FULL ABSTRACTION AND LIMITING COMPLETENESS IN EQUATIONAL LANGUAGES

Satish R. THATTE
Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109, U.S.A.

Communicated by M. Nivat
Received January 1988

Abstract. This paper introduces a notion of full abstraction for equational languages under which each language has a unique fully abstract model which can also be characterized as the *final* object in a category of coherent computable models for the language. We describe two potentially different approaches to limiting completeness with respect to the fully abstract model—a traditional one based on normal forms and a new one based on the usable data content of terms. The former is used to prove limiting completeness for the language of *regular systems* [5] which includes as subsets and restrictions the equational parts of many other languages. The latter is used to define an abstract version of limiting completeness based on *information systems* [17] which allows us to derive a set of sufficient conditions for an equational language to be limiting complete for its fully abstract semantics. We discuss cases where the two notions coincide—as they do for regular systems—and cases where they do not. We believe that limiting completeness based on observable data content accurately reflects programming intuition. If this thesis is accepted, the appropriateness of the corresponding definition of approximant can be seen as a *design principle* to test the mutual suitability of the parameters defining a language.

Contents

1. Introduction

We present a study of the semantics of programming languages (or those of their parts) which are based on the use of equational methods. Examples include Standard ML [11], Miranda[1] [20] and the equational language of Hoffmann and O'Donnell [5]. Operationally, equational computation can be understood as term rewriting and various aspects of this behavior have been thoroughly studied [6, 7, 14, 19]. The denotational properties have been studied by Nivat [13] and the ADJ group [1, 2] using the notion of initiality. Raoult and Vuillemin [16] have used a more complex approach to establish a correspondence between the denotational and operational properties of their language along the lines of full abstraction [10]. Intuitively, initiality formalizes a view of equational computation as a completely transparent visible process, while full abstraction views an equational program as a black box with both visible and invisible aspects; the visible values being typically those involved in input/output. The latter is a realistic view for many applications and it is the one we are concerned with in this paper.

We would like the denotational semantics of an equational language to possess two properties. It should on the one hand be abstract enough to semantically identify two expressions whenever there is no visible way to distinguish between them, i.e., it should not impose distinctions based on differences in internal representation. This is the intuition in full abstraction which is important in applications such as program transformation. On the other hand, the semantics should not be so abstract that equational rewriting is too weak to even approach the computational realization of semantic equality for closed (or ground) expressions. Otherwise the semantics could not be considered realistic. The notion of computability in the limit is required since equationally defined expressions can intuitively denote values with infinite extensions (like infinite lists or trees). One therefore cannot expect semantic equality to be recursive, and one certainly cannot expect simple equational deduction to be complete even for ground equations with respect to a sufficiently abstract semantics. Wadsworth [21] introduced the idea of *limiting completeness* when faced with a similar problem in his study of the D_∞ model of the λ-calculus. In the D_∞ model, semantic domains containing infinite values are constructed as completions of partially ordered sets of finite values in which the infinite values arise as least upper bounds (LUBs) of all their finite approximations. Limiting completeness uses the idea of *approximants*, which are a special class of expressions representing finite values—each expression has a best direct approximant which represents its "manifest" value, i.e., that part of its value which is apparent without further evaluation. The operational semantics of a language (e.g., β-reduction for the λ-calculus) is complete in the limit if the LUB of the (possibly infinite) set of approximants of the results of finite partial evaluations of an expression is (isomorphic to) the denotation of the expression. In short, this is the operational counterpart

[1] Miranda is a trademark of Research Software Ltd.

of the idea of continuity, and it is sufficient to ensure computation of semantic equality in the limit.

Our work can be viewed as an application of these ideas to equational languages. We show that it is possible to extend the idea of *finality* to models of equational languages to obtain a model which is the "right one" according to both of the criteria outlined above. We give a general definition of full abstraction for equational languages using sets of *visible terms* (e.g., $\{true, false, 0, succ^n(0)\}$) as observable values in place of ground domains. A unique fully abstract model in this sense exists for each language. We then show in Section 2 that the final model for any equational language is fully abstract provided that

- models for individual programs are constrained to be *computable* for visible values;
- models for entire equational languages are constrained to be *coherent* with respect to language parameters in a natural sense.

The rest of the paper explores limiting completeness with respect to the final, or equivalently, fully abstract model. We describe two potentially different approaches to defining approximants; a traditional one based on normal forms and a new one based on the usable data content of terms. In Section 3 the former is used to prove limiting completeness for the language of *regular systems* [5] (elsewhere called nonambiguous linear term rewriting systems [6]), which is the most general syntactically well-demarcated language of deterministic rewriting systems in current use and includes as subsets and restrictions the equational parts of the three languages mentioned at the beginning. This confirms a conjecture in [6]. The latter approach is used in Section 4 to describe a more abstract version of limiting completeness based on *information systems* [17] which allows us to derive a set of sufficient conditions for an equational language to be limiting complete for its final semantics. We discuss cases where the two notions of approximant coincide—as they do for regular systems—and cases where they do not. The two notions are not comparable in their generality in that there are cases where one leads to limiting completeness while the other does not, and vice versa. Finally, it is worth noting that besides allowing the construction of a language independent limiting operational model, the use of information systems reveals the fact that the functions and values computed by equational programs usually possess the pleasant properties that are traditionally assumed for the base values and functions which are used with λ-calculus dialects, and often supplied in real life by equational definitions.

2. Languages, models, full abstraction and finality

We begin by defining the precise notions of program, language, and model which we use in the rest of this paper. Since we are considering first-order equational programs, we borrow most of the machinery for describing syntactic and semantic aspects of individual programs from the literature on the semantics of algebraic specifications. It is assumed that the reader is familiar with the basic notions of algebraic semantics. An excellent introduction is given in [2]. The description of an equational *language*, in addition to describing a collection of programs, also needs to specify the language specific aspects of observable behavior. In our case this consists of the notions of visible terms and program extension.

Visible terms are one way to formalize *what* is visible. Unlike the λ-calculus, where the basic domains and functions used for this purpose are usually external to the language, our visible values are also algebraically specified. We choose a simple but realistic approach in which the language description specifies a signature of visible constructor symbols, and any well-formed term constructed from these symbols is considered visible. The following is a typical example of such a signature:

$$zero\colon \to Nat, \qquad succ\colon Nat \to Nat,$$

$$true\colon \to Bool, \qquad false\colon \to Bool.$$

Thus *true* and *succ*(*zero*) are visible values of sort *Bool* and *Nat* respectively, but even if *plus*(*zero*, *zero*) is of sort *Nat*, it is not visible since it involves the non-visible operator *plus*. We assume that the notion of *what* is visible is common to all programs in a language.

Program extension is involved in deciding what is a legitimate context for observation of "external" behavior. Clearly, the specific notion of context one uses is crucial to the meaning of full abstraction. Plotkin [15] and Milner [10] first explored full abstraction for typed dialects of the λ-calculus using expression contexts. Expression contexts were also used by Wadsworth [21] for the untyped λ-calculus. This makes sense because a program in a λ-calculus *is* an expression. When full abstraction was considered by Raoult and Vuillemin [16] for an equational language, they extended the notion of context to include all expressions in all possible extensions of a program, where the extension relation was taken to be simply inclusion between programs as sets of equations. Their adaptation takes account of the full power of discrimination to which the meaning of an expression may be subjected in practice. Of course, languages may use a notion of extension which does not coincide with inclusion. For instance, languages like Miranda require the equations for a single function definition to be grouped together as a syntactic unit. In such cases, extension is most naturally understood as consisting of new function definitions, rather than the arbitrary addition of equations including new equations for previously defined functions. We therefore include the specific notion of extension as a parameter in the language description.

Formally, an equational language will be a triple $(\mathfrak{R}, \leqslant, \Theta)$ where $\mathfrak{R}$ is the set of programs ($\mathcal{R}, \mathcal{S}$ range over individual programs), $\leqslant$ denotes program extension which is assumed to be a partial order over $\mathfrak{R}$ and Θ is a (many-sorted) signature over the *visible* sorts V. All Θ-terms will be called *visible* terms. A program $\mathcal{R}$ consists of

(1) A signature $\Sigma \supseteq \Theta$ over a set $S \supseteq V$ of sorts.

(2) A countable set, also called $\mathcal{R}$, of (oriented) Σ-equations, thought of as rewrite rules.

It is assumed that all Θ-terms are normal forms in all programs. We use the program name as a subscript to distinguish between different signatures when needed. It is assumed that if $\mathcal{R} \preccurlyeq \mathcal{S}$ then $\mathcal{R} \subseteq \mathcal{S}$ and $\Sigma_{\mathcal{R}} \subseteq \Sigma_{\mathcal{S}}$.

The *initial* Σ-algebra will be denoted by T_Σ, and T_Σ will also be used to denote the (many-sorted) set of all Σ-terms. Given a Σ-algebra A, the unique evaluation morphism from T_Σ to A which simply evaluates terms according to their interpretation in A will be denoted ambiguously by A itself. A $\Sigma_{\mathcal{R}}$-algebra A *satisfies* $\mathcal{R}$ if $A(t_1) = A(t_2)$ for every ground instance t_1, t_2 of an equation in $\mathcal{R}$. It *respects* visible terms if $A(t_1) \neq A(t_2)$ for any pair t_1, t_2 of distinct Θ-terms. It is $\Sigma_{\mathcal{R}}$*-reachable* if the unique homomorphism from $T_{\Sigma_{\mathcal{R}}}$ to A is *surjective*. Any reasonable program model must satisfy these three conditions. The reason for the first is obvious. Respect for visible terms ensures that visible distinctions such as those between, e.g., *true* and *false*, are not compromised. This is just a version of the idea of *consistency*. Reachability is not always considered essential. For instance, the standard models of the λ-calculus are not reachable. For a final semantics, reachability is technically convenient, and in any case, one is really only interested in the reachable part of any model. It is usually easy to *restrict* a model to its reachable part.

One of the basic results of algebraic semantics [2] is that there is an initial algebra in the category of $\Sigma_{\mathcal{R}}$-algebras which satisfy $\mathcal{R}$. This algebra semantically identifies ground terms exactly when their equality is provable from the equations. We use the notation $=_{\mathcal{R}}$ to denote semantic equality in the initial algebra for $\mathcal{R}$, or, equivalently, provable equality within $\mathcal{R}$. It is obvious that the initial algebra for $\mathcal{R}$ satisfies $\mathcal{R}$, respects visible terms and is $\Sigma_{\mathcal{R}}$-reachable.

In addition to the three properties defined above, a model for a program $\mathcal{R}$ needs to be computable for visible results in the sense of the following definition.

2.1. Definition. A $\Sigma_{\mathcal{R}}$-algebra A is $\mathcal{R}$*-computable* if for each $t \in T_{\Sigma_{\mathcal{R}}}$, $v \in T_\Theta$, $A(t) = A(v) \Leftrightarrow t =_{\mathcal{R}} v$.

To put it another way, a $\Sigma_{\mathcal{R}}$-algebra is $\mathcal{R}$-computable if the word problem for visible terms is solvable with simple equational deduction. The connection between $=_{\mathcal{R}}$ and "real" computation is that for confluent programs (see Section 3.1 for a definition), equational deduction can be implemented with rewriting. This notion of computability is natural for the semantics of programs (cf. the "termination lemma" in [16]) because the ultimate means of observation in a program is the output of actual execution, and visible terms represent values that can be output. It would be strange to claim in such an environment that an expression is semantically equal to 1, say, when upon evaluation it does not yield 1 as output. Our definition of computability differs from the one given by Goguen and Meseguer [3] in two ways. They require the word problem to be solvable for *all* terms, but do not restrict the decision procedure to be simply $=_{\mathcal{R}}$. We feel that their definition is appropriate for models of data type specifications rather than programs. Note that $\mathscr{P}$-computability is a stronger version of respect for distinctions between visible terms. The respect condition can therefore be left out when computability is required.

A $\Sigma_{\mathscr{P}}$-algebra A is a *model* for a program $\mathscr{P}$ if A satisfies $\mathscr{P}$, is $\Sigma_{\mathscr{P}}$-reachable and $\mathscr{P}$-computable. Should a model for a language be anything more than a collection of program models? It is natural to require some coherence conditions based on the common language parameters—visible terms and extensions. Visible terms are already respected by program models. Technically, we need a coherence condition related to extensions which will allow us to derive the fully abstract model (yet to be defined, but known to depend on extensions to provide contexts for observation) as the final model in the category of language models. There is a very natural condition which meets this criterion. The elements of an equational program always support a dual interpretation (mediated by confluence or some similar restriction) as *equations* that must be *satisfied* and as *rewrite rules* that must be able to *compute* visible results. We have assumed that a program extension always amounts to adding new equations. Intuitively, when one adds new equations, one does not mean to discriminate between expressions that were known to be equal as a consequence of equations that were present before the addition. One could not reason about a program with any security if the reasoning may be rendered invalid when the program is extended. This leads to the notion of *stable extension* which is the coherence condition we need. Given $\mathscr{P} \leqslant \mathscr{S}$, a model B for $\mathscr{S}$ is said to be a *stable extension* of a model A for $\mathscr{P}$ if there is a *homomorphism* $h: A \to B|_{\Sigma_{\mathscr{P}}}$ where $B|_{\Sigma_{\mathscr{P}}}$ denotes the reduct of B to $\Sigma_{\mathscr{P}}$, i.e., B considered as a $\Sigma_{\mathscr{P}}$-algebra. A less abstract way to understand stable extension when A and B are $\Sigma_{\mathscr{P}}$-reachable is that it requires $A(t_1) = A(t_2) \Rightarrow B(t_1) = B(t_2)$ for all $\Sigma_{\mathscr{P}}$-terms t_1, t_2. A *model* $\mathscr{A}$ for a language $(\mathfrak{R}, \leqslant, \Theta)$ assigns an algebra $\mathscr{A}_{\mathscr{P}}$ to each program $\mathscr{P} \in \mathfrak{R}$, such that

(1) $\mathscr{A}_{\mathscr{P}}$ is a model for $\mathscr{P}$;

(2) $\mathscr{P} \leqslant \mathscr{S}$ implies $\mathscr{A}_{\mathscr{S}}$ is a stable extension of $\mathscr{A}_{\mathscr{P}}$.

Algebraic semantics traditionally considers initiality or finality in categories of algebras for individual programs. This approach can be extended to models of languages in a natural way. If $\mathscr{A}$ and $\mathscr{B}$ are models of a language $(\mathfrak{R}, \leqslant, \Theta)$, then a morphism $\phi: \mathscr{A} \to \mathscr{B}$ is simply a collection $\{\phi_{\mathscr{P}}: \mathscr{A}_{\mathscr{P}} \to \mathscr{B}_{\mathscr{P}} \mid \mathscr{P} \in \mathfrak{R}\}$ of $\Sigma_{\mathscr{P}}$-homomorphisms. It is easy to show that, whenever $\mathscr{P} \leqslant \mathscr{S}$, and $\chi^{\mathscr{S}}_{\mathscr{P}}$ and $\varphi^{\mathscr{S}}_{\mathscr{P}}$ are the homomorphisms from the models of $\mathscr{P}$ to those of $\mathscr{S}$ in $\mathscr{A}$ and $\mathscr{B}$ respectively, the following diagram commutes

$$\begin{tikzcd}
\mathscr{A}_{\mathscr{S}}|_{\Sigma_{\mathscr{P}}} \arrow[r, "\phi_{\mathscr{S}}"] & \mathscr{B}_{\mathscr{S}}|_{\Sigma_{\mathscr{P}}} \\
\mathscr{A}_{\mathscr{P}} \arrow[u, "\chi^{\mathscr{S}}_{\mathscr{P}}"] \arrow[r, "\phi_{\mathscr{P}}"] & \mathscr{B}_{\mathscr{P}} \arrow[u, "\varphi^{\mathscr{S}}_{\mathscr{P}}"']
\end{tikzcd}$$

The models of a language together with the morphisms between them clearly form a category. The existence of an initial language model follows easily from the existence of initial program models—the former is simply the collection of all initial program models together with the obvious homomorphisms guaranteed by initiality. We now show that the final model is also guaranteed to exist, and is moreover always the fully abstract model for the language.

