

We define $||x||_1$ and $||x||_2$ by induction on the structure of x. For $||x||_1$ we have

- $||x||_1 = 0$ if x is $\perp$, $\langle\!\langle\,\rangle\!\rangle$, $*$, or an atom
- $||\langle\!\langle x_1, \ldots, x_n \rangle\!\rangle||_1 = \max(||x_1||_1, \ldots, ||x_n||_1)$
- $||[x_1 \rightarrow y_1, \ldots, x_n \rightarrow y_n]|| = 1 + \max(||x_1||_1, ||y_1||_1, \ldots, ||x_n||_1, ||y_n||_1)$.

For $||x||_2$ we have

- $||x||_2 = 1$ if x is $\perp$, $*$, or an atom
- $||\langle\!\langle x_1, \ldots, x_n \rangle\!\rangle||_2 = 1 + ||x_1||_2 + \ldots + ||x_n||_2$
- $||[x_1 \rightarrow y_1, \ldots, x_n \rightarrow y_n]||_2 = 1 + ||x_1||_2 \cdot 2^{||y_1||_2} + \ldots + ||x_n||_2 \cdot 2^{||y_n||_2}$.

Finally, we define $||x|| = (||x||_1, ||x||_2)$, and take complexity to be lexicographically ordered, so that $||x|| < ||y||$ if either (1) $||x||_1 < ||y||_1$ or (2) $||x||_1 = ||y||_1$ and $||x||_2 < ||y||_2$.

It is easy to check that a sequence representation or a function representation has greater complexity than any of its components. Besides this property, our definition of complexity has two other key properties we shall need in our proofs.

Proposition C.1: *If S is a finite set of representations in Rep1 such that the elements of S are pairwise compatible, then we can effectively find a representation that we denote $\sqcup S$ such that (as the notation suggests) $\sqcup S$ is a least upper bound of S (with respect to $\leq_r$), and $|| \sqcup S||_1 = \max_{t \in S}(||t||_1)$.*

Proof: We proceed by induction on the structure of complexity of the representation in S of maximum complexity. We assume without loss of generality that $\perp \notin S$ (since y is a least upper bound for $(S - \{\perp\})$ iff y is a least upper bound for S). Since the elements of S are pairwise compatible, S either consists of a single atom, a set of sequence representations, or a set of function representations. Clearly if S consists of a single atom, the result is immediate, so we focus on the other cases here.

If S consists of sequence representations, then we can divide up S into two subsets S_1 and S_2, where S_1 consists of the representations in S of definite length, and S_2 consists of the remaining representations. The sequence representations in S_1 must all have the same definite length (otherwise the representations in S would not be pairwise compatible). In addition, if S_1 is nonempty, the number of non-$*$ elements in each sequence representation in S_2 must be no more than the length of the sequence representations in S_1. (That is, if the length of each sequence representation in S_1 is m, then the sequence representations in S_2 have the form $\langle\!\langle y_1, \ldots, y_k, *\rangle\!\rangle$, where $k \leq m$. Notice that the fact that the $*$ is on the right follows from our assumption that $S \subset$ *Rep1*.) There are now two cases to consider: $S_1 \neq \emptyset$ and $S_1 = \emptyset$. If $S_1 \neq \emptyset$, suppose the sequence representations in S_1 all have definite length m. Replace each sequence representation $\langle\!\langle y_1, \ldots, y_k, *\rangle\!\rangle$ in S_2 by the representation $\langle\!\langle y_1, \ldots, y_k, \perp, \ldots, \perp\rangle\!\rangle$, where there are $m-k$ $\perp$'s. Let S_2' be the set that results after this replacement, and let $S' = S_1 \cup S_2'$. Suppose

$$S' = \{\langle\!\langle x_1^1, \ldots, x_m^1\rangle\!\rangle, \ldots, \langle\!\langle x_1^n, \ldots, x_m^n\rangle\!\rangle\}.$$

Define

$$y = \langle\!\langle \sqcup\{x_1^1, \ldots, x_1^n\}, \ldots, \sqcup\{x_m^1, \ldots, x_m^n\}\rangle\!\rangle.$$

The inductive hypothesis guarantees us that y is well defined. We leave it to the reader to check that y is a least upper bound for both S' and S, and that $||y||_1 = \max_{t \in S} ||t||_1$. Thus, we can take $y = \sqcup S$.

If $S_1 = \emptyset$, let m be the number of non-$*$ elements in the longest representation in S_2. Replace each representation $\langle\!\langle y_1, \ldots, y_k, *\rangle\!\rangle$ in S_2 by the representation $\langle\!\langle y_1, \ldots, y_k, \perp, \ldots, \perp, *\rangle\!\rangle$, with $m-k$ $\perp$'s. Let S' be the set that results after this replacement. Suppose

$$S' = \{\langle\!\langle x_1^1, \ldots, x_m^1, *\rangle\!\rangle, \ldots, \langle\!\langle x_1^n, \ldots, x_m^n, *\rangle\!\rangle\}.$$

Define

$$y = \langle\!\langle \sqcup\{x_1^1, \ldots, x_1^n\}, \ldots, \sqcup\{x_m^1, \ldots, x_m^n\}, *\rangle\!\rangle.$$

Again, the inductive hypothesis guarantees that y is well defined. We leave it to the reader to check that y is a least upper bound for both S' and S, and that $||y||_1 = \max_{t \in S} ||t||_1$. Thus, we can take $y = \sqcup S$.

Finally, suppose that S consists of function representations. If $S = \{f_1, \ldots, f_n\}$, where $f_i = [x_1^i \to y_1^i, \ldots, x_{m_i}^i \to y_{m_i}^i]$, let

$$f = [x_1^1 \to y_1^1, \ldots, x_{m_1}^1 \to y_{m_1}^1, \ldots, x_1^n \to y_1^n, \ldots, x_{m_n}^n \to y_{m_n}^n].$$

We claim that f is a function representation and f is a least upper bound for S, so we can take $f = \sqcup S$. To check that f is a function representation, suppose $z \in D1^*$, and let $Y = \{y_j^i : emb(x_j^i) \leq_I z\}$. We must show that $emb(Y)$ has a least upper bound. It clearly suffices to show that Y has a least upper bound, since if y' is a least upper bound of Y (with respect to $\leq_r$), then $emb(y')$ is a least upper bound of $emb(Y)$. It suffices by the induction hypothesis to show that the items in Y are pairwise compatible. Suppose

$y^i_j, y^{i'}_{j'} \in Y$, and let $y = emb(y^i_j)$, $y' = emb(y^{i'}_{j'})$. We want to show that y^i_j and $y^{i'}_{j'}$ are compatible. Clearly y^i_j and $y^{i'}_{j'}$ are compatible iff y and y' are compatible. If $i = i'$, then it follows from the fact that f_i is a function representation that y and y' are compatible. If $i \neq i'$, let g be an upper bound of $f_i, f_{i'}$ (which exists since the elements of S are assumed to be pairwise compatible). Clearly $emb(g) : z$ is an upper bound of y and y'. Thus, f is indeed a function representation.

It is easy to see that f is an upper bound of S. To see that it is a least upper bound, it suffices to show that $emb(f)$ is a least upper bound for $emb(S)$. So let f' be an upper bound of $emb(S)$ and let x be an arbitrary item in $D1^*$. We want to show that $emb(f) : x \leq_I f' : x$. Let $X_i = \{x' \in dom(f_i) | emb(x') \leq_I x\}$ and let $X = \{x' \in dom(f) | emb(x') \leq_I x\}$. By construction, $X = X_1 \cup \ldots \cup X_n$. Let $Y_i = \{f_i(x') : x' \in X_i\}$ and let $Y = \{f(x') : x' \in X\}$. Our arguments in the previous paragraph show that Y is compatible; it follows from the induction hypothesis that $\sqcup Y$ is well defined. Similarly, we can show that $\sqcup Y_i$ is well defined. Suppose $z_i = \sqcup Y_i$ and $z = \sqcup Y$. Since $Y = Y_1 \cup \ldots \cup Y_n$, it is easy to show that $emb(z) = \sqcup\{emb(z_1), \ldots, emb(z_n)\}$. Since $emb(f_i) \leq_I f'$ for all i, we must have $emb(z_i) \leq_I f' : x$ for all i. Since $emb(f) : x = emb(z)$ by definition, it follows that $emb(f) : x \leq_I f' : x$, as desired. Finally, it is easy to see from the definition of f that $||f||_1 = \max\{||f_1||_1, \ldots, ||f_n||_1\}$. ∎

Corollary C.2: *If f is a finite function representation and $d \in D1^*$, then $\sqcup\{f(x) : x \in dom(f), emb(x) \leq_I d\}$ represents $emb(f) : d$, and $|| \sqcup \{f(x) : x \in dom(f), emb(x) \leq_I d\}||_1 < ||f||_1$.*

Proof: Suppose $f = [x_1 \to y_1, \ldots, x_n \to y_n]$. Given $d \in D1^*$, our definitions guarantee that $\sqcup\{y_i | emb(x_i) \leq_I d\}$ indeed represents $emb(f) : d$. By Lemma C.1, it follows that $||\{y_i | emb(x_i) \leq_I d\}||_1 \leq \max\{||y||_1 | y \in range(f)\} < ||f||_1$. ∎

We can also prove the following generalization of the fact that $D1^*$ is consistently complete.

Corollary C.3: *Let S be a pairwise compatible subset of $D1^*$. Then S has a least upper bound in $D1^*$.*

Proof: Let S_1 be the set of finite elements of $D1^*$ that are $\leq_I$ at least one element of S. Since $D1^*$ is algebraic, it suffices to show that S_1 has a least upper bound (which will also be the least upper bound of S). Since S is pairwise compatible, S_1 is also pairwise compatible. Let $S_2 = \{emb(\sqcup W) | W \subseteq rep(S_1), W \text{ is finite}\}$. (Note that by Proposition C.1, $\sqcup W$ is well-defined for each such set W.) It is clear that S_2 is a directed set and hence, by the completeness of $D1^*$, has a least upper bound y. It easily follows that y is the least upper bound of S_1 as desired. ∎

Corollary C.4: *$D1^*$ is consistently complete.*

The second property of complexity (which is the reason for the perhaps unexpectedly complicated definition of $|| \ ||_2$ in the case of function representations) is that the uncurried form of a finite function representation (that is, $\lambda x, y.f(x, y)$) has lower complexity than the curried form (that is, $\lambda x.\lambda y.f'(x)(y)$, where $f'(x)(y) = f(x, y)$). More precisely, given a finite function representation of the form

$$f = [x_1 \rightarrow [y_{11} \rightarrow z_{11}, \ldots, y_{1m_1} \rightarrow z_{1m_1}], \ldots, x_n \rightarrow [y_{n1} \rightarrow z_{n1}, \ldots, y_{nm_n} \rightarrow z_{nm_n}]],$$

let $uncurry(f)$ be the function representation

$$[\langle\!\langle x_1, y_{11} \rangle\!\rangle \rightarrow z_{11}, \ldots, \langle\!\langle x_1, y_{1m_1} \rangle\!\rangle \rightarrow z_{1m_1}, \ldots, \langle\!\langle x_n, y_{n1} \rangle\!\rangle \rightarrow z_{n1}, \ldots, \langle\!\langle x_n, y_{nm_n} \rangle\!\rangle \rightarrow z_{nm_n}].$$

Lemma C.5: *Taking f as above, we have $||uncurry(f)|| < ||f||$.*

Proof: It is easy to check that $||uncurry(f)||_1 \leq ||f||_1$. Thus, it suffices to show that $||uncurry(f)||_2 < ||f||_2$. This will follow if we let $a_i = ||x_i||_2$, $b_{ij} = ||y_{ij}||_2$, and $c_{ij} = ||z_{ij}||_2$, and show that

$$1 + \sum_{i=1}^{n} \sum_{j=1}^{m_i} (1 + a_i + b_{ij}) \cdot 2^{c_{ij}} \quad < \quad 1 + \sum_{i=1}^{n} a_i \cdot 2^{(1 + \sum_{j=1}^{m_i} b_{ij} \cdot 2^{c_{ij}})} \ .$$

It clearly suffices to show that for each i, we have

$$\sum_{j=1}^{m_i} (1 + a_i + b_{ij}) \cdot 2^{c_{ij}} < a_i \cdot 2^{(1 + \sum_{j=1}^{m_i} b_{ij} \cdot 2^{c_{ij}})}.$$

Observe that the right-hand side is equal to $2 \cdot a_i \cdot 2^{b_{i1} \cdot 2^{c_{i1}}} \cdots 2^{b_{im_i} \cdot 2^{c_{im_i}}}$. Since in general we have $t_1 \cdots t_n \geq t_1 + \cdots + t_n$ as long as each $t_i \geq 2$, it follows that the right-hand side is at least $2 \cdot a_i \cdot (2^{b_{i1} \cdot 2^{cz_{i1}}} + \ldots + 2^{b_{im_i} \cdot 2^{c_{im_i}}})$. Thus, it suffices to prove that for each i, j, we have

$$(1 + a_i + b_{ij}) \cdot 2^{c_{ij}} < 2 \cdot a_i \cdot 2^{b_{ij} \cdot 2^{c_{ij}}}.$$

This latter inequality follows from some straightforward algebraic manipulations.

$$\begin{array}{lll}
 & (1 + a_i + b_{ij}) \cdot 2^{c_{ij}} & \\
\leq & 3 \cdot a_i \cdot b_{ij} \cdot 2^{c_{ij}} & \text{(since, in general, } 1 + t_1 + t_2 \leq 3 \cdot t_1 \cdot t_2\text{)} \\
< & 2 \cdot a_i \cdot 2^{b_{ij} \cdot 2^{c_{ij}}} & \text{(since, in general, } 3 \cdot t < 2 \cdot 2^t\text{).}
\end{array}$$

This completes the proof. ∎

C.2 Proof of Theorem 6.1

Recall that we are trying to find an FPO1 expression $\mathbf{E}_1$ such that $\mu_1(\mathbf{E}_1) : code(z) = z$ for all finite elements z in $D1^*$. Recall that for a finite item z, we identify $code(z)$ with $code(rep(z))$, where $rep(z)$ is the canonical representation of z. We will in fact define an expression $\mathbf{E}_1$ that gives us the desired result for all representations, not just canonical representations. That is, we define $\mathbf{E}_1$ so that

- $\mu_1(\mathbf{E}_1) : code(z) = emb(z)$ for all $z \in Rep1$.

This is an example of a typical phenomenon in all our proofs below: we work with representations, rather than finite elements.

In order to define $\mathbf{E}_1$, we will need two auxiliary expressions, $\mathbf{B}_1$ and $\mathbf{split}_1$. Roughly speaking, given $code(z_1)$ for some $z_1 \in Rep1$ and an item $y \in D1^*$, we would like $\mu_1(\mathbf{B}_1)$ to return T if $emb(z_1) \leq_I y$ and F if $emb(z_1)$ and y are incompatible. Unfortunately, we cannot define an expression with these properties in FPO1. (As we shall see later, we can define such an expression if we enrich FPO1 slightly.) We can, however, define a "bounded" version of it, which still returns returns true if $emb(z_1) \leq_I y$ and returns false if it is given a finite witness z_2 to the incompatibility of z_1 and y. More formally, we require that for $z_1, z_2 \in Rep1$, $y \in D1^*$, we have:

- $\mu_1(\mathbf{B}_1) : \langle\langle code(z_1), code(z_2), y\rangle\rangle = T$ iff $emb(z_1) \leq_I y$
- if $emb(z_2) \leq_I y$ and z_1, z_2 have no upper bound, then $\mu_1(\mathbf{B}_1) : \langle\langle code(z_1), code(z_2), y\rangle\rangle = F$.

Notice that if $emb(z_2) \leq_I y$ and z_1 and z_2 are incompatible then it must be the case that $emb(z_1)$ and y are incompatible as well. Also notice the first condition is an "if and only if" condition, since in function application we need to know exactly when one item is greater than another, whereas the second condition is merely an "if" condition, since it suffices to know when the items are incompatible.

The expression $\mathbf{split}_1$ is somewhat similar to $\mathbf{B}_1$, but applies to function representations. We say that x is a *minimal element* in $D1^*$ if $y <_I x$ implies that $y = \perp$. We say that x is a *minimal representation* in *Rep1* if $emb(x)$ is a minimal element in $D1^*$. It is easy to check that the minimal representations are precisely the atoms, $\langle\langle\,\rangle\rangle$, $\langle\langle \perp, *\rangle\rangle$, and $[\,]$, the function representation that, intuitively, maps all elements to $\perp$. Notice that each minimal elements is finite, and has a unique representations.

If f is a function representation, x is a minimal representation, and y is an arbitrary element in $D1^*$, then we require:

- $\mu_1(\mathbf{split}_1) : \langle\langle code(f), code(x), y\rangle\rangle = T$ iff $emb(x) \leq_I emb(f) : y$
- if $emb(f) : y$ and $emb(x)$ are incompatible, then $\mu_1(\mathbf{split}_1) : \langle\langle code(f), code(x), y\rangle\rangle = F$.

As we shall see, the requirement that x be minimal is crucial here. As before, the first condition is "if and only if" whereas the second condition is merely an "if" condition.

The definitions of $\mathbf{E}_1$, $\mathbf{B}_1$, and $\mathbf{split}_1$ all involve nontrivial programming in FPO1. They are given in terms of mutual recursion (so that, for example, the definition of $\mathbf{split}_1$ involves $\mathbf{B}_1$, and so on). We describe the functions using "psuedo-FPO1", which is intended to convey the essential ideas of the definition without going into programming details. We first define $\mathbf{split}_1$, then $\mathbf{E}_1$, and finally $\mathbf{B}_1$. When proving correctness of our definitions, we proceed by induction on the complexity of the representations involved, assuming the correctness of all three functions for arguments of lower complexity, and for previously defined functions for arguments of the same complexity (so that, when proving correctness of $\mathbf{B}_1$, we assume the correctness of $\mathbf{split}_1$ and $\mathbf{E}_1$ for arguments of the same complexity).

We define $\mathbf{split}_1$ as follows:

$$
\begin{array}{ll}
\lambda\mathbf{z}.(& \textbf{if } \mathbf{z} = \langle \mathbf{code}(f), \mathbf{code}(x), \mathbf{y}\rangle \text{ where } f \text{ is a function representation and} \\
& \quad x \text{ is a minimal representation} \\
& \textbf{then if } \{t \in \mathit{dom}(f) | x \not\leq_r f(t)\} = \emptyset \\
& \quad \textbf{then } \bigvee_{\{t\in \mathit{dom}(f)\}} \mathbf{B}_1 : \langle \mathbf{code}(t), \mathbf{code}(\perp), \mathbf{y}\rangle \\
& \quad \textbf{else } \bigvee_{\{t\in \mathit{dom}(f) | x\leq_r f(t)\}} \bigwedge_{\{t'\in \mathit{dom}(f) | x\not\leq_r f(t')\}} \mathbf{B}_1 : \langle \mathbf{code}(t), \mathbf{code}(t'), \mathbf{y}\rangle \\
& \textbf{else } \Omega)
\end{array}
$$

A few comments are in order regarding this definition. Suppose f, the first argument to $\mathbf{split}_1$, is of the form $[t_1 \to u_1, \ldots, t_k \to u_k]$. From $\mathit{code}(f)$, we can easily compute codes for each element $t_i \in \mathit{dom}(f)$ and the corresponding element u_i in the range. Since we are given the code of x and can easily compute the code of $f(t_i)$ (it is just the code of the corresponding u_i), we can easily write an FPO1 expression that, given $\mathit{code}(x)$ and $\mathit{code}(f(t_i))$, checks if $x \leq_r f(t_i)$. Thus, it is straightforward programming to write an FPO1 expression that, given $\mathbf{code}(f)$, returns $\mathbf{T}$ if $\{t \in \mathit{dom}(f) | x \not\leq_r f(t)\} = \emptyset$ and $\mathbf{F}$ otherwise.

The FPO1 expression that represents $\bigvee_{\{t\in \mathit{dom}(f)\}} \mathbf{B}_1 : \langle \mathbf{code}(t), \mathbf{code}(\perp), \mathbf{y}\rangle$ uses the **or** and an easily definable dual **and**. Strictly speaking, we actually need to extend **or** to a function **orr** that works on sequences (and not just pairs). We can do this as follows:

$$\mathbf{orr} = \lambda\mathbf{x}.(\mathbf{cond} : \langle \mathbf{null} : \mathbf{x}, \mathbf{F}, \mathbf{or} : \langle \mathbf{first} : \mathbf{x}, \mathbf{orr} : (\mathbf{tl} : \mathbf{x})\rangle\rangle).$$

We can then define **and** (that also works on sequences) by first defining a Boolean negation $\mathbf{not} = \lambda\mathbf{x}.(\mathbf{cond}\langle \mathbf{x}, \mathbf{F}, \mathbf{T}\rangle)$, and then taking $\mathbf{and} = \lambda\mathbf{x}.(\mathbf{not} : \mathbf{orr} : (\mathbf{apall}1 : \mathbf{not} : \mathbf{x})))$. Using these expressions, it is now straightforward to write an FPO1 expression that captures the informal description of $\mathbf{split}_1$ above; we leave details to the reader.