In defining the fully abstract model for an arbitrary equational language, the key step is a definition of observably distinct behavior. Informally, one can say that two expressions are observably distinct when there is a context in which insertion of each leads to two distinct visible results. In classical terminology, two expressions which are distinct in this sense are said to be *separable*. This definition is actually too strong because it does not take into account distinctions between unsolvable and solvable expressions. For instance, suppose g is defined by the equation "$g = g$". Clearly, g is unsolvable and does not provide any information about its value. In most languages, g will not be separable from any other term because any context in which the insertion of g leads to a visible result must essentially ignore the inserted term, which means that the insertion of any other term also leads to exactly the same visible result. A more realistic definition is obtained if separability is weakened by not requiring the insertion of both expressions to produce visible results. Wadsworth [21] used a similar weakening of separability which he called *semi-separability* in studying the correspondence between operational and denotational distinctions in the λ-calculus. In order to give a version for equational languages, it is necessary to introduce the notion of a (term) context, which needs a few syntactic notions about terms.

Syntactically, Σ-terms are understood as trees labeled with a function symbol from Σ at each node. The symbols at leaves may also be variables. A subterm is just a subtree reached by a *path* (which we shall sometimes also call an *occurrence*). A path p is a (possibly empty) string of integers, and t/p denotes the subterm reached by p in term t. The empty path Λ reaches the term itself, the string "k" reaches the kth argument, "km" reaches the mth argument of the kth argument, etc. Given paths p and q, $p \cdot q$ denotes their concatenation. The symbols $\leqslant$ and $<$ denote the *prefix* and *proper prefix* relations on paths respectively. *Paths*(t) denotes the set of all paths that reach some subterm in t. The expression $t[p \leftarrow w]$ denotes the term obtained by replacing t/p at p by w. Paths p and q are *independent* if neither $p \leqslant q$ nor $q \leqslant p$.

A *context* is a pair (c, q) such that c is a term and q is a path in c. A context in which $q = \Lambda$ is said to be *empty*. Following traditional notation, a context will be denoted by $C[\ ]$, and the term obtained by inserting a term t in $C[\ ]$ will be written as $C[t]$. If $C[\ ] = (c, q)$ then $C[t] = c[q \leftarrow t]$.

2.2. Definition. Two ground terms t and u are said to be *separable* in $\mathcal{R}$ iff there is an extension $\mathcal{S}$ of $\mathcal{R}$ and a context $C[\ ]$ such that $C[t] =_{\mathcal{S}} v$ and $C[u] =_{\mathcal{S}} w$, for some $v, w \in T_{\Theta}$, $v \neq w$. We write $t \simeq_{\mathcal{R}} u$ to mean that t and u are *not* separable.

It is possible to show that separability is closely connected to the final object (if it exists) in the category of all models for a language if the requirement that program models must be computable is replaced by the weaker requirement that distinctions between visible terms must be respected. Indeed, the final object exists iff the relation $\simeq_{\mathcal{R}}$ is a congruence on $T_{\Sigma_{\mathcal{R}}}$ for each program $\mathcal{R}$, and the final object in this case is obtained by simply taking the quotient $T_{\Sigma_{\mathcal{R}}}/\simeq_{\mathcal{R}}$ as the model for each program. These ideas are more relevant to the semantics of abstract data type specifications, and are discussed elsewhere [12]. As it happens, such a final object does *not* exist for most equational programming languages precisely because this notion is "too strong" in the sense of the discussion above. We now define the more relevant notion.

2.3. Definition. Two ground terms t and u are said to be *semi-separable* in $\mathcal{R}$ iff there is an extension $\mathcal{S}$ of $\mathcal{R}$ and a context $C[\ ]$ such that $C[t] =_{\mathcal{S}} v$ and $C[u] \neq_{\mathcal{S}} v$, or is vice versa, for some $v \in T_\Theta$. We write $t \approx_{\mathcal{R}} u$ to mean that t and u are *not* semi-separable.

It is not difficult to prove that every language has a model in which semantic equality for each program $\mathcal{R}$ coincides with $\approx_{\mathcal{R}}$.

2.4. Lemma. *The relation* $\approx_{\mathcal{R}}$ *is a congruence on* $T_{\Sigma_{\mathcal{R}}}$.

Proof. It is obvious that $\approx_{\mathcal{R}}$ is reflexive, transitive and symmetric. To see $t_i \approx_{\mathcal{R}} u_i$, $1 \leqslant i \leqslant k$, implies $f(t_1, \ldots, t_k) \approx_{\mathcal{R}} f(u_1, \ldots, u_k)$, suppose that $C[f(t_1, \ldots, t_k)] =_{\mathcal{R}} v$ for some context $C[\ ]$. By the definition of $\approx_{\mathcal{R}}$, using the rest of the term as the context for t_i at each step, we have

$$\begin{aligned} C[f(t_1, \ldots, t_k)] &=_{\mathcal{R}} C[f(u_1, t_2, \ldots, t_k)] =_{\mathcal{R}} C[f(u_1, u_2, \ldots, t_k)] \\ &= \cdots =_{\mathcal{R}} C[f(u_1, u_2, \ldots, u_k)] =_{\mathcal{R}} v. \qquad \square \end{aligned}$$

We would like the required language model to assign to each program $\mathcal{R}$ the quotient $T_{\Sigma_{\mathcal{R}}}/\approx_{\mathcal{R}}$ as its meaning. It is obvious that the quotient satisfies $\mathcal{R}$ and is $\Sigma_{\mathcal{R}}$-reachable. It respects visible terms and is $\mathcal{R}$-computable because an *empty* context can be used to semi-separate a visible term from any term not provably equivalent to it. The model also satisfies the stable extension property.

2.5. Lemma. $t_1 \approx_{\mathcal{R}} t_2 \Rightarrow t_1 \approx_{\mathcal{S}} t_2$ *for all* $\mathcal{S} \leqslant \mathcal{R}$.

Proof. Easy given the observation that any extension of $\mathcal{S}$ is also an extension of $\mathcal{R}$ by the transitivity of $\leqslant$. Any demonstration of semi-separability in $\mathcal{S}$ is also a demonstration of semi-separability in $\mathcal{R}$. $\square$

We can now define full abstraction.

2.6. Definition. $\mathcal{A}$ is the *fully abstract* model for $(\mathfrak{R}, \leqslant, \Theta)$ iff $\mathcal{A}_{\mathcal{R}}(t_1) = \mathcal{A}_{\mathcal{R}}(t_2) \Leftrightarrow t_1 \approx_{\mathcal{R}} t_2$.

The fully abstract model is obviously unique up to isomorphism. It is interesting to compare this definition with the direct counterpart of the original definition of full abstraction [10]. We shall call a model satisfying the original definition a *Milner–Plotkin* model.

2.7. Definition. $\mathcal{A}$ is a *Milner–Plotkin* model if $\mathcal{A}_{\mathcal{R}}(t_1)=\mathcal{A}_{\mathcal{R}}(t_2) \Leftrightarrow \mathcal{A}_{\mathcal{S}}(C[t_1])=\mathcal{A}_{\mathcal{S}}(C[t_2])$ for all $\mathcal{S} \geqslant \mathcal{R}$ and all nonempty $C[\ ]$.

Full abstraction in the Milner–Plotkin sense is a rather weak property for pure equational languages. In particular, Milner–Plotkin models are not unique. It is straightforward to show that the fully abstract model for a language is a Milner–Plotkin model, subject to the following simple constraint.

2.8. Definition. We shall say that the language $(\mathfrak{R}, \preccurlyeq, \Theta)$ is *articulated* iff whenever there is a term $t \in T_{\Sigma_{\mathcal{R}}}$ and a visible term v such that $t \neq_{\mathcal{R}} v$, there is a nonempty context $C[\ ]$ which semi-separates t and v. □

An articulated language is simply one that is sufficiently fleshed out to allow the definition of a function to recognize a visible term in any program, as all real languages obviously do. The easiest context to effect the separation would normally be a *visible* context.

2.9. Theorem. *If $\mathcal{L}=(\mathfrak{R}, \preccurlyeq, \Theta)$ is articulated then the fully abstract model for $\mathcal{L}$ is a Milner–Plotkin model.*

Proof. Suppose $\mathcal{A}$ is the fully abstract model. Given Lemmas 2.4 and 2.5, it is obvious that $\mathcal{A}_{\mathcal{R}}(t_1)=\mathcal{A}_{\mathcal{R}}(t_2)$ implies $\mathcal{A}_{\mathcal{S}}(C[t_1])=\mathcal{A}_{\mathcal{S}}(C[t_2])$ for all $\mathcal{S} \geqslant \mathcal{R}$ and all $C[\ ]$. The converse, if $\mathcal{A}_{\mathcal{S}}(C[t_1])=\mathcal{A}_{\mathcal{S}}(C[t_2])$ for all $\mathcal{S} \geqslant \mathcal{R}$ and all nonempty $C[\ ]$ then $\mathcal{A}_{\mathcal{R}}(t_1)=\mathcal{A}_{\mathcal{R}}(t_2)$, follows from the definition of $\approx$ since equality in $\mathcal{A}$ coincides with $\approx$. The only contexts involved in the definition of $\approx_{\mathcal{R}}$ which are missing in the antecedent of the converse are the empty ones. However, if $t_1 =_{\mathcal{S}} v$ but $t_2 \neq_{\mathcal{S}} v$ for some $\mathcal{S} \geqslant \mathcal{R}$ and visible term v, then in an articulated language there is a *nonempty* context $C[\ ]$ and $\mathcal{S}' \geqslant \mathcal{S}$ which can be used to separate t_2 from v, and therefore also from t_1. The "missing" contexts therefore do not provide new information. □

To show that the final model is fully abstract, we need the following lemma.

2.10. Lemma. *A model $\mathcal{A}$ is final in a category of models for a given language iff for every model $\mathcal{B}$ in the category, any program $\mathcal{R}$ and any $t, u \in T_{\Sigma_{\mathcal{R}}}$, $\mathcal{B}_{\mathcal{R}}(t)=\mathcal{B}_{\mathcal{R}}(u) \Rightarrow \mathcal{A}_{\mathcal{R}}(t)=\mathcal{A}_{\mathcal{R}}(u)$.*

Proof. Easy with the observation that there is a homomorphism from $\mathscr{B}_{\mathscr{R}}$ to $\mathscr{A}_{\mathscr{R}}$ iff $\mathscr{B}_{\mathscr{R}}(t)=\mathscr{B}_{\mathscr{R}}(u) \Rightarrow \mathscr{A}_{\mathscr{R}}(t)=\mathscr{A}_{\mathscr{R}}(u)$ when both $\mathscr{B}_{\mathscr{R}}$ and $\mathscr{A}_{\mathscr{R}}$ are $\Sigma_{\mathscr{R}}$-reachable. □

2.11. Theorem. *For any language $\mathscr{L}=(\mathfrak{R}, \leqslant, \Theta)$, there is a final object $\mathscr{A}$ in the category of models for $\mathscr{L}$. Moreover, $\mathscr{A}$ is the fully abstract model for $\mathscr{L}$.*

Proof. The proof naturally proceeds by showing that the fully abstract model $\mathscr{A}$ for $\mathscr{L}$ is final in the category of models for $\mathscr{L}$. Suppose $\mathscr{B}$ is a model for $\mathscr{L}$, and $\mathscr{B}_{\mathscr{R}}(t)=\mathscr{B}_{\mathscr{R}}(u)$. We need to show that $\mathscr{A}_{\mathscr{R}}(t)=\mathscr{A}_{\mathscr{R}}(u)$. Suppose by way of contradiction that $\mathscr{A}_{\mathscr{R}}(t)\neq \mathscr{A}_{\mathscr{R}}(u)$. There is therefore a context $C[\ ]$ and an extension $\mathscr{S}$ of $\mathscr{R}$ such that $C[t]=_{\mathscr{S}} v$ and $C[u]\neq_{\mathscr{S}} v$, for some $v\in T_{\Theta}$. Since $\mathscr{B}_{\mathscr{S}}$ is computable, $\mathscr{B}_{\mathscr{S}}(C[t])=v$ and $\mathscr{B}_{\mathscr{S}}(C[u])\neq v$. Since $\mathscr{B}_{\mathscr{S}}$ (as a morphism) is homomorphic, $\mathscr{B}_{\mathscr{S}}(t)\neq \mathscr{B}_{\mathscr{S}}(u)$. This contradicts the assumption that $\mathscr{B}$ is a model since the stable extension condition in the definition of language models is violated. □

The definitions and results of this section are completely general. For instance they apply to Raoult and Vuillemin's language [16] of simple recursive equations and canonical simplification rules, with a single visible constant 0. Raoult and Vuillemin have shown that the fully abstract model for their language is *initial* in the category of models which assign least extensions of algebras satisfying the simplification rules component of each program. Theorem 2.11 shows that it can also be characterized as being *final* in a natural category of models. The concrete fully abstract model they construct uses approximate terms and approximate reduction (see next section) and as such can be seen as a proof of limiting completeness. We now turn to proving limiting completeness for a different language—the language of regular systems.

3. Limiting completeness with approximate normal forms

Limiting completeness is a language specific property, since it amounts to an assertion about the power of actual computation in a language. We would like to show that realistic equational languages have sufficient computing power to realize their final semantics. Such a proof needs a detailed and complex analysis of term rewriting in the chosen language [16, 22]. In this section we consider limiting completeness based on approximate normal forms as the partial finite operational values of expressions. We have chosen the language of regular systems [5] as a test case both because it includes as subsets many current languages and also because quite a bit is already known about its operational behavior, including a standard definition of approximate normal forms. The first part of this section is a review of the notation and terminology of term rewriting and important definitions and results about regular systems from the literature. We then construct the limiting operational model for the language and show that it is isomorphic to the final model.

3.1. Term rewriting and regular systems

We begin by reviewing some of the standard machinery of term rewriting systems, i.e., of the operational semantics of equational programs. Our notation is similar to that of [6]. The ideas of paths, subterms and contexts were introduced in Section 2. An equational program $\mathcal{R}$ is described by a countable set of *rewrite rules* which are just ordered pairs of terms with variables. The operational interpretation of $\mathcal{R}$ is embodied in the usual *reduction* relation $\rightarrow_{\mathcal{R}}$ between terms. Suppose $(l = r) \in \mathcal{R}$, $t, u \in T_{\Sigma_{\mathcal{R}}}$, and $t/p = l\alpha$ for some substitution α and some $p \in \text{Paths}(t)$. Such a path p is called a *redex occurrence*, and the set of all redex occurrences in a term t (w.r.t. $\mathcal{R}$) will be denoted by $RO_{\mathcal{R}}(t)$. We shall write $t \rightarrow^{p}_{\mathcal{R}} u$ if $u = t[p \leftarrow r\alpha]$. In general, we write $t \rightarrow_{\mathcal{R}} u$ if $t \rightarrow^{p}_{\mathcal{R}} u$ for some $p \in \text{Paths}(t)$. The reflexive transitive closure of $\rightarrow_{\mathcal{R}}$ is denoted by $\rightarrow^{*}_{\mathcal{R}}$. If $t \rightarrow^{*}_{\mathcal{R}} u$, we say that t *derives* u, and call the process a *derivation*, or an $\mathcal{R}$-derivation to be more precise; $t{\downarrow}\mathcal{R}$ denotes the set of all u such that $t \rightarrow^{*}_{\mathcal{R}} u$. When a term t contains a number of redices reached by a set U of mutually independent paths, all of them can be replaced *simultaneously*, since the order in which they are replaced is immaterial to the final result. If the term u is that result, then we write $t \rightarrow^{U}_{\mathcal{R}} u$, and call it a (single step of a) *multiderivation. All derivations in the following are assumed to be multiderivations unless otherwise stated.* We frequently need to *label* derivations, as in $A : t \rightarrow^{*}_{\mathcal{R}} u$, whereupon A stands for the derivation. If A_1 and A_2 are derivations then $A_1 \cdot A_2$ denotes their *concatenation.* The empty derivation (equivalently, a derivation of length zero) is denoted by ε. The set of *all* $\mathcal{R}$-derivations starting with the term t is denoted by $D_{\mathcal{R}}(t)$. If $A : t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n$, then $Last(A)$ denotes t_n, and $A[i, j]$ denotes the subderivation $t_i \rightarrow \cdots \rightarrow t_j$ provided $0 \leq i \leq j \leq n$. $A[0, 0]$ is simply ε. The *length* of a (multi)derivation A, denoted by $|A|$ is simply the number of steps in it, with $|\varepsilon| = 0$.