As mentioned above, the proof that $\mathbf{split}_1$ has the required properties proceeds by induction on complexity. Consider $\mu_1(\mathbf{split}_1) : \langle\langle \mathit{code}(f), \mathit{code}(x), y\rangle\rangle$, where f is a finite function representation with $||f|| = (k, k')$ and x is a minimal representation. It is easy to see that if $\mu_1(\mathbf{split}_1) : \langle\langle \mathit{code}(f), \mathit{code}(x), y\rangle\rangle = T$, then there exists a q such that $\mathbf{B}_1$:

$\langle \mathbf{code}(t), \mathbf{code}(q), \mathbf{y}\rangle = \mathbf{T}$ for some $t \in dom(f)$ with $x \leq_r f(t)$. Hence, by the induction hypothesis, we have that $emb(x) \leq_I emb(f) : y$. In proving the other properties, we can assume inductively that the definition of $\mathbf{B}_1$ is correct for representations z_1, z_2 such that $\max(||z_1||, ||z_2||) < (k, k')$. Because x is minimal, if $emb(x) \leq_I emb(f) : y$, then there is some $t_i \in dom(f)$ such that $x \leq_r f(t_i)$ and $emb(t_i) \leq_I y$. (We remark that this would not be true in general if x were not minimal.) The definition of complexity guarantees that $||t_i|| < (k, k')$. There are now two cases to consider. If $\{t \in dom(f) | x \not\leq_r f(t)\} = \emptyset$, so that $x \leq_r f(t)$ for all $t \in dom(f)$, then it is easy to see, using the induction hypothesis, that $\mathbf{B}_1 : \langle \mathbf{code}(t_i), \mathbf{code}(\perp), \mathbf{y}\rangle = \mathbf{T}$, giving us $\mu_1(\mathbf{split}_1) : \langle\langle code(f), code(x), y\rangle\rangle = T$, as desired. If it is not the case that $x \leq_r f(t)$ for all $t \in dom(f)$, then for all $t' \in dom(f)$ such that $x \not\leq_r f(t')$ we must have $\mathbf{B}_1 : \langle \mathbf{code}(t_i), \mathbf{code}(t'), \mathbf{y}\rangle = \mathbf{T}$. Again, this gives us the desired result. If $emb(f) : y$ and $emb(x)$ have no upper bound, then there must be some $t'' \in dom(f)$ such that $x \not\leq_r f(t'')$ and $emb(t'') \leq_I y$. It is easy to see that if $t \in dom(f)$ and $x \leq_r f(t)$, then t and t'' have no upper bound (for otherwise f would not be a well-defined function representation). By the inductive assumption it follows $\mathbf{B}_1 : \langle \mathbf{code}(t), \mathbf{code}(t''), \mathbf{y}\rangle = \mathbf{F}$ for all $t \in dom(f)$ such that $x \leq_r f(t)$, and hence $\bigvee_{\{t \in dom(f) | x \leq_r f(t)\}} \bigwedge_{\{t' \in dom(f) | x \not\leq_r f(t')\}} \mathbf{B}_1 : \langle \mathbf{code}(t), \mathbf{code}(t'), \mathbf{y}\rangle = \mathbf{F}$. This completes the inductive argument for the correctness of $\mathbf{split}_1$.

The definition of $\mathbf{E}_1 : \mathbf{code}(z_1)$ proceeds as expected by induction on the structure of the representation z_1. For example, if z_1 is a nonempty sequence representation (which can be easily checked by looking at $\mathbf{code}(z_1)$), let $\mathit{first}(z_1)$ and $\mathit{tl}(z_1)$ be the obvious representations of the first element of z_1 and the tail of z_1. (We shall similarly abuse notation in viewing other function items in $D1^*$ as functions on representations.) Then $\mathbf{E}_1 : \mathbf{code}(z_1) = \mathbf{al} : \langle \mathbf{E}_1 : (\mathbf{codefirst} : \mathbf{code}(z_1)), \mathbf{E}_1 : (\mathbf{codetl} : \mathbf{code}(z_1))\rangle$, where **codefirst** and **codetl** are (easily definable) FPO1 expressions such that $\mathbf{codefirst} : \mathbf{code}(z_1) = \mathbf{code}(\mathit{first}(z_1))$ and $\mathbf{codetl} : \mathbf{code}(z_1) = \mathbf{code}(\mathit{tl}(z_1))$. Since our definition of complexity guarantees that both $\mathit{first} : z_1$ and $\mathit{tl} : z_1$ have lower complexity than z_1, it is easy to see inductively that this definition has the right properties.

The hard part of the definition of $\mathbf{E}_1$ is, as expected, the case of function items. If z_1 is a function item, then $\mathbf{E}_1 : \mathbf{code}(z_1)$ has the form $\lambda \mathbf{v}.(\ldots)$. The argument in the $(\ldots)$ should be the result of applying $\mathbf{E}_1 : \mathbf{code}(z_1)$ to $\mathbf{v}$. In order to compute this, we use the $\mathbf{split}_1$ function defined above. For example, if $\mathbf{split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}(a), \mathbf{v}\rangle = \mathbf{T}$, then we know that $a \leq_I emb(z_1) : \mu_1(\mathbf{v})$. This means that we must in fact have $a = emb(z_1) : \mu_1(\mathbf{v})$, so we can return $\mathbf{a}$. Similarly, if $\mathbf{split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}(\langle\langle \perp, *\rangle\rangle), \mathbf{v}\rangle = \mathbf{T}$, then we know that $emb(z_1) : \mu_1(\mathbf{v})$ must be a nonempty sequence item. If f is the function representation $[x_1 \to y_1, \ldots, x_k \to y_k]$ and g is a function on representations, then we take $\lambda y.(g : (f : y))$ to be the function representation that results by starting with $[x_1 \to g(y_1), \ldots, x_k \to g(y_k)]$ and removing all components such that $g(y_1) = \perp$. Notice that $emb(\lambda y.(\mathit{first} : (z_1 : y))) : \mu_1(\mathbf{v})$ gives us the first element of the sequence item $emb(z_1) : \mu_1(\mathbf{v})$, while $emb(\lambda y.(\mathit{tl} : (z_1 : y))) : \mu_1(\mathbf{v})$ gives us its tail. Let **codefirstf** and **codetlf** be FPO1 expressions such that

$$\mathbf{codefirstf} : \mathbf{code}(z_1) = \mathbf{code}(\lambda y.(\mathit{first} : (z_1 : y))) \text{ and}$$
$$\mathbf{codetlf} : \mathbf{code}(z_1) = \mathbf{code}(\lambda y.(\mathit{tl} : (z_1 : y))).$$

Thus, if $z_1 : \mu_1(\mathbf{v})$ is a nonempty sequence item, then $\mathbf{E}_1 : \mathbf{code}(z_1) : \mathbf{v} = \mathbf{al} : \langle \mathbf{E}_1 : (\mathbf{codefirstf} : \mathbf{code}(z_1)) : \mathbf{v}, \mathbf{E}_1 : (\mathbf{codetlf} : \mathbf{code}(z_1)) : \mathbf{v}\rangle$. It easily follows from the definitions that $\lambda y.(\mathit{first} : (z_1 : y))$ and $\lambda y.(\mathit{tl} : (z_1 : y))$ both have lower complexity than z_1, so we can apply the inductive hypothesis.

The most interesting case is where $\mathbf{split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}([\,]), \mathbf{v}\rangle = \mathbf{T}$, so that $\mathit{emb}(z_1) : \mu_1(\mathbf{v})$ is a function item. In this case, we replace z_1 by $\mathit{uncurry}(z_1)$. As shown in Lemma C.5, this decreases the complexity, allowing us to apply the inductive hypothesis. Let **codeuncurry** be an FPO1 expression such that $\mathbf{codeuncurry} : \mathbf{code}(f) = \mathbf{code}(\mathit{uncurry}(f))$.

Recall that we assume that $\mathcal{A}$, the set of atoms, is finite. For the remainder of this section we take $\mathcal{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$. While we use the finiteness assumption throughout this section, the proofs can typically be easily modified to deal with the case where $\mathcal{A}$ is finitely generated. We define $\mathbf{E}_1$ as follows:

$$
\begin{array}{lll}
\lambda\mathbf{z}.(& \mathbf{if} & \mathbf{z} = \mathbf{code}(z_1) \\
 & \mathbf{then} & \mathbf{if}\ z_1 = \bot\ \mathbf{then}\ \Omega \\
 & & \mathbf{else\ if}\ z_1 = a_1\ \mathbf{then}\ \mathbf{a}_1 \\
 & & \ldots \\
 & & \mathbf{else\ if}\ z_1 = a_n\ \mathbf{then}\ \mathbf{a}_n \\
 & & \mathbf{else\ if}\ z_1 = \langle\!\langle\,\rangle\!\rangle\ \mathbf{then}\ \langle\,\rangle \\
 & & \mathbf{else\ if}\ z_1 \text{ is a nonempty sequence representation } \mathbf{then} \\
 & & \quad \mathbf{al} : \langle \mathbf{E}_1 : (\mathbf{codefirst} : \mathbf{z}), \mathbf{E}_1 : (\mathbf{codetl} : \mathbf{z})\rangle \\
 & & \mathbf{else\ if}\ z_1 \text{ is a function representation } \mathbf{then} \\
 & & \quad \lambda\mathbf{v}.(\\
 & & \qquad \mathbf{if\ split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}(a_1), \mathbf{v}\rangle\ \mathbf{then}\ \mathbf{a}_1 \\
 & & \qquad \ldots \\
 & & \qquad \mathbf{else\ if\ split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}(a_n), \mathbf{v}\rangle\ \mathbf{then}\ \mathbf{a}_n \\
 & & \qquad \mathbf{else\ if\ split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}(\langle\!\langle\,\rangle\!\rangle), \mathbf{v}\rangle\ \mathbf{then}\ \langle\rangle \\
 & & \qquad \mathbf{else\ if\ split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}(\langle\!\langle \bot, * \rangle\!\rangle), \mathbf{v}\rangle\ \mathbf{then} \\
 & & \qquad\quad \mathbf{al} : \langle \mathbf{E}_1 : (\mathbf{codefirstf} : \mathbf{z}) : \mathbf{v}, \mathbf{E}_1 : (\mathbf{codetlf} : \mathbf{z}) : \mathbf{v}\rangle \\
 & & \qquad \mathbf{else\ if\ split}_1 : \langle \mathbf{code}(z_1), \mathbf{code}([\,]), \mathbf{v}\rangle\ \mathbf{then} \\
 & & \qquad\quad \lambda\mathbf{w}.(\mathbf{E}_1 : (\mathbf{codeuncurry} : \mathbf{z}) : \langle \mathbf{v}, \mathbf{w}\rangle) \\
 & & \qquad\quad \mathbf{else}\ \Omega) \\
 & \mathbf{else} & \Omega)
\end{array}
$$

Using our discussion above as a guide, it is straightforward to show that $\mathbf{E}_1$ has the required properties. Formally, we show that if $||z|| = (k, k')$, then $\mu_1(\mathbf{E}_1) : \mathit{code}(z) = \mathit{emb}(z)$, assuming that the correctness of the definition of $\mathbf{E}_1$ for all z' such that $||z'|| < (k, k')$ and the correctness of the definition of $\mathbf{split}_1$ for all $\langle\!\langle \mathit{code}(z_1), \mathit{code}(z_2), y\rangle\!\rangle$ such that $\max(||z_1||, ||z_2||) < (k, k')$. We leave details to the reader.

Finally, we must deal with $\mathbf{B}_1$. We need to introduce a little more technical machinery first. Given a set S of representations in *Rep1*, define the set $CL(S)$ to be the smallest superset of S that is closed under $\sqcup$, in the sense that if x and y are compatible representations in $CL(S)$, then $x \sqcup y$ is in $D1^*$. It is easy to show that if S is finite, then so is $CL(S)$. (Indeed, we can show that $|CL(S)| \leq 2^{|S|}$: we simply add $\sqcup S'$ for every compatible subset S' of S. The resulting set is easily seen to be closed under $\sqcup$.) Moreover, given codes for all the elements in S, it is easy to compute codes for all the elements in $CL(S)$.

If z is a function representation and t is a representation, we define $z : t$ to be the representation $\sqcup\{z(x) : x \in dom(z), x \leq_r t\}$. We can now define $\mathbf{B}_1$ as follows:

$$
\begin{array}{ll}
\lambda\langle \mathbf{z}_1, \mathbf{z}_2, \mathbf{y}\rangle.(& \textbf{if } \mathbf{z}_1 = \mathbf{code}(z_1) \wedge \mathbf{z}_2 = \mathbf{code}(z_2) \textbf{ then} \\
& \quad \textbf{if } z_1 = \perp \textbf{ then T} \\
& \quad \textbf{else if } z_1 = a_1 \textbf{ then isatom} : \mathbf{y} \wedge \textbf{eqatom} : \langle \mathbf{a}_1, \mathbf{y}\rangle \\
& \quad \ldots \\
& \quad \textbf{else if } z_1 = a_n \textbf{ then isatom} : \mathbf{y} \wedge \textbf{eqatom} : \langle \mathbf{a}_n, \mathbf{y}\rangle \\
& \quad \textbf{else if } z_1 = \langle\!\langle\,\rangle\!\rangle \textbf{ then isseq} : \mathbf{y} \wedge \textbf{null} : \mathbf{y} \\
& \quad \textbf{else if } z_1 \text{ is a nonempty sequence representation } \textbf{then} \\
& \qquad \textbf{isseq} : \mathbf{y} \wedge \textbf{not} : (\textbf{null} : \mathbf{y}) \wedge \\
& \qquad \mathbf{B}_1 : \langle \textbf{codefirst} : \mathbf{z}_1, \textbf{codefirst} : \mathbf{z}_2, \textbf{first} : \mathbf{y}\rangle \wedge \\
& \qquad \mathbf{B}_1 : \langle \textbf{codetl} : \mathbf{z}_1, \textbf{codetl} : \mathbf{z}_2, \textbf{tl} : \mathbf{y}\rangle \\
& \quad \textbf{else if } z_1 \text{ is a function representation } \textbf{then} \\
& \qquad \textbf{isfunc} : \mathbf{y} \wedge \\
& \qquad \bigwedge_{t \in CL(dom(z_1) \cup dom(z_2))} (\mathbf{B}_1 \text{:} \ \langle \mathbf{code}(z_1 : t), \mathbf{code}(z_2 : t)\,, \\
& \qquad \mathbf{y} : (\mathbf{E}_1 : \mathbf{code}(t))\rangle) \\
& \textbf{else } \Omega)
\end{array}
$$

To show that this definition of $\mathbf{B}_1$ is correct, we again proceed by induction on complexity. Consider $\mu_1(\mathbf{B}_1) : \langle\!\langle code(z_1), code(z_2), y\rangle\!\rangle$, where $\max(||z_1||, ||z_2||) = (k, k')$. Suppose inductively that the definition of $\mathbf{E}_1$ is correct for all representations z such that $||z'|| < (k, k')$, while the definition of $\mathbf{B}_1$ is correct for all representations z_1 and z_2 such that $\max(||z_1||, ||z_2||) < (k, k')$. It is easy to see by inspecting the definition that if $\mu_1(\mathbf{B}_1) : \langle\!\langle code(z_1), code(z_2), y\rangle\!\rangle = T$, then it must be the case that $emb(z_1) \leq_I y$. The only difficulty comes if z_1 is a function representation. To see that $\mathbf{B}_1$ works correctly in this case, first suppose $emb(z_1) \leq_I y$. In that case, y must be a function item, so $isfunc : y = T$. Moreover, we must have $emb(z_1 : t) \leq_I y : emb(t)$ for each $t \in CL(dom(z_1) \cup dom(z_2))$. By Lemma C.1, if $t \in CL(dom(z_1) \cup dom(z_2))$, then $||t||_1 \leq \max_{t' \in dom(z_1) \cup dom(z_2)} ||t'||_1 < \max(||z_1||_1, ||z_2||_1)$. Thus, it follows that if $t \in CL(dom(z_1) \cup dom(z_2))$, then $||t|| < \max(||z_1||, ||z_2||)$. From the inductive hypothesis, we get that $\mu_1(\mathbf{B}_1) : \langle\!\langle code(z_1 : t), code(z_2 : t), (y : t)\rangle\!\rangle = T$ for each $t \in CL(dom(z_1) \cup dom(z_2))$. The result immediately follows.

Next suppose that z_1 and z_2 have no upper bound and $emb(z_2) \leq_I y$. If y is not a function item, then $isfunc : y = F$, and we get F as desired. If y is a function item then z_2 must be a function representation (since $emb(z_2) \leq_I y$ and $z_2 \neq \perp$, since z_1 and z_2 have no upper bound). Since $emb(z_2) \leq_I y$, we must have $emb(z_2 : t) \leq_I y : emb(t)$ for all $t \in CL(dom(z_1) \cup dom(z_2))$. Moreover, since z_1 and z_2 have no upper bound, there must be some $t \in CL(dom(z_1) \cup dom(z_2))$ such that $z_1 : t$ and $z_2 : t$ have no upper bound. (We remark that there might not be such a t in $dom(z_1) \cup dom(z_2)$; this is why we need the closure here.) By the inductive hypothesis, it follows that for this t, we have $\mu_1(\mathbf{B}_1) : \langle\langle code(z_1 : t), code(z_2 : t), y : emb(t)\rangle\rangle = F$. Again, the result now follows. We leave further details to the reader.

This completes the proof of the theorem. ▮

C.3 Proof of Theorem 6.5

Recall that we want to be able to define an expression $\mathbf{sup}_1$ that takes sups of increasing sequences. More formally, we want to define $\mathbf{sup}_1$ such that if $\{z_1, \ldots, z_k\}$ is an increasing sequence of finite elements in $D1^*$, then $\mu_1(\mathbf{sup}_1) : \langle\langle code(z_1), \ldots, code(z_k), *\rangle\rangle = z_k$. Again, it is useful to extend this definition to all representations, so we actually require that $\{z_1, \ldots, z_k\}$ is an increasing sequence of representations in *Rep1*, then $\mu_1(\mathbf{sup}_1) : \langle\langle code(z_1), \ldots, code(z_k), *\rangle\rangle = emb(z_k)$.

To define $\mathbf{sup}_1$, we again need three auxiliary functions, $\mathbf{sup}_2$, $\mathbf{split}_2$ and $\mathbf{B}_2$. The function $\mathbf{sup}_2$ is actually just a special case of $\mathbf{sup}_1$. Its definition is similar to that of $\mathbf{sup}_1$, but it is restricted to sequences with two terms other than $*$, rather an arbitrary number of terms. More precisely, we require that for all $z_1, z_2 \in$ *Rep1*,

- if $z_1 \leq_I z_2$, then $\mu_1(\mathbf{sup}_2) : \langle\langle code(z_1), code(z_2), *\rangle\rangle = emb(z_2)$
- $\mu_1(\mathbf{sup}_2) : \langle\langle code(z_1), \perp, *\rangle\rangle = z_1$

The function $\mathbf{B}_2$ is precisely the one we were hoping to define in FPO1, that, given $code(z)$ and y, returns $\mathbf{T}$ if $emb(z_1) \leq_I y$ and $\mathbf{F}$ if $emb(z_1)$ and y are incompatible. Thus, if z is a finite representation in *Rep1* and y is an arbitrary element of $\mathcal{D}^*$, we require that:

- $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y\rangle\rangle = T$ iff $emb(z) \leq_I y$
- $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y\rangle\rangle = F$ if $emb(z)$ and y are incompatible.

Once again, we emphasize that first condition is "if and only if", whereas the second condition is not.

The requirements for $\mathbf{split}_2$ are somewhat similar to $\mathbf{split}_1$. If f is a finite function representation, x is a minimal representation, and y is an arbitrary element in $D1^*$, then we require that:

- $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle = T$ iff $emb(x) \leq_I emb(f) : y$

- $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle = F$ iff $emb(t)$ and y are incompatible for all $t \in dom(f)$ such that $x \leq_r f(t)$.

Notice the first clause in $\mathbf{split}_2$ is identical to that of $\mathbf{split}_1$. The second clause is somewhat stronger: For $\mathbf{split}_1$ we required that if $emb(f) : y$ and $emb(x)$ are incompatible, then $\mu_1(\mathbf{split}_1) : \langle\langle code(f), code(x), y\rangle\rangle = F$. For $\mathbf{split}_2$ the second requirement is equivalent to requiring that $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle = F$ iff for all $y' \geq_I y$, we have $emb(x) \not\leq_I emb(f) : y'$. Notice that by the monotonicity of $emb(f)$, the second condition will automatically be met if $emb(x)$ and $emb(f) : y$ are already incompatible. Therefore, $\mathbf{split}_2$ returns F whenever $\mathbf{split}_1$ is required to, but is also required to return F in more general circumstances.

The definitions of $\mathbf{split}_2$, $\mathbf{sup}_2$, and $\mathbf{B}_2$ are much in the same spirit as those of $\mathbf{split}_1$, $\mathbf{E}_1$ and $\mathbf{B}_1$. Again, we start with $\mathbf{split}_2$, and then define $\mathbf{sup}_2$ and $\mathbf{B}_2$, and when proving correctness of our definitions, we proceed by induction on complexity, assuming the correctness of all three functions for arguments of lower complexity, and for previously defined functions for arguments of the same complexity. After we have defined these three functions and proved their correctness, we define and prove the correctness of $\mathbf{sup}_1$.

The definition of $\mathbf{split}_2$ is quite straightforward:

$$
\begin{aligned}
\lambda\mathbf{z}.\ \ & \mathbf{if}\ \mathbf{z} = \langle \mathbf{code}(f), \mathbf{code}(x), \mathbf{y}\rangle \text{ where } f \text{ is a function representation and} \\
& \quad x \text{ is a minimal representation} \\
& \mathbf{then}\ \textstyle\bigvee_{\{t\in dom(f) \mid x\leq_r f(t)\}} \mathbf{B}_2 : \langle \mathbf{code}(t), \mathbf{y}\rangle
\end{aligned}
$$

The proof that $\mathbf{split}_2$ has the required properties proceeds, as usual, by induction on complexity. Consider $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle$, where f is a finite function representation with $||f|| = (k, k')$ and x is a minimal element. If $emb(x) \leq_I emb(f) : y$, since x is minimal, it must be the case that there is some $t_i \in dom(f)$ such that $x \leq_r f(t_i)$ and $emb(t_i) \leq_I y$. Thus, by the induction hypothesis, we have that $\mu_1(\mathbf{B}_2) : \langle\langle code(t_i), y\rangle\rangle = T$, from which it follows that $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle = T$, as desired. Conversely, if $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle = T$, then it must be the case that for some $t \in dom(f)$ such that $x \leq_r f(t)$, we have $emb(t) \leq_I y$. It follows that $emb(x) \leq_I emb(f) : y$.