Note that we have tacitly assumed that the operational semantics of equational languages allows arbitrary rewriting, rather than restricting rewriting to replacement of leftmost-innermost or leftmost-outermost redices. Neither of these traditional evaluation strategies is adequate to implement the full logical power of equational deduction. Indeed, in general a language may not possess a simple "safe" evaluation rule of this kind at all. We are thus concerned with the most powerful operational semantics possible. The problem of efficient implementation of such a semantics has been explored elsewhere [5, 6, 14, 18].

The derivation relation $\rightarrow^{*}_{\mathcal{R}}$ is said to be *confluent* iff whenever $t \rightarrow^{*}_{\mathcal{R}} u$ and $t \rightarrow^{*}_{\mathcal{R}} v$, there is a w such that $u \rightarrow^{*}_{\mathcal{R}} w$ and $v \rightarrow^{*}_{\mathcal{R}} w$. The importance of confluence is that it is a sufficient condition for *determinism* in the operational semantics—it ensures that normal forms are unique when they exist among other things. The theory of confluent derivations is of independent interest, and it can be developed in a very abstract manner [7] with many applications.

One of the advantages of describing rewriting using equations is that it is easy to trace the fate of parts of a term as the term is repeatedly reduced. Formally, we would like to define a function which, given a path q in t and a derivation $A: t \to^*_{\mathcal{R}} u$, will give us the "addresses" of (possibly reduced) copies of t/q in u. This is the classical *residual* function and the residuals of q after A are denoted by $q\backslash A$. If $A: t \to^p_{\mathcal{R}} u$, where $t/p = l\alpha$ and $u = t[p \leftarrow r\alpha]$ for $(l, r) \in \mathcal{R}$, $q \in Paths(t)$, and $\boldsymbol{X}$ denotes the set of all variables, then $q\backslash A$ is defined by cases as follows:

(1) $q \not\geq p$: $q\backslash A = \{q\}$,

(2) $q = p \cdot w$, $w \in Paths(l)$ and $l/w \notin \boldsymbol{X}$: $q\backslash A = \emptyset$,

(3) $q = p \cdot w \cdot s$, and $l/w = x \in \boldsymbol{X}$: $q\backslash A = \{p \cdot v \cdot s \mid r/v = x\}$.

The process of residual formation traces the way pattern matching associates parts of the matched expression with variables in the (left-hand side) pattern, which are then rearranged in the result of the rewriting step according to the occurrences of the variables in the right-hand side. Strictly speaking, the notation for residuals should mention the rewriting system involved, but we shall omit this for readability; the system will usually be obvious from context. It is easy to extend this notion to more general arguments. If the first argument is a *set* of paths, we have $U\backslash A = \bigcup_{u \in U} u\backslash A$. If A is a multistep derivation, define

(1) If $A = A_1 \cdot A_2$ then $U\backslash A = (UA_1)\backslash A_2$,

(2) $U\backslash \varepsilon = U$.

If A is a multiderivation, then $U\backslash A = U\backslash B$ where B is just a version of A in which each step $A[k, k+1]$: $t_k \to^U_{\mathcal{R}} t_{k+1}$ of A is stretched out into a simple derivation reducing the redices in U one at a time.

A crucial concept which we shall often need later is that of derivations which are "free" from reductions corresponding to a set of paths in the original term in the derivation. Given a derivation

$$A: t_0 \xrightarrow[\mathcal{R}]{U_0} t_1 \xrightarrow[\mathcal{R}]{U_1} \cdots \xrightarrow[\mathcal{R}]{U_{n-1}} t_n, \quad n \geq 0,$$

and $U \subseteq Paths(t_0)$, A is said to be *U-free* iff $(U\backslash A[0, i]) \cap U_i = \emptyset$ for $0 \leq i < n$.

A regular system is a set of equations $\{l_1 = r_1, \ldots, l_n = r_n\}$ which satisfy the following conditions:

(1) *Left-linearity*: There are no duplicate occurrences of a variable in any left-hand side l_i, $1 \leq i \leq n$; i.e., all left-hand sides are *linear*.

(2) *No stray variables*: Each variable in r_i occurs in l_i, $1 \leq i \leq n$.

(3) *No ambiguity*: If a substitution unifies l_i, l_j, then it must unify r_i and r_j as well.

(4) *No overlap*: If $q \in Paths(l_i)$ and $q \neq \Lambda$ then l_i/q cannot be unified with any l_j, $1 \leq i, j \leq n$.

The rewriting relation defined by a regular system is guaranteed to be *confluent*. Indeed, the restrictions above are motivated mainly by the need to ensure this. They nevertheless yield a language that is very attractive for many applications as demonstrated by Hoffmann and O'Donnell [4]. A program is thought of as a set of rules rather than as a set of function definitions, and *any superset* of rules that is a valid program is considered a program extension. The particular visible signature used is immaterial so long as the set of visible terms is nonempty. As we shall see, all normal forms are effectively visible in a regular system. We could allow a countably infinite set of rules in a regular system without affecting the results in the following in any way. This would be useful in modeling the operational semantics of equational definitions where ambiguous equations are allowed and ambiguities are resolved by considering the equations sequentially rather than as a set, or by using the most specific equation matching the expression.

The idea of residuals can be generalized into an elegant theory of derivation spaces for regular systems [6]. We have room here to recall only the properties we actually need. Readers interested in the full treatment as well as its application to the solution of the important problem of sequential (call-by-need) computation with equations should consult the original reference.

The basic idea is that if it is possible to produce many different derivations starting with a single term, then one can speak of the residual of one such derivation with respect to another. By analogy with residuals of paths (or occurrences), the residual of A and B is denoted by $A\backslash B$. The residual in this case is itself a derivation which can be thought of as "finishing the job" of A with the result of B. Formally, if $A\colon t \to_{\mathcal{R}}^{U} u$ and $B\colon t \to_{\mathcal{R}}^{*} v$ then $A\backslash B = C\colon v \to_{\mathcal{R}}^{U\backslash B} w$. This leads to the definition of the concatenation of two derivations from the same starting term, achieving the effects of both. The concatenation is denoted by $B \sqcup A$, where $B \sqcup A = B \cdot (A\backslash B)$. We can now state a fundamental property of regular systems which underlies the pleasant properties of their derivation spaces.

Parallel Moves Lemma. *Let $A, B \in D_{\mathcal{R}}(t)$ for a regular system $\mathcal{R}$, with $|A| = |B| = 1$. Then $Last(A \sqcup B) = Last(B \sqcup A)$ and for each $p \in Paths(t)\, p\backslash(A \sqcup B) = p\backslash(B \sqcup A)$.*

This lemma, which is a strong form of confluence, permits a generalization of the residual relation to arbitrary derivations. For $A, B \in D_{\mathcal{R}}(t)$, with $|B| = 1$, define $A\backslash B \in D_{\mathcal{R}}(Last(B))$ by induction on $|A|$.

(1) $\varepsilon\backslash B = \varepsilon$,

(2) $(A_1 \cdot A_2)\backslash B = (A_1\backslash B)(A_2\backslash(B\backslash A_1))$, with $|A_1| = 1$.

Note that $A_2\backslash(B\backslash A_1)$ is defined by induction, since $|B\backslash A_1| = 1$, and the former can be concatenated after $A_1\backslash B$ by the preceding lemma. Now this can be generalized to arbitrary derivations $A, B \in D_{\mathcal{R}}(t)$. Define $A\backslash B \in D_{\mathcal{R}}(Last(B))$ by induction on $|B|$.

(1) $A\backslash\varepsilon = A$,

(2) $A\backslash(B_1 \cdot B_2) = (A\backslash B_1)\backslash B_2$, with $|B_1| = 1$.

The definition of $\sqcup$ and the parallel moves lemma generalize to derivations of arbitrary length in the natural way. For $A, B \in D_{\mathcal{R}}(t)$, define

- $A \equiv B \Leftrightarrow \forall C \in D_{\mathcal{R}}(t).\ C\backslash A = C\backslash B$,
- $A \sqsubseteq B \Leftrightarrow A \sqcup B \equiv B$.

We now state some properties of the derivation space assumed in later proofs.

For any regular systems $\mathscr{R}$, and for all A, B, $C \in D_{\mathscr{R}}(t)$
(1) $C\backslash(A \sqcup B) = C\backslash(B \sqcup A)$,
(2) $A \sqcup (B \sqcup C) = (A \sqcup B) \sqcup C$,
(3) $A \sqcup B \equiv B \sqcup A$,
(4) $A \equiv B \Rightarrow Last(A) = Last(B)$,
(5) $|A\backslash B| = |A|$,
(6) $A \sqsubseteq B \Leftrightarrow A \cdot E \equiv B$ for some E.

3.2. *Approximate normal forms and limiting completeness*

It is known that nontermination is not the same as meaninglessness in computation based on rewriting. Wadsworth [21] made this point strikingly for the λ-calculus by showing that there is a nonterminating λ-expression which is semantically and operationally equivalent to the identity function ($\lambda x.x$). Lazy evaluation allows nonterminating expressions to be used in equational computation. For instance, the equation

$$inf = cons(1, inf)$$

can be taken to define the infinite list of 1's. One problem with expressions like *inf* is that they never manifest their entire result in a finite normal form, which has traditionally been considered the operational counterpart of denotational value. In a sense their value resides in the entire range of their visible uses, but it would be desirable to establish the correspondence in a more direct way. Approximate terms, originally introduced by Wadsworth [21], are now commonly used for this purpose [1, 9, 16]. The key intuition here is an exact counterpart of the intuition underlying Scott's theory of continuous domains [17] that infinite values in computing are limits of directed sets of finite approximations. To make this precise, one needs a way to fix the revealed partial value in the results of finite rewriting. Wadsworth used the notion of a direct approximant obtained by replacing the unevaluated parts of an expression by a new constant Ω. The successive results of rewriting *inf* are then Ω, $cons(1, \Omega)$, $cons(1\ cons(1, \Omega))$, etc. Assuming that Ω is the least element in a partial order on terms, this directed set (actually a chain) will clearly produce the entire infinite list as its limit. Of course terms containing Ω need not be in normal form. The extension of rewriting to terms with Ω is called *approximate reduction.*

In the λ-calculus, one simply replaces all redices by Ω to obtain the direct approximant, which is the approximate normal form in the sense that it shows the outer (head) structure of any eventual normal form. The corresponding construction in equational languages is a bit trickier. Since redices are program specific, so are approximate normal forms. It is convenient to formulate an *approximate normal form function* $\omega_{\mathscr{R}}$ for a given program $\mathscr{R}$. Suppose we assume that

- Ω is a new symbol distinct from all symbols used in programs.
- $L_{\mathscr{R}}$ is the set of left-hand sides of equations in $\mathscr{R}$ where each variable has been replaced by Ω.

- The partial order $\leqslant$ on terms is generated by monotonically extending the relation $\forall t \in T_\Sigma.\ \Omega \leqslant t$ over all function symbols, i.e.,

$$f(t_1, \ldots, t_k) \leqslant f(u_1, \ldots, u_k) \Leftrightarrow t_i \leqslant u_i, \quad 1 \leqslant i \leqslant k.$$

- The relation $t \uparrow u$ (t is *compatible* with u) is equivalent to $\exists v.\ t \leqslant v$ and $u \leqslant v$.

Here is the approximate normal form function defined by Huet and Levy [6] for a regular system.

Let $t = f(t_1, \ldots, t_k)$, $k \geqslant 0$

$$\bar{\omega}_{\mathcal{R}}(t) = f(\omega_{\mathcal{R}}(t_1), \ldots, \omega_{\mathcal{R}}(t_k)), \qquad \omega_{\mathcal{R}}(t) = \begin{cases} \Omega, & \text{if } \bar{\omega}_{\mathcal{R}}(t) \uparrow L_{\mathcal{R}} \\ \bar{\omega}_{\mathcal{R}}(t), & \text{otherwise,} \end{cases}$$

where $t \uparrow L_{\mathcal{R}}$ if there is a $u \in L_{\mathcal{R}}$ such that $t \uparrow u$. The idea of $\omega_{\mathcal{R}}$ is to replace with Ω all redices and also all those subterms which may eventually become redices after some rewriting, leaving the maximal prefix (in the $\leqslant$ order on terms) which is guaranteed to be a prefix of any eventual normal form. The definition of $\omega_{\mathcal{R}}$ is obviously completely general for equational languages. The only qualification is that it is a little conservative in deciding on the "maximal" prefix because it takes into account only the left-hand sides of the equations concerned. This is in line with Wadsworth's original definition. The following three properties of $\omega_{\mathcal{R}}$ are easy to prove [6]:

(1) (*Idempotence*) $\omega_{\mathcal{R}}(\omega_{\mathcal{R}}(t)) = \omega_{\mathcal{R}}(t)$,

(2) (*Monotonicity*) $x \leqslant y \Rightarrow \omega_{\mathcal{R}}(x) \leqslant \omega_{\mathcal{R}}(y)$,

(3) (*Increase with rewriting*) $x \rightarrow^{*}_{\mathcal{R}} y \Rightarrow \omega_{\mathcal{R}}(x) \leqslant \omega_{\mathcal{R}}(y)$.

Limiting completeness asserts that the limiting operational model is the same as the intended denotational model. The actually computed *naive* result of an expression t in a program $\mathcal{R}$ is simply $t{\downarrow}\mathcal{R}$, which is already a little abstract since it takes into account all possible rewriting sequences; the effect is usually achieved by a single *safe* sequence (see [6, 18] for results on safe sequential computation in regular systems). The limiting model is obtained from the naive operational model by using approximants to eliminate incidental and superficial differences. For instance, the two equations in

$$\mathcal{R} = \{a = cons(1, a), b = cons(1, b)\}$$

obviously define the same infinite list, but the sets $a{\downarrow}\mathcal{R}$ and $b{\downarrow}\mathcal{R}$ are disjoint. However, we have

$$\omega_{\mathcal{R}}(a{\downarrow}\mathcal{R}) = \omega_{\mathcal{R}}(b{\downarrow}\mathcal{R}) = \{\Omega, cons(1, \Omega), \ldots\}$$

thus revealing their sameness. There are cases where one more refinement is required. For instance, with

$$\mathcal{R} = \{a = cons(1, a), b = cons(1, cons(1, b))\},$$

we have $\omega_{\mathcal{R}}(a{\downarrow}\mathcal{R}) \not\subseteq \omega_{\mathcal{R}}(b{\downarrow}\mathcal{R})$ although the two lists are again intuitively identical. The difference this time is that b skips the computation of certain approximations (those with an odd number of 1's) to the final value. The straightforward way to smooth out this wrinkle is to require that the sets of approximants used as values in the limiting model be *downward complete*. Formally, let $T_{\mathscr{R}}$ denote $\omega_{\mathscr{R}}(T_{\Sigma_{\mathscr{R}}})$. For any subset $V \subseteq T_{\mathscr{R}}$, the downward completion of V, denoted by $\bar{V}$, is defined as

$$\bar{V} = \{u \in T_{\mathscr{R}} \mid \exists v \in V.\ u \leq v\}.$$

Observe that the set $\omega_{\mathscr{R}}(t{\downarrow}\mathscr{R})$ is always *directed* since the regular system $\mathscr{R}$ is confluent and $\omega_{\mathscr{R}}$ increases with rewriting. A natural set of values for the limiting model of $\mathscr{R}$ is therefore the (many-sorted) set of all (finite and infinite) downward complete directed subsets of $T_{\mathscr{R}}$. It can be shown that such values are *ideals*, and for each sort they form a domain in Scott's sense [17]. We defer this aspect until Section 4, where a domain-based model is constructed using information systems.