For the second requirement of $\mathbf{split}_2$, observe that $emb(t)$ and y are incompatible for all $t \in dom(f)$ such that $x \leq_r f(t)$ iff (by the induction hypothesis) $\mu(\mathbf{B}_2) : \langle\langle code(t), y\rangle\rangle = F$ for all $t \in dom(f)$ such that $x \leq_r f(t)$ iff $\mu_1(\mathbf{split}_2) : \langle\langle code(f), code(x), y\rangle\rangle = F$.

We now turn to $\mathbf{sup}_2$. Although $\mathbf{sup}_2$ is quite similar in spirit to $\mathbf{E}_1$, there are two significant differences in the requirements of $\mathbf{sup}_2$ which makes its definition somewhat more complicated than that of $\mathbf{E}_1$. The first difference is that the significant arguments to $\mathbf{sup}_2$ are sequence representations. This is fairly easy to take care of: we need to use **apall1** in a number of the clauses of $\mathbf{sup}_2$ to make sure that the same function gets applied to all arguments of the sequence representation which is the argument to $\mathbf{sup}_2$. The second difference comes in the case that the argument of $\mathbf{sup}_2$ is a sequence of function representations. Recall that when computing $\mathbf{E}_1$ in this case, we compute $\lambda\mathbf{v}.((\mathbf{E}_1 : \mathbf{code}(z_1)) : \mathbf{v})$. Using $\mathbf{split}_1$, we check if there is some minimal element x such that $x \leq_I emb(z_1) : \mu_1(\mathbf{v})$. If so, we take the appropriate action; otherwise, we return Ω. The problem is, when computing $\mathbf{sup}_2$, it is not necessarily appropriate to return Ω. That is, suppose we are attempting to compute $\mu_1(\mathbf{sup}_2) : \langle\langle code(z_1), code(z_2), *\rangle\rangle)$. If z_1 is a function representation, we want to compute $emb(z_1) : v$ for each argument v. If the result is not $\perp$, then we can use essentially the same techniques as for $\mathbf{E}_1$. However, if the result is $\perp$, then we need to compute $emb(z_2) : v$ $(= \mu_1(\mathbf{E}_1) : \mathbf{code}(z_2) : v)$.

We cannot do this using a normal conditional, since the conditional will diverge if $\mu_1(emb(z_1) : v) = \perp$. Instead, we need to use a parallel conditional. Roughly speaking, this will allow us to carry out all the necessary computations in parallel. Let *pcond* be the function item in $D1^*$ such that $pcond : x = \perp$ unless x is of the form $\langle x_1, x_2, x_3\rangle$, and (taking glb to denote the greatest lower bound, as usual)

$$pcond : \langle\langle x_1, x_2, x_3\rangle\rangle = \begin{cases} x_2 & \text{if } x_1 = T \\ x_3 & \text{if } x_1 = F \\ \mathrm{glb}(x_2, x_3) & \text{otherwise.} \end{cases}$$

Of course, we need to show that this is indeed a function item in $D1^*$, which amounts to showing that every pair of elements in $D1^*$ has a greatest lower bound, and that glb is a continuous function. We leave these details to the reader.

Observe that $or : \langle\langle x, y\rangle\rangle = pcond : \langle\langle x, T, y\rangle\rangle$, so that that *or* is definable in terms of *pcond*. As we now show, the converse also holds. (The proof is similar to the proof in [St91].)

Lemma C.6: *There exists an FPO1 (and,* a fortiori*, FP1*) expression* **pcond** *such that* $\mu_1(\mathbf{pcond}) = pcond$.

Proof: Let **nn** be an abbreviation for $\lambda\mathbf{y}.(\mathbf{isseq} : \mathbf{y} \wedge \mathbf{not} : \mathbf{null} : \mathbf{y})$. Thus, $\mathbf{nn} : \mathbf{y}$ returns true exactly if $\mu_1(\mathbf{y})$ is a nonempty sequence item. We now define **pcond** as follows.

$\lambda\langle\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\rangle.($
  if $((\mathbf{x}_1 = \mathbf{T} \wedge \mathbf{x}_2 = \mathbf{a}_1) \vee (\mathbf{x}_1 = \mathbf{F} \wedge \mathbf{x}_3 = \mathbf{a}_1) \vee (\mathbf{x}_2 = \mathbf{x}_3 = \mathbf{a}_1))$ **then** $\mathbf{a}_1$
    ...
  else if $((\mathbf{x}_1 = \mathbf{T} \wedge \mathbf{x}_2 = \mathbf{a}_n) \vee (\mathbf{x}_1 = \mathbf{F} \wedge \mathbf{x}_3 = \mathbf{a}_n) \vee (\mathbf{x}_2 = \mathbf{x}_3 = \mathbf{a}_n))$ **then** $\mathbf{a}_n$
  else if $((\mathbf{x}_1 = \mathbf{T} \wedge \mathbf{x}_2 = \langle\,\rangle) \vee (\mathbf{x}_1 = \mathbf{F} \wedge \mathbf{x}_3 = \langle\,\rangle) \vee (\mathbf{x}_2 = \mathbf{x}_3 = \langle\,\rangle))$ **then** $\langle\,\rangle$
  else if $((\mathbf{x}_1 = \mathbf{T} \wedge \mathbf{nn} : \mathbf{x}_2) \vee (\mathbf{x}_1 = \mathbf{F} \wedge \mathbf{nn} : \mathbf{x}_3) \vee (\mathbf{nn} : \mathbf{x}_2 \wedge \mathbf{nn} : \mathbf{x}_3))$
    then $\mathbf{al} : \langle\mathbf{pcond} : \langle\mathbf{x}_1, \mathbf{first} : \mathbf{x}_2, \mathbf{first} : \mathbf{x}_3\rangle, \mathbf{pcond} : \langle\mathbf{x}_1, \mathbf{tl} : \mathbf{x}_2, \mathbf{tl} : \mathbf{x}_3\rangle\rangle$
  else if $((\mathbf{x}_1 = \mathbf{T} \wedge \mathbf{isfunc} : \mathbf{x}_2) \vee (\mathbf{x}_1 = \mathbf{F} \wedge \mathbf{isfunc} : \mathbf{x}_3) \vee (\mathbf{isfunc} : \mathbf{x}_2 \wedge \mathbf{isfunc} : \mathbf{x}_3))$
    then $\lambda\mathbf{v}.(\mathbf{pcond} : \langle\mathbf{x}_1, \mathbf{x}_2 : \mathbf{v}, \mathbf{x}_3 : \mathbf{v}\rangle)$
  else $\Omega)$

In order to prove that $\mu_1(\mathbf{pcond}) = pcond$, it clearly suffices to show that $\mu_1(\mathbf{pcond}) : \langle\langle x_1, x_2, x_3 \rangle\rangle = pcond : \langle\langle x_1, x_2, x_3 \rangle\rangle$. By continuity, we can restrict attention to the case where x_2 and x_3 are finite. Define the *type* of a finite element x to be a pair (n, m) where n is the least k such that $x \in D_k$ and $m = 0$ if x is not a sequence item, otherwise it is the length of the sequence (not counting the $*$ if x is a sequence of indefinite length). The proof of correctness now proceeds by a straightforward induction on type (ordered lexicographically). We leave details to the reader. ∎

In analogy with **cond**, we use **parif** ... **then**...**else** to denote a use of the parallel conditional.

We define $\mathbf{sup}_2$ as follows (where we take **second** : **z** as an abbreviation for **first** : (**tl** : **z**); i.e., **second** picks the second element in a sequence):

$$
\begin{array}{l}
\lambda\mathbf{z}.(\ \ \mathbf{if\ first} : \mathbf{z} = \mathbf{code}(z_1)\ \ \mathbf{then} \\
\qquad\quad \mathbf{if}\ z_1 = \bot\ \ \mathbf{then}\ \mathbf{E}_1 : (\mathbf{second} : \mathbf{z}) \\
\qquad\quad \mathbf{else\ if}\ z_1 = a_1\ \mathbf{then}\ \mathbf{a}_1 \\
\qquad\quad \ldots \\
\qquad\quad \mathbf{else\ if}\ z_1 = a_n\ \mathbf{then}\ \mathbf{a}_n \\
\qquad\quad \mathbf{else\ if}\ z_1 = \langle\langle\,\rangle\rangle\ \mathbf{then}\ \langle\,\rangle \\
\qquad\quad \mathbf{else\ if}\ z_1\ \text{is a nonempty sequence representation}\ \mathbf{then} \\
\qquad\qquad \mathbf{al} : \langle \mathbf{sup}_2 : (\mathbf{apall1} : \mathbf{codefirst} : \mathbf{z})), \mathbf{sup}_2 : (\mathbf{apall1} : \mathbf{codetl} : \mathbf{z}) \rangle \\
\qquad\quad \mathbf{else}\ \lambda\mathbf{v}.(\\
\qquad\qquad \mathbf{parif\ split}_2 : \langle \mathbf{first} : \mathbf{z}, \mathbf{code}(a_1), \mathbf{v} \rangle\ \ \mathbf{then}\ \mathbf{a}_1 \\
\qquad\qquad \ldots \\
\qquad\qquad \mathbf{else\ parif\ split}_2 : \langle \mathbf{first} : \mathbf{z}, \mathbf{code}(a_1), \mathbf{v} \rangle\ \ \mathbf{then}\ \mathbf{a}_n \\
\qquad\qquad \mathbf{else\ parif\ split}_2 : \langle \mathbf{first} : \mathbf{z}, \mathbf{code}(\langle\langle\,\rangle\rangle), \mathbf{v} \rangle\ \mathbf{then}\ \langle\,\rangle \\
\qquad\qquad \mathbf{else\ parif\ split}_2 : \langle \mathbf{first} : \mathbf{z}, \mathbf{code}(\langle\langle \bot, * \rangle\rangle), \mathbf{v} \rangle\ \ \mathbf{then} \\
\qquad\qquad\quad \mathbf{al} : \langle \mathbf{sup}_2 : (\mathbf{apall1} : \mathbf{codefirstf} : \mathbf{z}) : \mathbf{v}, \\
\qquad\qquad\quad\ \ \mathbf{sup}_2 : (\mathbf{apall1} : \mathbf{codetlf} : \mathbf{z}) : \mathbf{v} \rangle \\
\qquad\qquad \mathbf{else\ parif\ split}_2 : \langle \mathbf{first} : \mathbf{z}, \mathbf{code}([\,]), \mathbf{v} \rangle\ \ \mathbf{then} \\
\qquad\qquad\quad \lambda\mathbf{w}.(\mathbf{sup}_2 : (\mathbf{apall1} : \mathbf{codeuncurry} : \mathbf{z}) : \langle \mathbf{v}, \mathbf{w} \rangle) \\
\qquad\qquad \mathbf{else}\ \mathbf{E}_1 : (\mathbf{second} : \mathbf{z}) : \mathbf{v}) \\
\quad\ \ \mathbf{else}\ \Omega)
\end{array}
$$

The proof of correctness of this definition proceeds by induction on the complexity of the first argument to $\mathbf{sup}_2$, assuming that $\mathbf{split}_2$ works correctly for all arguments of the same or lower complexity. The case where $\mu(\mathbf{second}) : z = \bot$ is similar to the proof for $\mathbf{E}_1$. So assume that $z = \langle\langle code(z_1), code(z_2), * \rangle\rangle$, $||z_1|| = (k, k')$ and $z_1 \leq_I z_2$. We need to show that $\mu_1(\mathbf{sup}_2) : emb(z) = emb(z_2)$, assuming that the correctness of the definition of $\mathbf{sup}_2$ for all sequences whose first term has complexity less than (k, k'), and the correctness of the definition of $\mathbf{split}_2$ provided its first two arguments have complexity at most (k, k'). For the most part, the proof is essentially the same as that of the correctness of $\mathbf{E}_1$. The most interesting case is where z_1 is a function representation. We sketch the proof of that case here, leaving the remaining cases to the reader. We must show that $\mu_1(\mathbf{sup}_2) : emb(z) : v = emb(z_2) : v$ for all v. The proof proceeds by cases, depending on the structure of $emb(z_1) : v$. If $emb(z_1) : v = \perp$, then the test of the **parif** in each case will return either F or $\perp$. In the case that $emb(z_2) : v = \perp$, it is easy to see that $\mu_1(\mathbf{sup}_2) : z = \perp$ as desired. In the case that $emb(z_2) : v = a_1$ for the atom a_1, all the **parif** tests other than the one for a_1 produce F (since $z_1 \leq_I z_2$). By the results already proven for $\mathbf{E}_1$, the final clause yields a_1. Since $\mu_1(\mathbf{split}_2 : \langle \mathbf{first} : \mathbf{z}, \mathbf{code}(a_1), \mathbf{v}\rangle) \in \{\perp, F\}$, by the parallel properties of **parif**, $\mu_1(\mathbf{sup}_2) : z = a_1$ as desired. The other subcases for different values of $emb(z_2) : v$ are similar. If $emb(z_1) : v \neq \perp$, then all the tests in the **parif** involving $\mathbf{split}_2$ will return either T or F (in fact, it was only to deal with the possibility that $emb(z_1) : v = \perp$ that we had to use **parif** rather the **if**). If $emb(z_1) : v$ is either an atom or the empty sequence $\langle\!\langle\,\rangle\!\rangle$, then we must have $emb(z_2) : v = emb(z_1) : v$, and the definition is easily seen to be correct. If $emb(z_1) : v$ is either a nonempty sequence item or a function item, then the result follows using the induction hypothesis (again, we need Lemma C.5 to guarantee that the complexity goes down in the case of function items). This completes the proof of correctness of $\mathbf{sup}_2$.

Next, we must deal with $\mathbf{B}_2$. It is useful to have the dual of **exists** by defining $\mathbf{all} = \lambda\mathbf{f}.(\mathbf{not} : (\mathbf{exists} : (\lambda\mathbf{x}.(\mathbf{not} : (\mathbf{f} : \mathbf{x})))))$. Thus, $\mu_1(\mathbf{all}) : f = T$ iff $f : \perp = T$ and $\mu_1(\mathbf{all}) : f = F$ iff $f : t = F$ for some observable t. It is also helpful to have the function $\mathbf{sup}_2' = \lambda\mathbf{v}.(\mathbf{sup}_2 : \langle \mathbf{first} : \mathbf{v}, \mathbf{grow} : \mathbf{v}, *\rangle)$ where $\mathbf{grow} : \mathbf{v}$ is $\mathbf{second} : \mathbf{v}$ if $\mathbf{v}$ is a pair of the form $\langle \mathbf{code}(x_1), \mathbf{code}(x_2)\rangle$ and $x_1 \leq_r x_2$; otherwise, **grow** returns Ω. We define $\mathbf{B}_2$ as follows:

$$
\begin{array}{l}
\lambda\langle \mathbf{z}, \mathbf{y}\rangle.(\quad \textbf{if } \mathbf{z} = \mathbf{code}(z) \textbf{ then} \\
\qquad\qquad \textbf{if } z = \perp \textbf{ then T} \\
\qquad\qquad \textbf{else if } z = a_1 \textbf{ then } \mathbf{isatom} : \mathbf{y} \wedge \mathbf{eqatom} : \langle \mathbf{a}_1, \mathbf{y}\rangle \\
\qquad\qquad \ldots \\
\qquad\qquad \textbf{else if } z = a_n \textbf{ then } \mathbf{isatom} : \mathbf{y} \wedge \mathbf{eqatom} : \langle \mathbf{a}_n, \mathbf{y}\rangle \\
\qquad\qquad \textbf{else if } z = \langle\!\langle\,\rangle\!\rangle \textbf{ then } \mathbf{isseq} : \mathbf{y} \wedge \mathbf{null} : \mathbf{y} \\
\qquad\qquad \textbf{else if } z \text{ is a nonempty sequence representation } \textbf{then} \\
\qquad\qquad\quad \mathbf{nn} : \mathbf{y} \wedge \mathbf{B}_2 : \langle \mathbf{codefirst} : \mathbf{z}, \mathbf{first} : \mathbf{y}\rangle \wedge \mathbf{B}_2 : \langle \mathbf{codetl} : \mathbf{z}, \mathbf{tl} : \mathbf{y}\rangle \\
\qquad\qquad \textbf{else if } z \text{ is a function representation } \textbf{then} \\
\qquad\qquad\quad \mathbf{isfunc} : \mathbf{y} \wedge \bigwedge_{t \in dom(z)} \mathbf{all} : (\lambda\mathbf{w}.(\\
\qquad\qquad\qquad\qquad \mathbf{B}_2 : \langle \mathbf{code}(z(t)), \mathbf{y} : (\mathbf{sup}_2' : \langle \mathbf{code}(t), \mathbf{w}\rangle)\rangle) \\
\qquad \textbf{else } \Omega)
\end{array}
$$

The proof that $\mathbf{B}_2$ is correct again proceeds by induction on complexity. More precisely, we want to show that $\mu_1(\mathbf{B}_2) : \langle\!\langle code(z), y\rangle\!\rangle$ satisfies its specifications provided that $\mathbf{B}_2$ satisfies the inductive hypothesis and that $\mu_1(\mathbf{sup}_2)$ gives the right answer when applied to sequences of the form $\langle\!\langle code(z_1), code(z_2), *\rangle\!\rangle$ with $||z_1|| \leq ||z||$. (Note for future reference that all that matters here is the complexity of the first argument of $\mathbf{sup}_2$; this is legitimate since in our proof of correctness for $\mathbf{sup}_2$, we proceeded by induction on the complexity of the first argument.) The only difficult case is if z is a function representation. If $emb(z) \leq_I y$, then y must be a function item and for all $t \in dom(z)$, it must be the case that $emb(z(t)) \leq_I y : emb(t)$. Moreover, for $t \in dom(z)$, by the induction hypothesis, we have $\mu_1(\mathbf{sup}_2) : \langle\langle code(t), \perp, * \rangle\rangle = emb(t)$, so (by the induction hypothesis and Corollary C.2) we have $\mu_1(\mathbf{B}_2) : \langle\langle code(z(t)), y : (\mu_1(\mathbf{sup}_2) : \langle\langle code(t), \perp, * \rangle\rangle)\rangle\rangle = T$. It now follows from the definition of $\mu_1(\mathbf{all})$ that $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y \rangle\rangle = T$ if $emb(z) \leq_I y$. It is easy to check that if $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y \rangle\rangle = T$, then $emb(z) \leq_I y$.

Similarly, it is easy to check that if $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y \rangle\rangle = F$, then $emb(z)$ and y are incompatible. Assume that $emb(z)$ and y are incompatible, with the goal of proving that $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y \rangle\rangle = F$. The only interesting case occurs when y is a function item. For each finite $t \in D1^*$, define $W_t = \{w \in D1^* | w \leq_I y : t, \ w \text{ is finite}\}$ and define $T_t = \{emb(z(t'))| t' \in dom(z), emb(t') \leq_I t\}$. Suppose that $W_t \cup T_t$ is pairwise compatible for all finite $t \in D1^*$. By Corollary C.3, we can let $f(t)$ be the least upper bound of $W_t \cup T_t$. Using the fact that $D1^*$ is algebraic, we can easily extend f to a continuous function on $D1^*$ that is an upper bound to $emb(z)$ and y. This contradicts our supposition. Therefore, there is a finite $t_1 \in D1^*$ such that $W_{t_1} \cup T_{t_1}$ is not pairwise compatible. Since both W_{t_1} and T_{t_1} are pairwise compatible, there is a $w_1 \in W_{t_1}$ and $t'_1 \in T_{t_1}$ that are not compatible (i.e. have no upper bound). By definition of T_{t_1}, we can find $t_0 \in dom(z)$ such that $emb(t_0) \leq_I t_1$ and such that $emb(z(t_0))$ and w_1 are incompatible. Therefore, $emb(z(t_0))$ and $y : t_1$ are incompatible as well. We have (using the induction hypothesis) that $\mu_1(\mathbf{B}_2) : \langle\langle code(z(t_0)), y : (\mu_1(\mathbf{sup}_2) : \langle\langle code(t_0), code(rep(t_1)), * \rangle\rangle)\rangle\rangle = F$. From the definition of **all**, it follows that $\mu_1(\mathbf{B}_2) : \langle\langle code(z), y \rangle\rangle = F$ as desired.