The discussion above leads naturally to a straightforward construction for the limiting operational model for regular systems. Defining $\mathcal{M}_{\mathscr{R}}(t) = \omega_{\mathscr{R}}(t{\downarrow}\mathscr{R})$, we would like $\mathcal{M}_{\mathscr{R}}$ to be the unique homomorphism from $T_{\Sigma_{\mathscr{R}}}$ to the algebra (also called $\mathcal{M}_{\mathscr{R}}$ as usual) assigned to $\mathscr{R}$ in the limiting model $\mathcal{M}$. We therefore define the (putative) algebra $\mathcal{M}_{\mathscr{R}}$ by letting the carrier for each sort be the sets of values $\mathcal{M}_{\mathscr{R}}(t)$ for all terms t of that sort, and define the denotation for each $f \in \Sigma_{\mathscr{R}}$ as the function $f_{\mathscr{R}}$ where

$$f_{\mathscr{R}}(x_1, \ldots, x_k) = \overline{\{y \in \mathcal{M}_{\mathscr{R}}(f(y_1, \ldots, y_k)) \mid y_i \in x_i\}}$$

assuming of course that $x_1, \ldots, x_k$ are values of the right sort. For constants $c \in \Sigma_{\mathscr{R}}$, $c_{\mathscr{R}}$ reduces to the denotation $\mathcal{M}_{\mathscr{R}}(c)$ of c as an expression. To show that $\mathcal{M}_{\mathscr{R}}$ is a proper $\Sigma_{\mathscr{R}}$-algebra, all one needs to do is to show that there is a homomorphism from $T_{\Sigma_{\mathscr{R}}}$ to $\mathcal{M}_{\mathscr{R}}$, i.e., to show that

$$\mathcal{M}_{\mathscr{R}}(f(t_1, \ldots, t_k)) = f_{\mathscr{R}}(\mathcal{M}_{\mathscr{R}}(t_1), \ldots, \mathcal{M}_{\mathscr{R}}(t_k)).$$

This coherence condition is actually quite difficult to prove, and in order to avoid obscuring the structure of the limiting completeness proof, the proof of the following continuity lemma is deferred to the next section. The intuition underlying the lemma is that any finite computation can be done in a "call-by-value" fashion *if* one knows ahead of time just how far to go in evaluating the arguments before evaluating the function applied to them. In the terminology of information systems (see Section 4.1), it asserts that computation in regular systems can be seen *operationally* as the application of functions defined by *approximable maps*. This is commonly assumed to be a property of computation. Theorem 5.4 of Wadsworth [21] proves a similar syntactic property for the λ-β-calculus.

Continuity Lemma. *For each* $x \in \mathcal{M}_{\mathscr{R}}(f(t_1, \ldots, t_k))$ *there are* $y_1 \in \mathcal{M}_{\mathscr{R}}(t_1), \ldots, y_k \in \mathcal{M}_{\mathscr{R}}(t_k)$ *such that* $x \in \mathcal{M}_{\mathscr{R}}(f(y_1, \ldots, y_k))$.

We also often need the following pleasant consequence of equational computation.

Monotonicity Lemma. *If* $t \leq u$ *and* $t \rightarrow_{\mathscr{R}}^{U} v$, *then* $u \rightarrow_{\mathscr{R}}^{U} w$ *and* $v \leq w$.

Proof. Easy by structural induction on t. □

3.1. Lemma. *Each* $\mathcal{M}_{\mathscr{R}}$ *is a* $\Sigma_{\mathscr{R}}$-*reachable* $\Sigma_{\mathscr{R}}$-*algebra.*

Proof. By the Continuity Lemma, we have

$$\mathcal{M}_{\mathcal{R}}(f(t_1, \ldots, t_k)) \subseteq f_{\mathcal{R}}(\mathcal{M}_{\mathcal{R}}(t_1), \ldots, \mathcal{M}_{\mathcal{R}}(t_k)).$$

The converse,

$$\mathcal{M}_{\mathcal{R}}(f(t_1, \ldots, t_k)) \supseteq f_{\mathcal{R}}(\mathcal{M}_{\mathcal{R}}(t_1), \ldots, \mathcal{M}_{\mathcal{R}}(t_k)),$$

follows easily by the Monotonicity Lemma. The reachability is obvious from the definition of the carriers in $\mathcal{M}_{\mathcal{R}}$. □

We have not yet shown that $\mathcal{M}$ is a model for the language of regular systems in the sense of the definition in Section 2. It is easy to show that each $\mathcal{M}_{\mathcal{R}}$ is a model for $\mathcal{R}$.

3.2. Lemma. $\forall t, u \in T_{\Sigma_{\mathcal{R}}}.\ t =_{\mathcal{R}} u \Rightarrow \mathcal{M}_{\mathcal{R}}(t) = \mathcal{M}_{\mathcal{R}}(u)$, *i.e.*, $\mathcal{M}_{\mathcal{R}}$ *satisfies* $\mathcal{R}$.

Proof. If $t =_{\mathcal{R}} u$ then by the confluence of $\mathcal{R}$ there is a v such that $t \to_{\mathcal{R}}^* v$ and $u \to_{\mathcal{R}}^* v$ (see Lemma 2.1 in [7] for a proof). Now suppose $x \in \mathcal{M}_{\mathcal{R}}(t)$. There must be a w such that $t \to_{\mathcal{R}}^* w$ and $x \leqslant \omega_{\mathcal{R}}(w)$. Since $t \to_{\mathcal{R}}^* v$, by the confluence of $\mathcal{R}$, there must be a z such that $v \to_{\mathcal{R}}^* z$ and $w \to_{\mathcal{R}}^* z$. Since $\omega_{\mathcal{R}}$ increases with derivation, $\omega_{\mathcal{R}}(z) \geqslant \omega_{\mathcal{R}}(w)$ and $x \leqslant \omega_{\mathcal{R}}(z)$. Therefore, since $u \to_{\mathcal{R}}^* v \to_{\mathcal{R}}^* z$, $x \in \mathcal{M}_{\mathcal{R}}(u)$. This shows that $\mathcal{M}_{\mathcal{R}}(t) \subseteq \mathcal{M}_{\mathcal{R}}(u)$ and also vice versa since the choice of t was arbitrary. □

3.3. Lemma. $\forall v \in T_{\Theta}.\ \mathcal{M}_{\mathcal{R}}(t) = \mathcal{M}_{\mathcal{R}}(v) \Rightarrow t \to_{\mathcal{R}}^* v$, *i.e.*, $\mathcal{M}_{\mathcal{R}}$ *is* $\mathcal{R}$*-computable.*

Proof. Since visible terms are normal forms, $\mathcal{M}_{\mathcal{R}}(v) = \overline{\{v\}}$. Therefore $\mathcal{M}_{\mathcal{R}}(t) = \mathcal{M}_{\mathcal{R}}(v)$ implies $v \in \mathcal{M}_{\mathcal{R}}(t)$ which is possible only if $t \to_{\mathcal{R}}^* v$. □

This result holds for *all* normal forms, not just for visible terms, so all normal forms are in a sense visible in a regular system. Indeed, one could generalize the idea of visible terms to include all those terms which are their own approximants in a limiting complete model. In this sense, constructor terms can be shown to be the visible terms in languages like Miranda (see Section 4.1).

Proving the stable extension condition for $\mathcal{M}$ is much more difficult and requires a detailed analysis of the operational behavior of regular systems along the same lines as the Continuity Lemma. We therefore defer the proof of the following lemma to the next section.

Stable Extension Lemma. *For all regular systems* $\mathcal{R}$, $\mathcal{S}$, $\mathcal{R} \subseteq \mathcal{S}$ *and* $\mathcal{M}_{\mathcal{R}}(t) = \mathcal{M}_{\mathcal{R}}(u) \Rightarrow \mathcal{M}_{\mathcal{S}}(t) = \mathcal{M}_{\mathcal{S}}(u)$.

To finish the proof of limiting completeness, $\mathcal{M}$ must be shown to be *final* in the category of models for the language of regular systems. One way to prove this is to show that the equality congruence induced by $\mathcal{M}_{\mathcal{R}}$ includes $\approx_{\mathcal{R}}$, since $\approx_{\mathcal{R}}$ is known to be the congruence in the final model. This requires the following result, which asserts that for each approximant there is an "observer" which recognizes exactly the information represented by that approximant. The idea is similar to the defining property of "articulated languages" in Section 2.

Recognition Lemma. *For each $\mathcal{R} \in \mathfrak{R}$ and each $x \in \omega_{\mathcal{R}}(T_{\Sigma_{\mathcal{R}}})$, there is an extension $\mathcal{S} \geqslant \mathcal{R}$, a context $C[\ ]$ and a visible term v such that for any $t \in T_{\Sigma_{\mathcal{R}}}$, $C[t] =_{\mathcal{S}} v \Leftrightarrow x \in \mathcal{M}_{\mathcal{R}}(t)$.*

Proof. Let y be the term obtained by replacing each Ω in x by a distinct new variable. Let $\mathcal{S} = \{g(y) = v\} \cup \mathcal{R}$, where g is a new unary symbol and $v \in T_{\Theta}$. This is a regular system because all conditions except the no overlap condition are obviously satisfied and since x is not compatible with any existing left-hand side, the no-overlap condition is not violated. We need to show that $g(t) \to^*_{\mathcal{S}} v$ iff $x \in \mathcal{M}_{\mathcal{R}}(t)$. The if part follows easily by the Monotonicity Lemma. For the only if part, observe that the last step for any derivation $g(t) \to^*_{\mathcal{S}} v$ must be of the form $g(u) \to^{\Lambda}_{\mathcal{R}} v$ where $t \to^*_{\mathcal{R}} u$ and $u \geqslant x$. The rest follows easily from the fact that $\omega_{\mathcal{R}}$ is monotonic ($u \geqslant x \Rightarrow \omega_{\mathcal{R}}(u) \geqslant \omega_{\mathcal{R}}(x)$) and idempotent ($\omega_{\mathcal{R}}(x) = x$). $\square$

3.4. Lemma. $t \approx_{\mathcal{R}} u \Rightarrow \mathcal{M}_{\mathcal{R}}(t) = \mathcal{M}_{\mathcal{R}}(u)$.

Proof. Easier in contrapositive form. Suppose $\mathcal{M}_{\mathcal{R}}(t) \neq \mathcal{M}_{\mathcal{R}}(u)$. Suppose without loss of generality that $x \in \mathcal{M}_{\mathcal{R}}(t)$ but $x \notin \mathcal{M}_{\mathcal{R}}(u)$. The Recognition Lemma implies that t and u are semi-separable. $\square$

These results are summarized in the following theorem.

3.5. Theorem. *The limiting operational model ($\mathcal{M}$) based on approximate normal forms is isomorphic to the final model for the language of regular systems.*

Proof. Immediate by preceding lemmas and Lemma 2.10. $\square$

3.3. Proof of key lemmas

There are intuitively two reasons why regular systems satisfy the Continuity and Stable Extension Lemmas. The first is that if a derivation A does not reduce any redex in a set U of redex occurrences in the initial term t (A is U-free), then the approximate normal form of t with respect to U contains all the information about t required and revealed by A. The second reason is the property called the Parallel Moves Lemma in Section 3.1, which allows permutations of derivations. Among the technical machinery required to prove the first property, we need to formulate an alternative definition for $\omega_{\mathcal{R}}$ to allow us to speak of the approximate normal form of a term with respect to only a *subset* of the redices. This notion is captured in the function $\sigma_{\mathcal{R}}$.

$$Q_{\mathcal{R}}(t) = \{q \in Paths(t) \mid t/q \not\geqslant L_{\mathcal{R}},\ t/q \uparrow L_{\mathcal{R}} \text{ and } t/q \neq \Omega\},$$

$$\tau_{\mathcal{R}}(t) = \text{if } Q_{\mathcal{R}}(t) = \emptyset \text{ then } t \text{ else } \tau_{\mathcal{R}}(t[Q_{\mathcal{R}}(t) \leftarrow \Omega]),$$

$$\sigma_{\mathcal{R}}(t, U) = \tau_{\mathcal{R}}(t)(t[U \leftarrow \Omega]),$$

$$\omega_{\mathcal{R}}(t) = \sigma_{\mathcal{R}}(t, RO_{\mathcal{R}}(t)).$$

The property to be proved can now be expressed graphically as follows

$$\begin{tikzcd}
A\colon t \arrow[r, "\overset{(U\text{-free})}{*}", "\mathscr{R}"'] \arrow[d, Rightarrow] & u \arrow[r, phantom, "\Rightarrow"] \arrow[d, Rightarrow] & \omega_{\mathscr{R}}(u) \arrow[d, equal] \\
t[V \leftarrow \Omega] \arrow[r, "*", "\mathscr{R}"'] \arrow[d, equal] & u[V \backslash A \leftarrow \Omega] = w \arrow[r, phantom, "\Rightarrow"] \arrow[d, equal] & \omega_{\mathscr{R}}(w) \\
\sigma_{\mathscr{R}}(t, U) \arrow[r, "*", "\mathscr{R}"'] & \sigma_{\mathscr{R}}(u, U \backslash A) &
\end{tikzcd}$$

Lemma 3.10 below proves most of the relationships in this diagram. The proof relies on the fact that subterms which are compatible with redex patterns without being redices themselves (which are replaced by $\sigma_{\mathscr{R}}$ with Ω) are essentially useless in derivations besides not contributing to the approximate normal form of the result of the derivation. This is shown in Proposition 3.8, which requires some auxiliary results. Let $SUB_{\mathscr{R}} = \{t/p \mid t \in L_{\mathscr{R}}$ and $p \in Paths(t)\}$—$SUB_{\mathscr{R}}$ is the set of all parts of redex patterns in Ω-term form.

3.6. Proposition. *If $t \uparrow v$ and $t \not\geq v$ for some $v \in SUB_{\mathscr{R}}$, then, for any derivation $A \in D_{\mathscr{R}}(t)$,*

(1) *$Last(A) \not\geq L_{\mathscr{R}}$,*

(2) *$Last(A) \uparrow v$ and $Last(A) \not\geq v$.*

Proof. By induction on the structure of t. The basis, $t \in \Sigma_0$ or $t = \Omega$ is trivial. Assume that the proposition holds for all proper subterms of $t = f(t_1, \ldots, t_k)$, and consider some $v = f(v_1, \ldots, v_k)$ which satisfies the antecedents of the proposition. Suppose for the sake of contradiction that derivations which contradict (1) exist, and let $t \rightarrow^*_{\mathscr{R}} u$ be a shortest derivation such that $u \geq L_{\mathscr{R}}$. Since it is a shortest derivation, we must have $u = f(u_1, \ldots, u_k)$, and $t_i \rightarrow^*_{\mathscr{R}} u_i$, $1 \leq i \leq k$. Suppose $u \geq c \in L_{\mathscr{R}}$. For each t_i, there are two possible cases:

Case 1: $t_i \geq v_i$. By the nonoverlap condition in the definition of regular systems, $\omega(v_i) = v_i$, and by the monotonicity of ω, $\omega(t_i) \geq v_i$. Since ω increases with derivation, we have $\omega(u_i) \geq v_i$ and hence $u_i \geq v_i$.

Case 2: $t_i \not\geq v_i$. This implies $v_i \neq \Omega$ and therefore $v_i \in SUB_{\mathscr{R}}$. Since $t_i \uparrow v_i$, we have $u_i \uparrow v_i$ and $u_i \not\geq v_i$ by the inductive assumption.

In each of these two cases, $u_i \uparrow v_i$ for each i, and hence $u \uparrow v$. However, $u \geq c$, therefore $v \uparrow c \in L_{\mathscr{R}}$. The only way to reconcile this with the nonoverlap condition is if $v = c$, which gives $u_i \geq v_i$, $1 \leq i \leq k$. Case 2 is therefore irrelevant, but then $t_i \geq v_i$, $1 \leq i \leq k$ and $t \geq v$ thus contradicting the antecedent of the proposition. This completes the proof of assertion (1). Assertion (2) follows from this argument also since we have shown that $u \uparrow v$ and $u \not\geq v$ in any derivation from t that does not reduce a redex at Λ, which is now seen to include *all* derivations, given (1). $\square$

3.7. Corollary. *If $t \uparrow v$ and $t \not\geq v$ for some $v \in L_{\mathcal{R}}$ then $\forall A \in D_{\mathcal{R}}(t)$, $Last(A) \not\geq SUB_{\mathcal{R}}$.*

Proof. Consider an arbitrary u such that $t \to^*_{\mathcal{R}} u$. By Proposition 3.6, we have $u \uparrow v$ and $u \not\geq v$ since $L_{\mathcal{R}} \subseteq SUB_{\mathcal{R}}$. The rest follows from the non-overlap condition for regular systems. □

3.8. Proposition. *Suppose $z = t/q$ and $z \uparrow v$ and $z \not\geq v$ for some $v \in L_{\mathcal{R}}$. Then, for every A: $t \to^*_{\mathcal{R}} u$, there is an A_q: $t_q \to^*_{\mathcal{R}} u_q$ such that*

(1) *$t_q = t[q \leftarrow \Omega]$ and $u_q = u[U \leftarrow \Omega]$ where $U = q \backslash A$.*

(2) *$Paths(t_q) \backslash A = Paths(t_q) \backslash A_q$.*

Proof. By induction on $|A|$. The basis case is trivial: if $A = \varepsilon$, then $A_q = \varepsilon$. Suppose $A = A_1 \cdot A_2$, A_1: $t \to^*_{\mathcal{R}} u_1$, A_2: $u_1 \to^r_{\mathcal{R}} u$ and $|A_1| = n$. By the inductive hypothesis, there is a u_{q_1}, such that $U_1 = q \backslash A_1$ and $u_{q_1} = u_1[U_1 \leftarrow \Omega]$. For each $p \in U_1$, $z \to^*_{\mathcal{R}} u_1/p$, and therefore, by Proposition 3.6 and Corollary 3.7 above, $u_1/p \uparrow v$ and $u_1/p \not\geq SUB_{\mathcal{R}}$; which means, among other things, that $r \in U_1$ is ruled out. Now define A_{q_2}: $u_{q_1} \to^*_{\mathcal{R}} u_q$ based on the possible relationship between r and U_1.

Case 1: $r > p \in U_1$: In this case, $A_{q_2} = \varepsilon$ and $U = U_1$.

Case 2: $r < p \in U_1$: Since $u_1 \backslash p \not\geq SUB_{\mathcal{R}}$ for any $p \in U_1$, A_{q_2}: $u_{q_1} \to^r_{\mathcal{R}} u_q$, and u_q will satisfy the required relationship with u.

Case 3: $r \not> U_1$ and $r \not< U_1$: In this case also, A_{q_2}: $u_{q_1} \to^r_{\mathcal{R}} u_q$, and moreover, $U = U_1$.

In all three cases, $Paths(t_q) \backslash A_q = Paths(t_q) \backslash A$ given that $Paths(t_q) \backslash A_{q_1} = Paths(t_q) \backslash A_1$ by the inductive hypothesis. □

Proposition 3.9 makes the similar point that a redex that is not reduced during a derivation also does not contribute anything and could be replaced by Ω at the start of the derivation without effect.

3.9. Proposition. *Suppose $z = t/q \geq L_{\mathcal{R}}$ and A: $t \to^*_{\mathcal{R}} u$ is $\{q\}$-free. Then there is a corresponding A_q: $t_q \to^*_{\mathcal{R}} u_q$ such that $u_q = u[U \leftarrow \Omega]$, where $t_q = t[q \leftarrow \Omega]$ and $U = q \backslash A$.*

Proof. Similar to Proposition 3.8 except that $r \in U_1$ is ruled out by the explicit assumption that A is $\{q\}$-free, and $t_1/p \not\geq SUB_{\mathcal{R}}$ for any $p \in U_1$ by the nonoverlap condition. □

We can now state and prove the main properties of the partial approximate normal forms produced by $\sigma_{\mathcal{R}}$, which were diagrammed above.

3.10. Lemma. *Suppose $U \subseteq RO_{\mathcal{R}}(t)$, $v = \sigma(t, U) = t[V \leftarrow \Omega]$, and A: $t \to^*_{\mathcal{R}} u$ is U-free. Then there is a derivation B: $v \to^*_{\mathcal{R}} w$ such that*

(1) *$w = u[W \leftarrow \Omega]$ where $W = V \backslash A$;*

(2) *$\omega_{\mathcal{R}}(u) = \omega_{\mathcal{R}}(w)$.*

Proof. The proof of both parts is based on a fairly easy recursion induction on the definition of τ, i.e., an induction on the depth of the tail recursion required to obtain $\sigma(t, U)$ from $t[U \leftarrow \Omega]$. Consider the sequence $t_0, t_1, \ldots, t_n$ where $t_0 = t[U \leftarrow \Omega]$, $t_{i+1} = t_i[Q(t_i) \leftarrow \Omega]$, and $t_n = \tau(t_0) = v$, where n is the smallest number such that $t_n = t_{n+1}$. Define U_i to be the set of paths such that $t_i = t_{i-1}[U_i \leftarrow \Omega]$, and $U_0 = U$. By Proposition 3.9, there is a derivation A_0: $t_0 \rightarrow^*_{\mathscr{R}} w_0$ such that $w_0 = u[W_0 \leftarrow \Omega]$, $W_0 = U_0 \backslash A$. By Proposition 3.8 there are derivations A_i: $t_i \rightarrow^*_{\mathscr{R}} w_i$, $1 \leqslant i \leqslant n$, such that $w_i = w_{i-1}[W_i \leftarrow \Omega]$, $W_i = U_i \backslash A_{i-1}$. Let V_i be sets of paths such that $t_i = t[V_i \leftarrow \Omega]$, $0 \leqslant i \leqslant n$. Clearly, $V_0 = U_0 = U$ and $V_n = V$. It is easy to see that $w_i = u[X_i \leftarrow \Omega]$ where $X_i = V_i \backslash A$, since V_i is simply the union of $U_0, \ldots, U_i$ except for the elimination of those paths which have a prefix in the union, since by (2) in Proposition 3.8, $U_i \backslash A_{i-1} = U_i \backslash A$, $0 < i \leqslant n$. We thus have $V_n = V$, $w_n = w$, $X_n = W$ and $A_n = B$.

For part (2) we need to show that $\omega_{\mathscr{R}}(u) = \omega_{\mathscr{R}}(w_i)$ for $1 \leqslant i \leqslant n$, of which $\omega_{\mathscr{R}}(u) = \omega_{\mathscr{R}}(w_0)$ follows from the fact that $w_0 = u[U \backslash A \leftarrow \Omega]$ and $U \backslash A \subseteq RO_{\mathscr{R}}(u)$. To see that $\omega_{\mathscr{R}}(w_i) = \omega_{\mathscr{R}}(w_{i-1})$, note that $t_i = t_{i-1}[U_i \leftarrow \Omega]$, $w_i = w_{i-1}[U_i \backslash A_{i-1} \leftarrow \Omega]$, and A_i: $t_i \rightarrow^*_{\mathscr{R}} w_i$, $0 < i \leqslant n$. For each $q \in U_i$, $t_{i-1}/q \uparrow L_{\mathscr{R}}$ and $t_{i-1}/q \not\geqslant L_{\mathscr{R}}$ by the definition of Q. Therefore, by part (2) of Proposition 3.6, for each $p \in U_i \backslash A_{i-1}$, $w_{i-1}/p \uparrow L_{\mathscr{R}}$. The rest follows from the definition of $\omega_{\mathscr{R}}$. $\square$

It is not hard to see how this property can be combined with the permutations permitted by the Parallel Moves Lemma to derive Stable Extension Lemma. That Lemma in effect states that each finite element in the value of an expression t in an extension $\mathscr{S} \geqslant \mathscr{R}$ is implied by a finite element of its value in the original program $\mathscr{R}$ (see Lemma 3.13 below). This can be proved by showing that all the steps in an $\mathscr{S}$-derivation for t which can be performed within the smaller program $\mathscr{R}$ can be *brought forward* into an initial subderivation, and since the $\mathscr{R}$-redices in the result (say u) of this subderivation are not used in the rest of the derivation, one can mimic the rest of the derivation with the approximate normal form $\omega_{\mathscr{R}}(u) = \sigma_{\mathscr{S}}(u, RO_{\mathscr{R}}(u))$. The permutation part is proved in Lemma 3.12. The basic permutation step needed in Lemma 3.12 as well as the similar Lemma 3.14 for continuity is given in Proposition 3.11.

3.11. Proposition. *Suppose U, V is a partition of $Paths(t)$, $A \in D_{\mathscr{R}}(t)$ is not U-free, and $A[0, k]$ is the longest U-free prefix of A. Then there is a $C = C_1 \cdot C_2 \sqsupseteq A$ such that C_1: $t \rightarrow^{U_0}_{\mathscr{R}} t'$ for some $U_0 \subseteq U$, $C_2[0, k+1]$ is $U \backslash C_1$-free, and $|C_2| = |A|$.*

Proof. Suppose A: $t_0 \rightarrow^{W_1}_{\mathscr{R}} t_1 \rightarrow^{W_2}_{\mathscr{R}} \cdots \rightarrow^{W_n}_{\mathscr{R}} t_n$, where $t = t_0$ and $n > 0$. Consider $t_k \rightarrow^{W_{k+1}}_{\mathscr{R}} t_{k+1}$. Clearly,

$$W_{k+1} \cap U \backslash A[0, k] = X \neq \emptyset.$$

Let $U_0 \subseteq U$ be the smallest subset of U such that $X \subseteq U_0 \backslash A[0, k]$. Let C_1: $t \rightarrow^{U_0}_{\mathscr{R}} t'$ and $C_2 = A \backslash C_1$. We know that $|C_2| = |A|$. Let

$$C_2\colon t'_0 \rightarrow^{W'_1}_{\mathscr{R}} t'_1 \rightarrow \cdots \rightarrow^{W'_n}_{\mathscr{R}} t'_n, \quad \text{where } t'_0 = t'.$$

To see that $C_2[0, k+1]$ is $U\backslash C_1$-free, it suffices to observe that $W'_{k+1} \cap (U\backslash C_1 \cdot C_2[0, k]) = \emptyset$. □

3.12. Lemma. *Suppose A: $t \rightarrow^*_{\mathcal{S}} u$ and $\mathcal{R} \subseteq \mathcal{S}$. Then there is a $B = B_1 \cdot B_2 \sqsupseteq A$ such that B_1: $t \rightarrow^*_{\mathcal{R}} v$ and B_2: $v \rightarrow^*_{\mathcal{S}} w$ is $RO_{\mathcal{R}}(v)$-free.*

Proof. Let $U = RO_{\mathcal{R}}(t)$. The proof is by induction on m where $m = |A| - k$ and k is the length of the longest U-free prefix of A. The basis $m = 0$ is trivial. If $m > 0$ then A is not U-free, and by Proposition 3.11, there is a derivation $C = C_1 \cdot C_2 \sqsupseteq A$ such that C_1: $t \rightarrow^{U_0}_{\mathcal{R}} t'$ for some $U_0 \subseteq U$, $C_2[0, k+1]$ is $U\backslash C_1$-free and $|C_2| = |A|$. Let $U' = RO_{\mathcal{R}}(t')$. Clearly, $C_2[0, k+1]$ is U'-free since the redices in U' are either those in U or created by the reduction of U_0. The former are not reduced in $C_2[0, k+1]$ by Proposition 3.11 as mentioned above and the latter were not even present in $A[0, k+1]$, hence cannot be present in its residue. Hence the value of m for C_2 is less than for A and by the induction hypothesis applied to C_2 there is a derivation $E = E_1 \cdot E_2 \in D_{\mathcal{R}}(t')$ such that $E \sqsupseteq C_2$, E_1: $t' \rightarrow^*_{\mathcal{R}} v$ and E_2 is $RO_{\mathcal{R}}(v)$-free. Therefore $B_1 = C_1 \cdot E_1$ and $B_2 = E_2$ provide the required solution. □

3.13. Lemma. *If $\mathcal{R} \subseteq \mathcal{S}$, then $\mathcal{M}_{\mathcal{S}}(t) = \mathcal{M}_{\mathcal{S}}(\mathcal{M}_{\mathcal{R}}(t))$.*

Proof. $\mathcal{M}_{\mathcal{S}}(t) \supseteq \mathcal{M}_{\mathcal{S}}(\mathcal{M}_{\mathcal{R}}(t))$ is obvious. Suppose A: $t \rightarrow^*_{\mathcal{R}} u$. By Lemma 3.12 we have a $B = B_1 \cdot B_2 \sqsupseteq A$ such that B_1: $t \rightarrow^*_{\mathcal{R}} v$ and B_2: $v \rightarrow^*_{\mathcal{S}} w$ is $RO_{\mathcal{R}}(v)$-free. Now apply part (2) of Lemma 3.10 to B_2 and let U be $RO_{\mathcal{R}}(v)$. We then have a C: $\omega_{\mathcal{R}}(v) \rightarrow^*_{\mathcal{S}} w'$ such that $\omega_{\mathcal{S}}(w') = \omega_{\mathcal{S}}(w)$. Since $\omega_{\mathcal{S}}$ increases with derivation and $B \sqsupseteq A$, $\omega_{\mathcal{S}}(w') = \omega_{\mathcal{S}}(w) \geqslant \omega_{\mathcal{S}}(u)$. We therefore have $\mathcal{M}_{\mathcal{S}}(t) \subseteq \mathcal{M}_{\mathcal{S}}(\mathcal{M}_{\mathcal{R}}(t))$. □

Stable Extension Lemma. *If $\mathcal{R} \subseteq \mathcal{S}$, then $\mathcal{M}_{\mathcal{R}}(t_1) = \mathcal{M}_{\mathcal{R}}(t_2) \Rightarrow \mathcal{M}_{\mathcal{S}}(t_1) = \mathcal{M}_{\mathcal{S}}(t_2)$.*

Proof. This is just a corollary of Lemma 3.13. □

The proof of the Continuity Lemma is similar to that of the Stable Extension Lemma, except that the permutation this time must bring forward the derivation steps applied to those subexpressions with respect to which the limiting value of the expression is required to be approximable and continuous. This is shown to be possible in the following lemma.

3.14. Lemma. *Suppose U, V is a partition of $Paths(t)$, and there is no $u \in U$, $v \in V$ such that $u < v$. Then for every $A \in D_{\mathcal{R}}(t)$, there is a $B \in D_{\mathcal{R}}(t)$ such that $B \sqsupseteq A$ and $B = B_1 \cdot B_2$ where B_1 is V-free and B_2 is $(Paths(Last(B_1)) - V)$-free.*

Proof. The proof is identical to the proof of Lemma 3.12 except that $U' = \{u' \in \mathit{Paths}(t') \mid \exists u \in U \backslash C_1 \text{ such that } u' \geq u\}$, and $V' = \mathit{Paths}(t') - U'$. Clearly, U' and V' satisfy the antecedents of the lemma with respect to t'. The induction hypothesis applied to C_2 yields $E = E_1 \cdot E_2 \in D_{\mathscr{R}}(t')$ such that $E \sqsupseteq C_2$, E_1 is V'-free and E_2 is $(\mathit{Paths}(\mathit{Last}(E_1)) - V')$-free. Therefore $B_1 = C_1 \cdot E_1$ and $B_2 = E_2$ again provide the required solution. □

Continuity Lemma. *For each* $x \in \mathcal{M}_{\mathscr{R}}(f(t_1, \ldots, t_k))$ *there are* $y_1 \in \mathcal{M}_{\mathscr{R}}(t_1), \ldots, y_k \in \mathcal{M}_{\mathscr{R}}(t_k)$ *such that* $x \in \mathcal{M}_{\mathscr{R}}(f(y_1, \ldots, y_k))$.