We are finally ready to define $\mathbf{sup}_1$. The definition is identical to that of $\mathbf{sup}_2$ except that all occurrences of $\mathbf{sup}_2$ are replaced by $\mathbf{sup}_1$, and the two occurences of $\mathbf{E}_1 : (\mathbf{second} : \mathbf{z})$ (in the second line and the last line of the pseudocode) are replaced by $\mathbf{sup}_1 : (\mathbf{tl} : \mathbf{z})$. We provide the definition here for completeness:

$$
\begin{array}{l}
\lambda \mathbf{z}.(\quad \textbf{if first} : \mathbf{z} = \mathbf{code}(z_1) \;\; \textbf{then} \\
\qquad\qquad \textbf{if } z_1 = \bot \;\; \textbf{then sup}_1 : (\mathbf{tl} : \mathbf{z}) \\
\qquad\qquad \textbf{else if } z_1 = a_1 \textbf{ then } \mathbf{a}_1 \\
\qquad\qquad \ldots \\
\qquad\qquad \textbf{else if } z_1 = a_n \textbf{ then } \mathbf{a}_n \\
\qquad\qquad \textbf{else if } z_1 = \langle\!\langle\, \rangle\!\rangle \textbf{ then } \langle\, \rangle \\
\qquad\qquad \textbf{else if } z_1 \text{ is a nonempty sequence representation } \textbf{then} \\
\qquad\qquad\quad \mathbf{al} : \langle \mathbf{sup}_1 : ((\mathbf{apall1} : \mathbf{codefirst}) : \mathbf{z})), \mathbf{sup}_1 : ((\mathbf{apall1} : \mathbf{codetl}) : \mathbf{z}) \rangle \\
\qquad\qquad \textbf{else } \lambda \mathbf{v}.(\\
\qquad\qquad\quad \textbf{parif split}_2 : \langle (\mathbf{first} : \mathbf{z}), \mathbf{code}(a_1), \mathbf{v} \rangle \;\; \textbf{then } \mathbf{a}_1 \\
\qquad\qquad\quad \ldots \\
\qquad\qquad\quad \textbf{else parif split}_2 : \langle (\mathbf{first} : \mathbf{z}), \mathbf{code}(a_n), \mathbf{v} \rangle \;\; \textbf{then } \mathbf{a}_n \\
\qquad\qquad\quad \textbf{else parif split}_2 : \langle (\mathbf{first} : \mathbf{z}), \mathbf{code}(\langle\!\langle\, \rangle\!\rangle), \mathbf{v} \rangle \textbf{ then } \langle\, \rangle \\
\qquad\qquad\quad \textbf{else parif split}_2 : \langle (\mathbf{first} : \mathbf{z}), \mathbf{code}(\langle\!\langle \bot, * \rangle\!\rangle), \mathbf{v} \rangle \;\; \textbf{then} \\
\qquad\qquad\qquad \mathbf{al} : \langle \mathbf{sup}_1 : (\mathbf{apall1} : \mathbf{codefirstf} : \mathbf{z}) : \mathbf{v}, \mathbf{sup}_1 : (\mathbf{apall1} : \mathbf{codetlf} : \mathbf{z}) : \mathbf{v} \rangle \\
\qquad\qquad\quad \textbf{else parif split}_2 : \langle (\mathbf{first} : \mathbf{z}), \mathbf{code}([\,]), \mathbf{v} \rangle \;\; \textbf{then} \\
\qquad\qquad\qquad \lambda \mathbf{w}.(\mathbf{sup}_1 : (\mathbf{apall1} : \mathbf{codeuncurry} : \mathbf{z}) : \langle \mathbf{v}, \mathbf{w} \rangle) \\
\qquad\qquad\quad \textbf{else sup}_1 : (\mathbf{tl} : \mathbf{z}) : \mathbf{v}) \\
\qquad \textbf{else } \Omega)
\end{array}
$$

The proof of correctness of this definition is essentially the same as that for $\mathbf{sup}_2$. The only difference is that now, since we are dealing with sequences of arbitrary length (not just length 2), we proceed by induction on the length of the sequence, with a subinduction on the complexity of the first argument to $\mathbf{sup}_2$. Note that we have already proved that $\mathbf{split}_2$ is correct, so we do not have to include the correctness of $\mathbf{split}_2$ in our induction hypothesis. The proof is almost identical to that of $\mathbf{sup}_2$, so we omit details here. This completes the proof of Theorem 6.5.

The reader may wonder at this point why we bothered with $\mathbf{sup}_2$. Could we not have omitted the definition of $\mathbf{sup}_2$, and used $\mathbf{sup}_1$ in its place in the definition of $\mathbf{B}_1$? Unfortunately, it seems that the answer is no. If we used $\mathbf{sup}_1$ in the definition of $\mathbf{B}_1$, then we could no longer in the proof of correctness of $\mathbf{sup}_1 : \mathbf{z}$ assume the correctness of $\mathbf{split}_2$ (since $\mathbf{split}_2$ uses $\mathbf{B}_1$, which in turn would use $\mathbf{sup}_1$). It seems that we could assume the correctness of $\mathbf{split}_2$ on arguments of "lower" complexity. To see the problem with this assumption, observe that in proving the correctness of $\mu_1(\mathbf{sup}_1) : \langle\!\langle code(z_1), \ldots, code(z_m), * \rangle\!\rangle$, we would need to assume the correctness of $\mu_1(\mathbf{split}_2) : \langle\!\langle code(z_i), \ldots \rangle\!\rangle$, $i = 1, \ldots, m$ (so that we do not run into problems as we apply the tail function to the sequence). This in turn would require the correctness of $\mu_1(\mathbf{B}_2) : \langle\!\langle code(z_i), \ldots \rangle\!\rangle$, which in turn would require the correctness of $\mu_1(\mathbf{sup}_1) : \langle\!\langle code(z_i), code(w), * \rangle\!\rangle$ for $i = 1, \ldots, m$ and *arbitrary* representations w. But this requires the correctness of $\mathbf{split}_2 : \langle\!\langle code(w), \ldots \rangle\!\rangle$ for arbitrary w. Suddenly we have lost the base of our induction!

Note this was not the case for $\mathbf{sup}_2$, where we only needed to assume the correctness of $\mu_1(\mathbf{split}_2) : \langle\langle code(z_1), \ldots\rangle\rangle$. We did not need to assume the correctness of $\mu_1(\mathbf{split}_2) : \langle\langle code(z_2), \ldots\rangle\rangle$, since $\mathbf{split}_2$ never gets applied to the second argument in the case of $\mathbf{sup}_2$. As in Plotkin's proof for LCF [Pl77], this apparently small difference is quite critical.

C.4 Proof of Theorems 7.1 and 7.4

In this section, we consider NFP. We first must define a set *NRep* of reprsentations of finite elements in ND^*. The definition is analogous to that of *Rep*. We take $NRep_0 = D_0$, and take $NRep_{n+1}$ to be the result of starting with $NRep_n$, and closing off under sequence representations, function representations, and *set representations*. Sequence representations and function respresentations are defined just as before; a set representation in $NRep_{n+1}$ has the form $\{x_1, \ldots, x_k\}$, where $x_1, \ldots, x_k \in NRep_n$. We can again define a function *emb* that maps elements of *NRep* to the finite elements in ND^*, and define a function *code* that maps elements of *NRep* to observable items in ND^*. The definition of *code* on atomic items, sequence representations, and function representations is just as before. We extend it to set representations as follows:

$$code(\{x_1, ..., x_n\}) = \langle\langle \langle\langle\ \rangle\rangle, code(x_1), ..., code(x_n)\rangle\rangle.$$

As before, we take $\mathbf{code}(x)$ to be the observable expression corresponding to the observable item $code(x)$.

We are now ready to prove Theorems 7.1 and 7.4. For convenience, we restate the results before proving them.

Theorem 7.1: *There is an NFP expression $\mathbf{E}_N$ such that for all for all finite elements x in* ND**, we have $\mu_N(\mathbf{E}_N{:}\mathbf{code}(x)) = x$.*

Proof: As usual, we define $\mathbf{E}_N$ on all representations, so that we require:

- $\mu_N(\mathbf{E}_N{:}\mathbf{code}(x)) = emb(x)$.

Again, it is helpful to define an expression $\mathbf{B}_N$ such that $\mu_n(\mathbf{B}_N) : \langle\langle code(x), y\rangle\rangle = T$ iff $emb(x) \leq_I y$. The definition of $\mathbf{E}_N$ is:

$$
\begin{array}{lll}
\lambda \mathbf{z}(& \textbf{if} & \mathbf{z} = \mathbf{code}(z_1) \\
 & \textbf{then} & \textbf{if } z_1 = a_1 \textbf{ then } \mathbf{a}_1 \\
 & & \ldots \\
 & & \textbf{else if } z_1 = a_n \textbf{ then } \mathbf{a}_n \\
 & & \textbf{else if } z_1 = \langle\!\langle\, \rangle\!\rangle \textbf{ then } \langle\, \rangle \\
 & & \textbf{else if } z_1 = \langle\!\langle x_1, \ldots, x_n \rangle\!\rangle \ \bigwedge x_1 \neq * \\
 & & \quad \textbf{then al} : \langle \mathbf{E}_N : \mathbf{code}(x_1), \mathbf{E}_N : \mathbf{code}(\langle\!\langle x_2, \ldots, x_n \rangle\!\rangle) \rangle \\
 & & \textbf{else if } z_1 = \langle\!\langle x_1, \ldots, x_n \rangle\!\rangle \ \bigwedge x_n \neq * \\
 & & \quad \textbf{then ar} : \langle \mathbf{E}_N : \mathbf{code}(\langle\!\langle x_1, \ldots, x_{n-1} \rangle\!\rangle), \mathbf{E}_N : \mathbf{code}(x_n) \rangle \\
 & & \textbf{else if } z_1 = \{x_1, \ldots, x_n\} \\
 & & \quad \textbf{then union} : \langle \mathbf{E}_N : \mathbf{code}(x_1), \ldots, \mathbf{E}_N : \mathbf{code}(x_n) \rangle \\
 & & \textbf{else if } z_1 = [x_1 \to y_1, \ldots, x_n \to y_n] \\
 & & \quad \textbf{then } \lambda \mathbf{v}.(\textbf{union} : \langle \textbf{cond} : \langle \mathbf{B}_N : \langle \mathbf{code}(x_1), \mathbf{v} \rangle, \mathbf{E}_N : \mathbf{code}(y_1), \Omega \rangle, \ldots, \\
 & & \qquad \textbf{cond} : \langle \mathbf{B}_N : \langle \mathbf{code}(x_n), \mathbf{v} \rangle, \mathbf{E}_N : \mathbf{code}(y_n), \Omega \rangle \rangle) \\
 & \textbf{else} & \Omega)
\end{array}
$$