Proof. Let $t = f(t_1, \ldots, t_k)$ and $V = \Lambda$. There is a $A\colon t \to^*_{\mathscr{R}} u$ such that $x \leq \omega_{\mathscr{R}}(u)$. By Lemma 3.14, there is a $B \in D_{\mathscr{R}}(t)$ such that $B \sqsupseteq A$ and $B = B_1 \cdot B_2$, B_1 is Λ-free and B_2 is $(\mathit{Paths}(z) - \Lambda)$-free, where $z = \mathit{Last}(B_1)$. Apply Lemma 3.10 with $U = RO_{\mathscr{R}}(z) - \Lambda$. Now $\sigma(z, U)$ is obviously of the form $f(y_1, \ldots, y_k)$ where $y_1 \in \mathcal{M}_{\mathscr{R}}(t_1), \ldots, y_k \in \mathcal{M}_{\mathscr{R}}(t_k)$. The rest follows by Lemma 3.10. □

3.4. Review

It is natural to ask how general this proof of limiting completeness is. It would be nice if one could distill sufficient conditions for a language to be limiting complete for its final model. In retrospect, one can enumerate the following sufficient conditions:

- All programs are confluent.
- $\omega_{\mathscr{R}}$ increases with derivation.
- The Recognition Lemma.
- The Continuity Lemma.
- The Stable Extension Lemma.

The first two conditions are easy to verify and are usually satisfied. The Recognition Lemma, used in the proof of finality for $\mathcal{M}$, was a consequence of the Monotonicity Lemma which holds for all equational languages, together with the idempotence and monotonicity of $\omega_{\mathscr{R}}$, which are inherent properties of $\omega_{\mathscr{R}}$ not connected with regular systems. The only language specific property it depends on is that for each $x \in \omega_{\mathscr{R}}(T_{\Sigma_{\mathscr{R}}})$, the partial value represented by x is precisely recognizable by a context because it can be used as an argument pattern on the left-hand side of a rule in an extension of $\mathscr{R}$ after replacing Ω's with variables. The last two of the conditions above are not particularly easy to verify as we have seen, but they do have intuitive interpretations. The main requirement for them seems to be the possibility of carrying out the "permutations" of derivations made possible by the Parallel Moves Lemma for regular systems, together with the no-overlap property. Many equational languages use programs that are restrictions of regular systems, and for these the two properties hold as special cases.

An important question is whether the approximate normal form function ($\omega_{\mathscr{R}}$) is always the right notion of approximant. Intuitively, this has to do with the nature of "information" in equational computation. Taking $\omega_{\mathscr{R}}$ as the approximant function implies the traditional view that normal forms are visible values, as we observed in proving that $\mathcal{M}_{\mathcal{R}}$ is $\mathcal{R}$-computable for a regular system $\mathcal{R}$. This view turns out to be inappropriate for a common style of equational programming, exemplified by the equational subsets of Standard ML and Miranda. These languages differ from regular systems in two ways. They enforce a rigid distinction between constructors and defined function symbols, and they in effect allow only new function definitions in extensions as we noted in the introduction. The intuitive view in these languages is that only constructor terms represent visible information. Interestingly, the corresponding notion of approximant turns out to be the right one for full abstraction. Consider for instance the following somewhat artificial program:

$$\mathcal{R} = \{f(con1) = 0,\ g(con2) = 1\},$$

where the constructors are $con1$, $con2$, 0 and 1, and the defined symbols are f and g. Clearly, $f(con2)$ and $g(con1)$ are normal forms in this program. If normal forms are in effect visible, these two terms should be separable. In fact, of course, they are not even semi-separable because an extension of $\mathcal{R}$ is not allowed to add new equations for f and g and other functions are not allowed to use the non-constructors f or g in their argument patterns. In other words, $f(con2) \approx_{\mathcal{R}} g(con1)$ but $\mathcal{M}_{\mathcal{R}}(f(con2)) \neq \mathcal{M}_{\mathcal{R}}(g(con1))$. If we use the approximant function suggested above (call it $\pi_{\mathcal{R}}$) which simply replaces all subterms with a non-constructor at the head with Ω, we have $\pi_{\mathcal{R}}(f(con2)) = \pi_{\mathcal{R}}(g(con1)) = \Omega$ which captures the intuition that normal forms with non-constructors at the head are "meaningless". Given the limiting completeness proof for regular systems in general, it is not hard to show that $\pi_{\mathcal{R}}$ is the right notion of approximant to prove limiting completeness for constructor-based regular systems with functional extensions. The details are left to the reader.

One would like to generalize the ideas of the last paragraph so that the right approximant does not have to be guessed. In the next section, we offer such a generalization based on a formalization of the notion of *observable data content*, and use it to demonstrate the sufficiency for limiting completeness of the five conditions listed above. In this context, we also demonstrate the connection between the final/fully abstract models of equational languages and the formulation of Scott's theory of domains based on information systems. It should be noted that this generalization does not work for all languages. For instance, the language of constructor-based regular systems with the original supersets-as-extensions parameter requires approximate normal forms as approximants whereas the observable data content of terms in the language in the sense of the next section yields constructor terms as approximants. Technically, therefore, the two approaches to the definition of approximant are incomparable in their generality in proving limiting completeness. We believe that the observable data content is a better reflection of programming intuition about "manifest value". If this thesis is accepted, the appropriateness of the corresponding definition of approximant for limiting completeness can be seen as a *design principle* to test the mutual suitability of the parameters defining a language. The language of regular systems does pass this test because the two notions of approximant coincide for that language as we show below.

4. A general limiting completeness construction based on observable data content

Instead of choosing a specific language to test the new notion of *observable data content*, we shall use the insight gained from Section 3, and specifically the sufficient conditions of Section 3.4, to construct a *language independent* operational model which is limiting complete for the final model for any language which satisfies these conditions. Our central assumption in this section is that equational programs define functions to manipulate recursive data structures which are represented directly by terms viewed as labeled trees. Like the λ-calculus, the single syntactic category of terms is used to represent both static data and dynamic computed expressions. In some equational languages such as Miranda these roles are clearly demarcated by the exclusive use of constructor symbols to represent data structures. In the language of regular systems the roles are more ambiguous but still discernible; it is in fact possible to interpret this and many other languages as constructor based languages which take a few short cuts [19]. The idea of coding data with λ-expressions is somewhat unnatural in that semantically λ-expressions are taken to denote functions. The price for a natural representation of data in equational programming is the need for a direct semantic interpretation of the dual role of terms. This leads to a potentially different notion of approximant, focusing not on the normal form but on the possible role of the term as a data structure. The two crucial properties of this variant are that the approximant of a term is indistinguishable from the term in its role as data, and it is moreover a *unique* representation of the observable data content—another term which behaves identically in its role as data will have exactly the same approximant.

4.1. Representing observable data content

To construct a general framework for interpreting equational computation based on these intuitions, one needs to find a language independent way to encode the observable information embodied in a term in its role as a data structure. Operationally, a data structure must be *passive* in that none of its redices or their residual copies can be reduced. The observable information about a term as a data structure is therefore simply the totality of contexts in which the insertion of the term in a passive form leads to the output of an observable value.

Formally, if $U=\{p \mid p \geq q \in \mathit{Paths}(t)\}$ and $A\colon t \to^*_{\mathcal{R}} u$ is U-free then A is said to be *q-static.* A is Q-static if it is q-static for each $q \in Q \subseteq \mathit{Paths}(t)$. A q-static derivation essentially treats t/q as a passive data-structure. The informal notion of a context for the observation of passive terms can be formalized by defining an *observer* as a triple $(C[\ ], v, \mathcal{S})$ such that $C[\ ]$ is a context in program $\mathcal{S}$ and v is a visible term. The set of all observers relevant for a program $\mathscr{R}$ will be denoted by $\mathcal{O}_{\mathscr{R}}$.

$$\mathcal{O}_{\mathscr{R}} = \{(C[\ ], v, \mathscr{S}) \mid \mathscr{R} \leqslant \mathscr{S} \text{ and } C[\ ] = (c, q) \text{ where } c \in T_{\Sigma_{\mathscr{S}}}\}.$$

We shall use $\langle t \rangle_{\mathscr{R}}$ to denote the information content of t as a data structure in $\mathscr{R}$. Before defining $\langle t \rangle_{\mathscr{R}}$ it is convenient to define the set $[\![u]\!]$ of all data structures that "satisfy" an observer $u = (C[\ ], v, \mathscr{S})$.

$$[\![u]\!] = \{t \in T_{\Sigma_{\mathscr{R}}} \mid \mathscr{R} \leqslant \mathscr{S}, C[\ ] = (c, q) \text{ and } C[t] \rightarrow^*_{\mathscr{S}} v \text{ by a } q\text{-static derivation}\}.$$

Intuitively, an observer is a context $C[\ ]$ in program $\mathscr{S}$ which wishes to "observe" (i.e., evaluate to) the visible value v. The set $[\![u]\!]$ contains precisely those data structures which, when used to fill the hole in $C[\ ]$, evaluate to the desired output v. We can now define $\langle t \rangle_{\mathscr{R}}$ as the set of all observers satisfied by t:

$$\langle t \rangle_{\mathscr{R}} = \{u \in \mathcal{O}_{\mathscr{R}} \mid t \in [\![u]\!]\}.$$

These definitions extend naturally to sets of contexts and sets of terms respectively. If U is a set of contexts then $[\![U]\!] = \bigcap_{u \in U} [\![u]\!]$, and if V is a set of terms then $\langle V \rangle_{\mathscr{R}} = \bigcup_{v \in V} \langle v \rangle_{\mathscr{R}}$. Intuitively, a data structure belongs to $[\![U]\!]$ iff it satisfies *all* observers in U whereas $\langle V \rangle_{\mathscr{R}}$ denotes *all* observers satisfiable through *some* term in V. $\langle t \rangle_{\mathscr{R}}$ is an abstract representation of the data content approximant $\pi_{\mathscr{R}}(t)$. The approximation relation $\leqslant$ between concrete approximants translates to the subset relation for abstract approximants. A nice feature of the abstract version is that one can speak naturally about the approximant of a *set* of terms, which roughly corresponds to the LUB of the individual approximants.

The concrete approximant function $\pi_{\mathscr{R}}$ can now be seen as a canonical data representation function for the language, acting as a retract from $T_{\Sigma_{\mathscr{R}}}$ to $\Sigma_{\mathscr{R}}$-Ω-terms. Such a retract and the corresponding approximation order must satisfy the following axioms which establish their relation to abstract approximants:

(1) $\langle t \rangle_{\mathscr{R}} = \langle \pi_{\mathscr{R}}(t) \rangle_{\mathscr{R}}$,

(2) $\langle t_1 \rangle_{\mathscr{R}} \subseteq \langle t_2 \rangle_{\mathscr{R}} \Leftrightarrow \pi_{\mathscr{R}}(t_1) \leqslant \pi_{\mathscr{R}}(t_2)$.

It is easy to show that these axioms imply that $\pi_{\mathscr{R}}$ is idempotent, and a projection ($\pi_{\mathscr{R}}(t) \leqslant t$). With the Monotonicity Lemma, the axioms also imply that $\pi_{\mathscr{R}}$ is monotonic. We leave the details to the reader. As an example for $\pi_{\mathscr{R}}$ consider a constructor-based language like Miranda. In such a language, passive terms with a non-constructor at the head have no observable data content since they cannot be matched by a pattern. The function $\pi_{\mathscr{R}}$ therefore replaces all such terms by Ω as we suggested in Section 3.4.

Although the normal form oriented approximant function $\omega_{\mathscr{R}}$ does not always satisfy these axioms, the two notions of approximant do coincide for regular systems as we now show. This result indicates that the language of regular systems is particularly well-rounded in some sense.

4.1. Proposition. *For any A: $t \rightarrow^*_{\mathscr{R}} u$ and any $q \in Paths(t)$, there is a $B = B_1 \cdot B_2 \sqsupseteq A$ such that B_1: $t \rightarrow^*_{\mathscr{R}} t[q \leftarrow v] = w$ for some v, and B_2: $w \rightarrow^*_{\mathscr{R}} x$ is q-static. Moreover, if A is Q-static for some Q then B_1 and B_2 are Q-static too.*

Proof. This is just a corollary of Lemma 3.14, letting $U = \{r \in \mathit{Paths}(t) \mid \exists p.\, r = p \cdot q\}$. □

4.2. Proposition. *$t_1 \leqslant t_2$ implies $\langle t_1 \rangle_{\mathcal{R}} \subseteq \langle t_2 \rangle_{\mathcal{R}}$.*

Proof. Follows easily by the Monotonicity Lemma. □

4.3. Proposition. *$\langle \omega_{\mathcal{R}}(t) \rangle_{\mathcal{R}} \supseteq \langle t \rangle_{\mathcal{R}}$ for all $t \in T_{\Sigma, \mathcal{R}}$.*

Proof. Suppose $(C[\ ], v, \mathcal{S}) \in \langle t \rangle_{\mathcal{R}}$, where $C[\ ] = (c, q)$. Therefore, A: $C[t] \rightarrow^*_{\mathcal{S}} v$ by a q-static derivation and $\mathcal{R} \leqslant \mathcal{S}$. Let $Q = \{q \cdot p \mid p \in RO_{\mathcal{S}}(t)\}$. A is clearly Q-free. Let $u = \sigma(C[t], Q) \leqslant C[\omega_{\mathcal{S}}(t)]$. By Lemma 3.10, there is a derivation $u \rightarrow^*_{\mathcal{S}} v$, therefore by the Monotonicity Lemma there is a derivation $C[\omega_{\mathcal{S}}(t)] \rightarrow^*_{\mathcal{S}} v$, and therefore, $(C[\ ], v, \mathcal{S}) \in \langle \omega_{\mathcal{S}}(t) \rangle_{\mathcal{R}}$. Since $\omega_{\mathcal{S}}(t) \leqslant \omega_{\mathcal{R}}(t)$, we have $(C[\ ], v, \mathcal{S}) \in \langle \omega_{\mathcal{R}}(t) \rangle_{\mathcal{R}}$ by Proposition 4.2. □

4.4. Corollary. $\langle \omega_{\mathcal{R}}(t) \rangle_{\mathcal{R}} = \langle t \rangle_{\mathcal{R}}$.

The following proposition points out that if an approximate normal form is a prefix of a term, then it is a prefix of the approximate normal form of that term.

4.5. Proposition. *$\omega_{\mathcal{R}}(t) \leqslant t'$ iff $\omega_{\mathcal{R}}(t) \leqslant \omega_{\mathcal{R}}(t')$.*

Proof. Easy induction on the structure of $\omega_{\mathcal{R}}(t)$. □

The next proposition is a variant of the Recognition Lemma.

4.6. Proposition. *For every $t \in T_{\Sigma, \mathcal{R}}$, there is a $u \in \langle t \rangle_{\mathcal{R}}$ such that $t' \in [\![u]\!]$ implies $\omega_{\mathcal{R}}(t') \geqslant \omega_{\mathcal{R}}(t)$.*

Proof. Let t_1 be the linear term obtained by substituting a distinct variable for each Ω in $\omega_{\mathcal{R}}(t)$. Let $\mathcal{S} = \mathcal{R} \cup \{g(t_1) = v\}$ where g is absent from the rules in $\mathcal{R}$, and v is a visible term. Let $u = ((g(t_1), 1), v, \mathcal{S})$. Clearly, $u \in \langle t \rangle_{\mathcal{R}}$. If $t' \in [\![u]\!]$ then $u \in \langle t' \rangle_{\mathcal{R}}$ since $\mathcal{R} \leqslant \mathcal{S}$. Given that the only rule for g is the one we added to $\mathcal{R}$ above, t' must be a substitution instance of t_1, so, by Proposition 4.5, $\omega_{\mathcal{R}}(t) \leqslant \omega_{\mathcal{R}}(t')$. □

4.7. Corollary. $\langle t_1 \rangle_{\mathcal{R}} \subseteq \langle t_2 \rangle_{\mathcal{R}} \Rightarrow \omega_{\mathcal{R}}(t_1) \leqslant \omega_{\mathcal{R}}(t_2)$.

Proof. If $\langle t_1 \rangle_{\mathcal{R}} \subseteq \langle t_2 \rangle_{\mathcal{R}}$, then the special $u \in \langle t_1 \rangle_{\mathcal{R}}$ given by Proposition 4.6 must also belong to $\langle t_2 \rangle_{\mathcal{R}}$, and therefore $t_2 \in [\![u]\!]$. □

4.8. Lemma. $\langle t_1 \rangle_{\mathcal{R}} \subseteq \langle t_2 \rangle_{\mathcal{R}} \Leftrightarrow \omega_{\mathcal{R}}(t_1) \leqslant \omega_{\mathcal{R}}(t_2)$.