In the definition of $\mathbf{B}_N$, we use the expression $\mathbf{andof}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, which is an abbreviation for $\mathbf{cond} : \langle \mathbf{x}_1, \ldots \mathbf{cond} : \langle \mathbf{x}, \mathbf{T}, \Omega \rangle \ldots, \Omega \rangle$. Thus, $\mu(\mathbf{andof}(\mathbf{x}_1, \ldots, \mathbf{x}_n))$ is T if $\mu(\mathbf{x}_i) = T$ for $i = 1, \ldots, n$, and is $\perp$ otherwise. The definition of $\mathbf{B}_N$ is:

$$
\begin{array}{lll}
\lambda \langle \mathbf{x}, \mathbf{y} \rangle .(& \textbf{if} & \mathbf{x} = \mathbf{code}(x_1) \\
 & \textbf{then} & \textbf{if } x_1 = a_1 \textbf{ then andof}(\textbf{isatom} : \mathbf{y}, \textbf{eqatom} : \langle \mathbf{a}_1, \mathbf{y} \rangle) \\
 & & \ldots \\
 & & \textbf{else if } x_1 = a_n \textbf{then andof}(\textbf{isatom} : \mathbf{y}, \textbf{eqatom} : \langle \mathbf{a}_n, \mathbf{y} \rangle) \\
 & & \textbf{else if } x_1 = \langle\!\langle\, \rangle\!\rangle \textbf{ then null} : \mathbf{y} \\
 & & \textbf{else if } x_1 = \langle\!\langle x'_1, \ldots, x'_n \rangle\!\rangle \ \bigwedge x'_1 \neq * \\
 & & \quad \textbf{then andof}(\textbf{isseq} : \mathbf{y}, \textbf{not} : (\textbf{null} : \mathbf{y}), \\
 & & \qquad \mathbf{B}_N : \langle \mathbf{code}(x'_1), \textbf{first} : \mathbf{y}, \mathbf{B}_N : \langle \mathbf{code}(\langle\!\langle x'_2, \ldots, x'_n \rangle\!\rangle) \\
 & & \textbf{else if } x_1 = \langle\!\langle x'_1, \ldots, x'_n \rangle\!\rangle \ \bigwedge x'_n \neq * \\
 & & \quad \textbf{then andof}(\textbf{isseq} : \mathbf{y}, \textbf{not} : (\textbf{null} : \mathbf{y}), \\
 & & \qquad \mathbf{B}_N : \langle \mathbf{code}(x'_n), \textbf{last} : \mathbf{y}, \mathbf{B}_N : \langle \mathbf{code}(\langle\!\langle x'_1, \ldots, x'_{n-1} \rangle\!\rangle), \textbf{tr} : \mathbf{y}) \\
 & & \textbf{else if } x_1 = \{x'_1, \ldots, x'_n\} \\
 & & \quad \textbf{then andof}(\mathbf{B}_N : \langle \mathbf{code}(x'_1), \mathbf{y} \rangle, \ldots, \langle \mathbf{code}(x'_n), \mathbf{y} \rangle) \\
 & & \textbf{else if } x_1 = [x'_1 \to y'_1, \ldots, x'_n \to y'_n] \\
 & & \quad \textbf{then andof}(\textbf{isfunc} : \mathbf{y}, \mathbf{B}_N : \langle \mathbf{code}(y'_1), \mathbf{y} : (\mathbf{E}_N : \mathbf{code}(x'_1)) \rangle, \ldots, \\
 & & \qquad \mathbf{B}_N : \langle \mathbf{code}(y'_n), \mathbf{y} : (\mathbf{E}_N : \mathbf{code}(x'_n)) \rangle) \\
 & \textbf{else} & \Omega)
\end{array}
$$

The proofs that $\mathbf{E}_N$ and $\mathbf{B}_N$ have the desired properties are left to the reader. ∎

Theorem 7.4: *There is an expression* $\mathbf{sup}_N$ *in NFP such that if* $\langle z_1, \ldots, z_k \rangle$ *is an increasing sequence of finite elements in* ND^*, *then*

$$\mu_1(\mathbf{sup}_N) : \langle\!\langle code(z_1), \ldots, code(z_k), * \rangle\!\rangle = z_k.$$

Proof: Again, we generalize the definition of $\mathbf{sup}_N$ to arbitrary representations. Define $\mathbf{sup}_N = \lambda\mathbf{x}.(\mathbf{union} : \langle \mathbf{first} : \mathbf{x}, \mathbf{sup}_N : (\mathbf{tl} : \mathbf{x})\rangle)$. It is easy to see that this definition has the desired properties. ∎

C.5 Rewrite rules for FPE and FPE*

Now that we have defined representations, we can also give the rewrite rules for the languages FPE and FPE* as defined in Section 8. For FPE we added the primitive function symbol **E**; for FPE*, we also added **sup**. As usual, we actually define the rules on the codes of all finite representations (not just canonical representations). In these rules we use **andof**, which was defined in Section C.4; recall that $\mu(\mathbf{andof}(\mathbf{x}_1, \ldots, \mathbf{x}_n))$ is T if $\mu(\mathbf{x}_i) = T$ for $i = 1, \ldots, n$, and is $\perp$ otherwise.

The rules presented for **sup** are only weakly sound. Despite the fact that they are only weakly sound, as we observed in Appendix A, by generalizing the notion of strong completeness appropriately, we still get Full Adequacy. Recall that $\langle\!\langle * \rangle\!\rangle$ is identified with $\perp$.

$\mathbf{E} : \mathbf{code}(a) \to \mathbf{a}$ if $\mathbf{a}$ is an atom
$\mathbf{E} : \mathbf{code}(\langle\!\langle\,\rangle\!\rangle) \to \langle\,\rangle$
$\mathbf{E} : \mathbf{code}(\langle\!\langle x_1, x_2, \ldots, x_n \rangle\!\rangle) \to \mathbf{al}{:}\langle \mathbf{E} : \mathbf{code}(x_1), \mathbf{E} : \mathbf{code}(\langle\!\langle x_2, \ldots, x_n \rangle\!\rangle)\rangle$ if x_1 is not $*$
$\mathbf{E} : \mathbf{code}(\langle\!\langle x_1, x_2, \ldots, x_n \rangle\!\rangle) \to \mathbf{ar}{:}\langle \mathbf{E} : \mathbf{code}(\langle\!\langle x_1, \ldots, x_{n-1} \rangle\!\rangle), \mathbf{E} : \mathbf{code}(x_n)\rangle$ if x_n is not $*$
$\mathbf{E} : \mathbf{code}(f) : \mathbf{z} \to \mathbf{E} : \mathbf{code}(y)$ if f is a function representation, and y is a finite element such that $y \leq_I \sqcup\{f(x) : x \in \mathit{dom}(f)$ and $\mathbf{E} : \mathbf{code}([x \to T]) : \mathbf{z} \to^* \mathbf{T}\}$
$\mathbf{E} : \mathbf{code}([a \to T]) : \mathbf{a} \to \mathbf{T}$
$\mathbf{E} : \mathbf{code}([\langle\!\langle\,\rangle\!\rangle \to T]) : \langle\,\rangle \to \mathbf{T}$
$\mathbf{E} : \mathbf{code}([\langle\!\langle x_1, \ldots, x_n \rangle\!\rangle \to T]) : \mathbf{z} \to \mathbf{andof}(\mathbf{E} : \mathbf{code}([x_1 \to T]) : (\mathbf{first} : \mathbf{z}), \mathbf{E} : \mathbf{code}([\langle\!\langle x_2, \ldots, x_n \rangle\!\rangle \to T]) : (\mathbf{tl} : \mathbf{z}))$ if $x_1 \neq *$
$\mathbf{E} : \mathbf{code}([\langle\!\langle x_1, \ldots, x_n \rangle\!\rangle \to T]) : \mathbf{z} \to \mathbf{andof}(\mathbf{E} : \mathbf{code}([x_n \to T]) : (\mathbf{last} : \mathbf{z}), \mathbf{E} : \mathbf{code}([\langle\!\langle x_1, \ldots, x_{n-1} \rangle\!\rangle \to T]) : (\mathbf{tr} : \mathbf{z}))$ if $x_n \neq *$
$\mathbf{E} : \mathbf{code}([[x_1 \to y_1, \ldots, x_n \to y_n] \to T]) : \mathbf{z} \to \mathbf{andof}(\mathbf{E} : \mathbf{code}([y_1 \to T]) : (\mathbf{z} : (\mathbf{E} : \mathbf{code}(x_1))), \ldots, \mathbf{E} : \mathbf{code}([y_n \to T]) : (\mathbf{z} : (\mathbf{E} : \mathbf{code}(x_n))))$
$\mathbf{sup} : \mathbf{x} \to \mathbf{E} : (\mathbf{first} : \mathbf{x})$
$\mathbf{sup} : \mathbf{x} \to \mathbf{sup} : (\mathbf{tl} : \mathbf{x})$

The above rules were presented in a weakly sound form so as make them more readable. It is possible to construct a sound variant of these rules. Roughly speaking, this can be done by repeating rules as necessary for **E** where f is a function representation and for **sup**. This would involve techniques such as the ones used in Section C.3, where we ensured that all the codes in **sup** got altered in the same that the first code did during a recursive call. We omit details here.

Acknowledgements: We would like to thank Albert Meyer for inspiring us to consider these questions and pointing out the relevance of the results in [Pl77]. John Williams first conjectured that apply-to-all was not definable in FP, and David Chase encouraged us to prove it. Samson Abramsky suggested the importance of showing the lack of full abstraction for FPO. John Backus, Ron Fagin, Matthias Felleisen, and John Williams provided useful comments on a draft of this paper. The referees also provided thoughtful and helpful comments.

References

[Ab90] S. Abramsky, The Lazy Lambda Calculus, in *Research Topics in Functional Programming* edited by David Turner, Addison-Wesley (1990), pp. 65-116.

[Ab91] S. Abramsky, Domain Theory in Logical Form, *Annals of Pure and Applied Logic* **58**:1/2 (1991), pp. 1–77.

[Ba78] J. W. Backus, Can programming be liberated from the von Neumann style? A functional style and its algebra of programs, *CACM* **21**:8 (1978), pp. 613-641.

[Cu86] P.-L. Curien, *Categorical Combinators, Sequential Algorithms and Functional Programming*, Pitman Publishing Limited (1986).

[Gu87] C. Gunter, Universal Profinite Domains, *Information and Computation* (1987), pp. 1-30.

[HWW90] J.Y. Halpern, J.H. Williams, and E.L. Wimmers, Completeness of rewrite rules and rewrite strategies for FP, *Journal of the ACM* **37**:1 (1990), pp. 86–143.

[Mi73] R. Milner, Models of LCF, Memo. AIM-186, Stanford Artificial Intelligence Laboratory, Stanford University (1973).

[Mi77] R. Milner, Fully abstract models of typed lambda-calculi, *Theoretical Computer Science* **4** (1977).

[Pl77] G.D. Plotkin, LCF considered as a programming language, *Theoretical Computer Science* **5** (1977), pp. 223-255.

[SP82] M.B. Smyth and G.D. Plotkin, The category-theoretic solution of recursive domain equations, *SIAM Journal on Computing* **11**:4 (1982), pp. 761-783.

[St90] A. Stoughton, Equationally fully abstract models of PCF, *Proc. 5th MFPS*, Lecture Notes in Computer Science No. 442, Springer-Verlag (1990).

[St91] A. Stoughton, Interdefinability of parallel operations in PCF, *Theoretical Computer Science* **79** (1991), pp. 357–358.

[Wa78] C.P. Wadsworth, Approximate reduction and lambda calculus models, *SIAM Journal of Computing* 7 (1978), pp. 337-356.

[Wi81] J.H. Williams, Formal representations for recursively defined functional programs, in *Formalization of Programming Concepts*, Lecture Notes in Computer Science No. 107, Springer-Verlag (April, 1981), pp. 460-470.