Proof. Immediate by Proposition 4.2, and Corollaries 4.4 and 4.7. □

Since $\omega_{\mathscr{R}} = \pi_{\mathscr{R}}$ for the language of regular systems, that language can be used as a proven example for the general construction in the rest of this section.

4.2. *Information systems*

Information systems are a natural vehicle for the task of building a general limiting model because they can easily accommodate the language independent representation of observable information and approximants described above. They also reveal the connection between the semantics of equational computation and that of higher-order systems such as the λ-calculus. This is helpful because the two paradigms coexist in many real languages, with the equational component providing the base values and functions for higher-order computation. It turns out that if the sufficient conditions of Section 3.4 are satisfied by a language, the model based on the new notion of approximant is not only limiting complete but also has other pleasant characteristics including functions that are continuous in a nice way and carriers that are well-formed and well-understood structures called domains. Such characteristics are often assumed for the sets of base values and functions used in λ-calculus dialects.

In order to keep the notation and technical machinery simpler, we assume that the language concerned has a single sort and all programs use single-sorted signatures based on this common sort. It is clear that the generalization from the single-sorted to the many-sorted case is not very difficult. Arity restrictions on functions are still assumed to be in force. Before carrying out the construction, we review the basic definitions concerning information systems. The original reference [17] gives a lucid explanation of the intuitive ideas. Note that our definitions of information system and cartesian product differ in minor ways from those given by Scott.

An *information system* is a triple $I = (D, Con, \vdash)$, where D is called the set of *propositions*, each asserting a *finite* amount of information regarding possible *data elements* defined by the information system. Con is a set of *finite* subsets of D, and the idea is that all finite collections of mutually consistent propositions are in Con. The *entailment* relation $\vdash$ between members of Con and members of D embodies the notion that if the information in a proposition is contained *completely* in the collective information provided by the propositions in a consistent set, then the set entails the proposition regarding the elements for which all propositions in the set hold. Following Larsen and Winskel [8], we have dropped the empty proposition Δ as a component in the definition. A triple constituting an information system must satisfy the following *axioms*, which are more or less self evident:

(1) $B \subseteq C \in Con \Rightarrow B \in Con$,

(2) $d \in D \Rightarrow \{d\} \in Con$,

(3) $B \vdash d \Rightarrow B \cup \{d\} \in Con$,

(4) $B \in Con$ and $b \in B \Rightarrow B \vdash b$.

(5) $B, C \in Con$, $B \vdash C$ and $C \vdash d \Rightarrow B \vdash d$.

In the last axiom, we have used the natural generalization of $\vdash$ to a relation between consistent sets, where $B \vdash C$ iff $B \vdash c$ for each $c \in C$. The set of data elements defined by such an information system I is denoted by $|I|$. Each member of $|I|$ is a set of propositions (not necessarily finite) that is *consistent* in itself and *deductively closed* or closed under entailment. More formally, given an information system $I = (D, Con, \vdash)$, $e \in |I|$ iff

(i) $B \subseteq e$ and B is finite $\Rightarrow B \in Con$,

(ii) $B \subseteq e$ and $B \vdash a \Rightarrow a \in e$.

Sets of propositions which satisfy (i) are said to be consistent, and for *any* set B of propositions, the deductive closure of B is $\bar{B} = \{a \mid C \subseteq B$ and $C \vdash a\}$. Obviously, $\bar{B}$ is an element in $|I|$ iff B is consistent. Roughly speaking, the element $\bar{B}$ represents the LUB of the finite values in the set B, and is very similar to the downward completion defined for sets of approximate normal forms in Section 3.2. The only difference is that deductive closure also completes the set B *upwards* in the sense of including all the data structures obtainable by overlapping members of B. This is immaterial for $\omega_{\mathscr{R}}(t\downarrow\mathscr{R})$ since it is directed. The approximation order on elements of $|I|$ is simply the subset relation, and $(|I|, \subseteq)$ forms a consistently complete algebraic cpo, i.e., a *domain*.

Approximable maps are a way to define continuous functions between domains without directly specifying the values of such a function for an *infinite* input. This is possible since any infinite value in a domain is in a sense imaginary, in that it is always obtained as the limit of all finite approximations to it. It is therefore sufficient to specify the behavior of the function for finite inputs and its behavior in the infinite cases then simply follows, given certain restrictions. Formally, an approximable map $f: I_1 \to I_2$ between information systems $I_1 = (D_1, Con_1, \vdash_1)$ and $I_2 = (D_2, Con_2, \vdash_2)$ is a relation between Con_1 and Con_2 such that the following axioms hold:

(1) $\emptyset f \emptyset$,

(2) $U f V_1$, and $U f V_2 \Rightarrow U f (V_1 \cup V_2)$,

(3) $U' \vdash_1 U$, $U f V$ and $V \vdash_2 V' \Rightarrow U' f V'$.

The intuitive justification for these axioms is straightforward, and is found in [17]. There is a natural way to think of such an approximable map as a (continuous) *function* between the corresponding *domains*. Given any $e \in |I_1|$,

$$f(e) = \{x \in D_2 \mid \exists\, U \subseteq e \text{ such that } U \in Con_1 \text{ and } U f \{x\}\}.$$

It is easy to show that f understood this way is indeed a continuous function of the appropriate type. Following Scott, we shall use the names of approximable maps ambiguously to also represent their incarnations as continuous functions. We need one final notion to accommodate the fact that we are dealing with polyadic functions—we need a notion of cartesian products. Again, for convenience, we have to modify the standard construction given by Scott [17], and use an alternative equivalent construction:

Let $I_1, I_2, \ldots, I_k$ be information systems where $I_i = (D_i, Con_i, \vdash_i)$. Then their product $I_1 \times I_2 \times \cdots \times I_k$ is the information system $I = (D, Con, \vdash)$ such that

(1) $D = D_1 \times \cdots \times D_k$,

(2) $Con = \{V \mid i\text{th}(V) \in Con_i,\ 1 \leq i \leq k\}$,

(3) $V \vdash d$ iff $i\text{th}(V) \vdash_i i\text{th}(d)$, $1 \leq i \leq k$,

where $i\text{th}((d_1, \ldots, d_k)) = d_i$ and $i\text{th}(V) = \{i\text{th}(d) \mid d \in V\}$. It is easy to check that I is indeed an information system. It is also straightforward to show that this notion of product is appropriate in the category of information systems and approximable maps. We leave the details to the reader.

4.3. *The general limiting model*

We shall construct a limiting operational model $\mathcal{N}$ for an arbitrary language $\mathcal{L}$ which satisfies the five conditions listed in Section 3.4 (and restated in a modified form below) and show that they are sufficient for limiting completeness by showing that $\mathcal{N}$ is isomorphic to the final model for $\mathcal{L}$. The algebra assigned by $\mathcal{N}$ to a program $\mathcal{R}$ will be single-sorted due to the restriction assumed above, and the carrier for the algebra will be an information system $I_{\mathcal{R}}$. The value domain generated by $I_{\mathcal{R}}$ (i.e., $|I_{\mathcal{R}}|$) is not $\Sigma_{\mathcal{R}}$-reachable since it is a complete partial order usually containing uncountably many elements. However, *all* the unreachable elements will be *infinite*. Since only finite data values are actually computable in any program, this is not very damaging. A restriction to reachability is straightforward, and we shall limit ourselves to showing that $\mathcal{N}$ is the final model for a given language assuming such a restriction.

The construction of $I_{\mathcal{R}}$ is easy since the construction of an information system only requires the specification of finite values. The material for finite values is already at hand—the set $T_{\Sigma_{\mathcal{R}}}$ of ground terms in their role as data structures. The abstract approximant $\langle t \rangle_{\mathcal{R}}$ will do as a unique representation of the information in the finite data structure t. It is of course possible that $\langle t_1 \rangle_{\mathcal{R}} = \langle t_2 \rangle_{\mathcal{R}}$ for distinct terms t_1 and t_2, so we are indirectly carrying out a quotient construction. Formally, the information system for the domain of $\mathcal{R}$ is defined as follows.

4.9. Definition. $I_{\mathcal{R}} = (T_{\Sigma_{\mathcal{R}}}, Con_{\mathcal{R}}, \vdash_{\mathcal{R}})$, where $U \in Con_{\mathcal{R}}$ iff $U \subseteq T_{\Sigma_{\mathcal{R}}}$, U is finite, and there is some $t \in T_{\Sigma_{\mathcal{R}}}$ such that $\langle U \rangle_{\mathcal{R}} \subseteq \langle t \rangle_{\mathcal{R}}$. For $U \in Con_{\mathcal{R}}$ and $t \in T_{\Sigma_{\mathcal{R}}}$, $U \vdash_{\mathcal{R}} t$ iff $\langle U \rangle_{\mathcal{R}} \supseteq \langle t \rangle_{\mathcal{R}}$.

In other words, a finite collection of data structures will be considered to be mutually consistent if there is a single data structure encompassing all the information in the collection. The *entailment* relation $\vdash_{\mathcal{R}}$ is simply containment with respect to information content. It is easy to check that $I_{\mathcal{R}}$ is indeed an information system.

We now restate the sufficient conditions for limiting completeness in a form that is suitable for working with information systems. In particular, we avoid the use of $\pi_{\mathcal{R}}$ and work with terms directly in their role as data structures. The actual verification of these conditions must usually involve the use of $\pi_{\mathscr{R}}$ mediated by the two axioms that relate it to abstract approximants.

For any $u \in \mathscr{O}_{\mathscr{R}}$, let $\tilde{u} = \{v \mid [\![u]\!] \subseteq [\![v]\!]\}$. The set $\tilde{u}$ contains observers less demanding than u. Therefore, if $u \in \langle t \rangle_{\mathscr{R}}$ for some t then $\tilde{u} \subseteq \langle t \rangle_{\mathscr{R}}$. Let P_m denote the set $\{1, \ldots, m\}$ of paths. The deductive closures used in the conditions below relate to entailment in the relevant (information system) carrier. The formulation of the recognition condition is similar to the variant in Proposition 4.6.

- All programs are confluent.
- (Unfolding) $t \rightarrow^*_{\mathscr{R}} u \Rightarrow \langle t \rangle_{\mathscr{R}} \subseteq \langle u \rangle_{\mathscr{R}}$.
- (Recognition) For each $t \in T_{\Sigma_{\mathscr{R}}}$ there is a $u \in \langle t \rangle_{\mathscr{R}}$ such that $\tilde{u} = \langle t \rangle_{\mathscr{R}}$.
- (Continuity) For each $u \in \overline{f(t_1, \ldots, t_k)\downarrow\mathscr{R}}$ there are $y_1 \in \overline{t_1\downarrow\mathscr{R}}, \ldots, y_k \in \overline{t_k\downarrow\mathscr{R}}$ such that $f(y_1, \ldots, y_k) \rightarrow^*_{\mathscr{R}} z$ by a P_k-static derivation and $\langle u \rangle_{\mathscr{R}} \subseteq \langle z \rangle_{\mathscr{R}}$.
- (Stable extension) $\mathscr{R} \leqslant \mathscr{S}$ and $\overline{t\downarrow\mathscr{R}} = \overline{u\downarrow\mathscr{R}} \Rightarrow \overline{t\downarrow\mathscr{S}} = \overline{u\downarrow\mathscr{S}}$.

In a language which satisfies the first two of the conditions above, the set $\pi_{\mathscr{R}}(t\downarrow\mathscr{R})$ is directed, which is equivalent to saying that the set $t\downarrow\mathscr{R}$ is consistent in $I_{\mathscr{R}}$. The deductive closure of the naive operational denotation of an expression is therefore an element in the value domain $|I_{\mathscr{R}}|$.

4.10. Lemma. *$\overline{t\downarrow\mathscr{R}} \in |I_{\mathscr{R}}|$ for each $t \in T_{\Sigma_{\mathscr{R}}}$ and $\mathscr{R} \in \mathfrak{R}$.*

Proof. All we need to do is to show that $t\downarrow\mathscr{R}$ is consistent. If V is any finite subset of $t\downarrow\mathscr{R}$, then by the confluence of $\mathscr{R}$, there is $u \in t\downarrow\mathscr{R}$ such that $v \rightarrow^*_{\mathscr{R}} u$ for each $v \in V$. By the Unfolding Condition and axiom (2) for $\pi_{\mathscr{R}}, \langle u \rangle_{\mathscr{R}} \supseteq \langle V \rangle_{\mathscr{R}}$, therefore $V \in Con_{\mathscr{R}}$. □

The two axioms for the approximant function $\pi_{\mathscr{R}}$ imply a natural correspondence between deductive closure and downward completion which means that the natural notions of semantic equality are the same for both abstract and concrete approximants.

4.11. Theorem. $\overline{t\downarrow\mathscr{R}} = \overline{u\downarrow\mathscr{R}} \Leftrightarrow \overline{\pi_{\mathscr{R}}(t\downarrow\mathscr{R})} = \overline{\pi_{\mathscr{R}}(u\downarrow\mathscr{R})}$, *using closure under entailment in the former and downward completion in the latter.*

Proof. ($\Rightarrow$): Suppose $\overline{t\downarrow\mathscr{R}} = \overline{u\downarrow\mathscr{R}}$, and $x \in \overline{\pi_{\mathscr{R}}(t\downarrow\mathscr{R})}$. There must be a $z \in t\downarrow\mathscr{R}$ such that $x \leqslant \pi_{\mathscr{R}}(z)$. By the antecedent, $z \in \overline{u\downarrow\mathscr{R}}$ hence there must be a $U \subseteq u\downarrow\mathscr{R}$ such that $U \vdash_{\mathscr{R}} z$, and therefore $\langle U \rangle_{\mathscr{R}} \supseteq \langle z \rangle_{\mathscr{R}}$. Since $\mathscr{R}$ is confluent, there is a $w \in u\downarrow\mathscr{R}$ such that $v \rightarrow^*_{\mathscr{R}} w$ for each $v \in U$. By unfolding, we have $\langle w \rangle_{\mathscr{R}} \supseteq \langle U \rangle_{\mathscr{R}} \supseteq \langle z \rangle_{\mathscr{R}}$. By axiom (2) $\pi_{\mathscr{R}}(w) \geqslant \pi_{\mathscr{R}}(z) \geqslant x$ and $x \in \overline{\pi_{\mathscr{R}}(u\downarrow\mathscr{R})}$.

($\Leftarrow$): Similar. □

Construction of the rest of the algebra $\mathscr{N}_{\mathscr{R}}$ is straightforward. For each constant $g \in \Sigma_{\mathscr{R}}$, $g_{\mathscr{R}} = \overline{g\downarrow\mathscr{R}}$. For each $f \in \Sigma_{\mathscr{R}}$ of arity k, we need an approximable map $f_{\mathscr{R}}\colon I^k_{\mathscr{R}} \rightarrow I_{\mathscr{R}}$ which will capture the computation of f in $\mathscr{R}$ for all possible *finite data* *structure* inputs. Given a term $u = f(t_1, \ldots, t_k)$, this includes all derivations from u in which all t_i are passive, i.e., all P_k-static derivations. Suppose we have the k-fold product system $I^k_{\mathcal{R}} = (T^k_{\Sigma_{\mathcal{R}}}, Con^k_{\mathcal{R}}, \vdash^k_{\mathcal{R}})$; $f_{\mathcal{R}} \subseteq Con^k_{\mathcal{R}} \times Con_{\mathcal{R}}$ is the *smallest* relation such that:

(1) $(t_1, \ldots, t_k) f_{\mathcal{R}} t$ if $f(t_1, \ldots, t_k) \rightarrow^*_{\mathcal{R}} t$ by a P_k-static derivation;

(2) $V f_{\mathcal{R}} U$ if $V f_{\mathcal{R}} u$ for each $u \in U$;

(3) if $V' \vdash^k_{\mathcal{R}} V$, $U \vdash_{\mathcal{R}} U'$ and $V f_{\mathcal{R}} U$ then $V' f_{\mathcal{R}} U'$.

Condition (1) gives the basic relation between data structures based on P_k-static derivations. The other two are simple closure conditions which round off the relation to ensure consistency and uniqueness.

4.12. Lemma. *$f_{\mathcal{R}}$ is an approximable map for each $\mathcal{R} \in \mathfrak{R}$, k-ary $f \in \Sigma_{\mathcal{R}}$, $k > 0$.*

Proof. We need to show that the three axioms for approximable maps hold for $f_{\mathcal{R}}$. Suppose $U \in Con^k_{\mathcal{R}}$, $U f_{\mathcal{R}} V$ and $U f_{\mathcal{R}} V'$. We have $\emptyset f_{\mathcal{R}} \emptyset$ by condition (2) in the definition of $f_{\mathcal{R}}$. Condition (3) is itself the last axiom. It remains to show that $U f_{\mathcal{R}} V$ and $U f_{\mathcal{R}} V'$ implies $U f_{\mathcal{R}} (V \cup V')$. Given condition (2) above, this reduces to showing that $(V \cup V') \in Con_{\mathcal{R}}$. Since $U \in Con^k_{\mathcal{R}}$, the k-tuples in U are mutually consistent as data structures. By the definition of consistency, there is a tuple $(t_1, \ldots, t_k)$ such that $\langle t_i \rangle_{\mathcal{R}} \supseteq \langle i\text{th}(U) \rangle_{\mathcal{R}}$, $1 \leq i \leq k$. Given the structure of the definition of $f_{\mathcal{R}}$, there must be $Y, Y' \in Con^k_{\mathcal{R}}$ and $W, W' \in Con_{\mathcal{R}}$ such that $U \vdash^k_{\mathcal{R}} Y$, $U \vdash^k_{\mathcal{R}} Y'$, $W \vdash_{\mathcal{R}} V$ and $W' \vdash_{\mathcal{R}} V'$ and one can show $Y f_{\mathcal{R}} W$ and $Y' f_{\mathcal{R}} W'$ using conditions (1) and (2) of the definition of $f_{\mathcal{R}}$ alone, i.e., for each $w \in W$, there is a $y = (y_1, \ldots, y_k) \in Y$ such that $f(y_1, \ldots, y_k) \rightarrow^*_{\mathcal{R}} w$ by a P_k-static derivation, and similarly for W' and Y'. By the definition of entailment, $\langle t_i \rangle_{\mathcal{R}} \supseteq \langle y_i \rangle_{\mathcal{R}}$, $1 \leq i \leq k$. By axiom (1) for $\pi_{\mathcal{R}}$, $f(\pi_{\mathcal{R}}(y_1), \ldots, \pi_{\mathcal{R}}(y_k)) \rightarrow^*_{\mathcal{R}} w$ by a P_k-static derivation, and by axiom (2), $\pi_{\mathcal{R}}(t_i) \geq \pi_{\mathcal{R}}(y_i)$, $1 \leq i \leq k$. By the Monotonicity Lemma and the fact that $\pi_{\mathcal{R}}(t_i) \leq t_i$, for each $w \in W$ there is a P_k-static derivation $f(t_1, \ldots, t_k) \rightarrow^*_{\mathcal{R}} z$ such that $\langle z \rangle_{\mathcal{R}} \supseteq \langle w \rangle_{\mathcal{R}}$, and similarly for each $w' \in W'$. We therefore have $V \cup V' \subseteq \overline{f(t_1, \ldots, t_k){\downarrow}\mathcal{R}}$ and hence $V \cup V' \in Con_{\mathcal{R}}$. □

We complete the proof that $\mathcal{N}_{\mathcal{R}}$ is a $\Sigma_{\mathcal{R}}$-algebra by showing that the natural denotation $\overline{t{\downarrow}\mathcal{R}}$ of each $t \in T_{\Sigma_{\mathcal{R}}}$ yields a homomorphism from $T_{\Sigma_{\mathcal{R}}}$ to $\mathcal{N}_{\mathcal{R}}$.

4.13. Lemma. *$\mathcal{N}_{\mathcal{R}}(t) = \overline{t{\downarrow}\mathcal{R}}$ for all terms t.*

Proof. Proceed by induction on the structure of t. The basis case is trivial. Now suppose $\mathcal{N}_{\mathcal{R}}(t_i) = \overline{t_i{\downarrow}\mathcal{R}}$, $1 \leq i \leq k$. Consider $t = f(t_1, \ldots, t_k)$. We first show that $f_{\mathcal{R}}(\overline{t_1{\downarrow}\mathcal{R}}, \ldots, \overline{t_k{\downarrow}\mathcal{R}}) \subseteq \overline{t{\downarrow}\mathcal{R}}$, which amounts to the claim that if $V \in Con^k_{\mathcal{R}}$, $V \subseteq (\overline{t_1{\downarrow}\mathcal{R}}, \ldots, \overline{t_k{\downarrow}\mathcal{R}})$ and $V f_{\mathcal{R}} U$ then $U \subseteq \overline{t{\downarrow}\mathcal{R}}$. By our definition of cartesian product, $i\text{th}(V) \subseteq \overline{t_i{\downarrow}\mathcal{R}}$, $1 \leq i \leq k$, and by confluence and unfolding, there are $v_i \in t_i{\downarrow}\mathcal{R}$ such that $i\text{th}(V) \subseteq \langle v_i \rangle_{\mathcal{R}}$. Let $v = (v_1, \ldots, v_k)$ and we have $\{v\} \vdash^k_{\mathcal{R}} V$. Now by the definition of $f_{\mathscr{R}}$, we must have a W such that $W \vdash_{\mathscr{R}} U$, and for each $w \in W$, there is $y \in T^k_{\Sigma,\mathscr{R}}$ such that $V \vdash^k_{\mathscr{R}} y$ and $\{y\} f_{\mathscr{R}} \{w\}$ by condition (1) in the definition of $f_{\mathscr{R}}$. Therefore, $\{v\} \vdash^k_{\mathscr{R}} \{y\}$ and the rest follows by the Monotonocity Lemma and the axioms for $\pi_{\mathscr{R}}$ as in the proof of Lemma 4.12.

The continuity condition comes into play in the converse, i.e., in showing $e = f_{\mathscr{R}}(\overline{t_1\downarrow\mathscr{R}}, \ldots, \overline{t_k\downarrow\mathscr{R}}) \supseteq \overline{t\downarrow\mathscr{R}}$. All we need to show is that $t\downarrow\mathscr{R} \subseteq e$. Suppose $u \in t\downarrow\mathscr{R}$, i.e., $t \to^*_{\mathscr{R}} u$. By continuity, there are $u_i \in \overline{t_i\downarrow\mathscr{R}}, 1 \leqslant i \leqslant k$, such that for some $w, f(u_1, \ldots, u_k) \to^*_{\mathscr{R}} w$ by a P_k-static derivation and $\langle u\rangle_{\mathscr{R}} \subseteq \langle w\rangle_{\mathscr{R}}$. Clearly, $w \in e$ and therefore $u \in e$. □

Since the stable extension property has been assumed, it only remains to show that each $\mathscr{N}_{\mathscr{R}}$ is $\mathscr{R}$-computable in order to demonstrate that $\mathscr{N}$ is a model (except for $\Sigma_{\mathscr{R}}$-reachability which can be easily obtained by restricting the domain $|I_{\mathscr{R}}|$) for any language that satisfies the five sufficient conditions above. This property as well as the fact that $\mathscr{N}$ is final and therefore fully abstract are easy corollaries of the following simple lemma. Let $\mathscr{F}_{\mathscr{R}}(t) = \langle \mathscr{N}_{\mathscr{R}}(t)\rangle_{\mathscr{R}}$.

4.14. Lemma. $\mathscr{F}_{\mathscr{R}}(t_1) = \mathscr{F}_{\mathscr{R}}(t_2) \Leftrightarrow \mathscr{N}_{\mathscr{R}}(t_1) = \mathscr{N}_{\mathscr{R}}(t_2)$.

Proof. The if part ($\Leftarrow$) is obvious by the definition of $\mathscr{F}_{\mathscr{R}}$. Suppose $\mathscr{F}_{\mathscr{R}}(t_1) = \mathscr{F}_{\mathscr{R}}(t_2)$ and $z \in \mathscr{N}_{\mathscr{R}}(t_1)$. By the recognition condition, $\forall u \in \langle z\rangle_{\mathscr{R}}$ such that $\langle z\rangle_{\mathscr{R}} = \tilde{u}$. But $u \in \mathscr{F}_{\mathscr{R}}(t_1) = \mathscr{F}_{\mathscr{R}}(t_2)$, hence there is an $x \in \mathscr{N}_{\mathscr{R}}(t_2)$ such that $u \in \langle x\rangle_{\mathscr{R}}$. Therefore $\langle z\rangle_{\mathscr{R}} = \tilde{u} \subseteq \langle x\rangle_{\mathscr{R}}$ and by Lemma 4.13, $z \in \mathscr{N}_{\mathscr{R}}(t_2)$. □

The $\mathscr{R}$-computability of $\mathscr{N}_{\mathscr{R}}$ follows as an easy corollary.

4.15. Lemma. *$\mathscr{N}_{\mathscr{R}}$ is $\mathscr{R}$-computable, i.e., for each visible term v and $t \in T_{\Sigma,\mathscr{R}}$, $\mathscr{N}_{\mathscr{R}}(v) = \mathscr{N}_{\mathscr{R}}(t) \Leftrightarrow t \to^*_{\mathscr{R}} v$.*

Proof. Assume $\mathscr{N}_{\mathscr{R}}(v) = \mathscr{N}_{\mathscr{R}}(t)$, and therefore $\mathscr{F}_{\mathscr{R}}(v) = \mathscr{F}_{\mathscr{R}}(t)$. Obviously, there is a $(C[\ ], \mathscr{R}, v) \in \mathscr{F}_{\mathscr{R}}(v)$ where $C[\ ]$ is an *empty* context. Therefore $(C[\ ], \mathscr{R}, v) \in \mathscr{F}_{\mathscr{R}}(t)$ which is only possible if $v \in t\downarrow\mathscr{R}$. □

4.16. Proposition. $t_1 \approx_{\mathscr{R}} t_2 \Rightarrow \mathscr{F}_{\mathscr{R}}(t_1) = \mathscr{F}_{\mathscr{R}}(t_2)$.

Proof. Obvious from the definitions, since $\langle \overline{t\downarrow\mathscr{R}}\rangle_{\mathscr{R}} = \langle t\downarrow\mathscr{R}\rangle_{\mathscr{R}}$ by the definition of entailment in $I_{\mathscr{R}}$. □

The finality of $\mathscr{N}_{\mathscr{R}}$ follows immediately.

4.17. Lemma. $t_1 \approx_{\mathscr{R}} t_2 \Rightarrow \mathscr{N}_{\mathscr{R}}(t_1) = \mathscr{N}_{\mathscr{R}}(t_2)$.

Proof. Obvious by Proposition 4.16 and Lemma 4.14 above. □

To summarize we state the following theorem.

4.18. Theorem. *For a language which satisfies the five conditions listed at the beginning of Section* 4.3, *the limiting operational model $\mathcal{N}$, built with domains and approximable maps, is isomorphic to the final model.*

Proof. Immediate by preceding lemmas. □

5. Concluding remarks

There are several reasons to believe that the semantics for equational languages presented above is the "right one". It is fully abstract and final in a natural sense. The fundamental operational semantics of many commonly used languages are limiting complete with respect to it. Its components can be constructed in the form of approximable maps and Scott-domains which makes it suitable for coupling with the standard semantics of λ-calculus dialects. One limitation on its generality is that it is effectively focused on *safe* (call-by-need) operational semantics, one that guarantees computation of all visible results derivable by equational deduction. This is also the case for most other formal approaches including initial algebra semantics. Unsafe evaluation strategies such as call-by-value obviously correspond to a different and in some ways much simpler semantics because only finite values are computable in call-by-value fashion even in the limit. It would be interesting to consider the use of our framework for computation rules other than call-by-need. In fact, the sufficient conditions used in Section 4 can be reinterpreted in a more general way as conditions on the derivation relation ($\to_{\mathcal{R}}^{*}$) which can be restricted to, say, innermost rewriting only. It seems possible to derive a more abstract theory based on an arbitrary $\to_{\mathcal{R}}^{*}$ along the lines of [7].

Acknowledgment

I am indebted to Lawrence Moss for suggesting the idea that language models form a category to which the usual notions of initiality and finality can be "lifted".

References

[1] ADJ—J.A. Goguen et al., Initial algebra semantics and continuous algebras, *J. Assoc. Comput. Mach.* **24** (1977) 68-95.

[2] ADJ—J.A. Goguen et al., An initial algebra approach to the specification, correctness, and implementation of abstract data types, in: R.T. Yeh, ed., *Current Trends in Programming Methodology IV* (Prentice-Hall, New York, 1978).

[3] J.A. Goguen and J. Meseguer, Initiality, induction and computability, in: M. Nivat and J.C. Reynolds, eds., *Algebraic Methods in Semantics* (Cambridge University Press, London, 1985) 459-542.

[4] C.M. Hoffmann and M.J. O'Donnell, Programming with equations, *ACM TOPLAS* **4**(1) (1982) 83-112.

[5] C.M. Hoffmann and M.J. O'Donnell, Implementation of an interpreter for abstract equations, in: *Proc. 11th POPL* (Salt Lake City, 1984) 111-121.

[6] G. Huet and J-J. Levy, Call-by-need computations in nonambiguous linear term rewriting systems, Tech. Rept. 359, INRIA (Le Chesney, France, 1979).

[7] G. Huet, Confluent reductions: abstract properties and applications to term rewriting, *J. Assoc. Comput. Mach.* **27**(4) (1980) 797-821.

[8] K.G. Larsen and G. Winskel, Using information systems to solve recursive domain equations effectively, Tech. Rept., CS Dept., Carnegie-Mellon University (Pittsburgh, 1983).

[9] M.R. Levy and T.S.E. Maibaum, Continuous data types, *SIAM J. Comput.* **11**(2) (1982) 201-216.

[10] R. Milner, Fully abstract models of typed λ-calculi, *Theoret. Comput. Sci.* **4** (1977) 1-22.

[11] R. Milner, A proposal for standard ML, in: *Proc. 1984 ACM Symp. on LISP and Functional Programming* (1984) 184-197.

[12] L. Moss and S.R. Thatte, Final semantics of languages of specifications, in: *Proc. 5th MFPS Workshop*, Lecture Notes in Computer Science (Springer, Berlin, to appear).

[13] M. Nivat, On the interpretation of recursive polyadic program schemes, *Symposia Mathematica* **15** (1975) 255-281.

[14] M.J. O'Donnell, *Computing in Systems Described by Equations*, Lecture Notes in Computer Science **58** (Springer-Verlag, New York, 1977).

[15] G. Plotkin, LCF as a programming language, in: *Proc. Conf. Program Proving and Improving* (Arc-et-Senans, 1975).

[16] J.C. Raoult, and J. Vuillemin, Operational and semantic equivalence between recursive programs, *J. Assoc. Comput. Mach.* **27**(4) (1980) 772-796.

[17] D. Scott, Domains for denotational semantics, in: *Proc. ICALP'82*, Lecture Notes in Computer Science **140** (Aarhus, Denmark, 1982) 577-613.

[18] S.R. Thatte, A refinement of strong sequentiality for term rewriting with constructors, *Informat. Comput.* **72**(1) (1987) 46-65.

[19] S.R. Thatte, Implementing term rewriting with constructor systems, *Theoret. Comput. Sci.* **61** (1988) 83-92.

[20] D.A. Turner, Miranda: A non-strict functional language with polymorphic types, in: Lecture Notes in Computer Science **201** (Springer-Verlag, New York, 1985) 1-16.

[21] C. Wadsworth, The relation between computational and denotational properties for Scott's D_∞ models of the λ-calculus, *SIAM J. Comput.* **5**(3) (1976) 488-520.

[22] C. Wadsworth, Approximate reduction and lambda calculus models, *SIAM J. Comput.* **7**(3) (1978) 337-356.