On the Power and Limitations of Strictness Analysis

R. SEKAR
Bellcore
and
P. MISHRA and I.V. RAMAKRISHNAN
SUNY at Stony Brook

Strictness analysis is an important technique for optimization of lazy functional languages. It is well known that all strictness analysis methods are *incomplete*, i.e., fail to report some strictness properties. In this paper, we provide a precise and formal characterization of the loss of information that leads to this incompleteness. Specifically, we establish the following characterization theorem for Mycroft's strictness analysis method and a natural generalization of his method to non-flat domains called ee-analysis: *Mycroft's method will deduce a strictness property for program P iff the property is independent of any constant appearing in any evaluation of P.* To prove this, we specify a small set of equations called E-axioms, that capture the information loss in Mycroft's method and develop a new proof technique called E-rewriting. E-rewriting extends the standard notion of rewriting to permit the use of reductions using E-axioms interspersed with standard reduction steps. E-axioms are a syntactic characterization of information loss and E-rewriting provides an *algorithm independent* proof technique for characterizing the power of analysis methods. It can be used to answer questions on completeness and incompleteness of Mycroft's method on certain natural classes of programs. Finally, the techniques developed in this paper provide a general principle for establishing similar results for other analysis methods such as those based on abstract interpretation. As a demonstration of the generality of our technique, we give a characterization theorem for another variation of Mycroft's method called dd-analysis.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory; D.3.2 [**Programming Languages**]: Language Classifications—*applicative languages*; D.3.4 [**Programming Languages**]: Processors—*compilers*; *optimization*

General Terms: Languages, Theory, Measurement

Additional Key Words and Phrases: Program analysis, abstract interpretation, strictness analysis, completeness

1. INTRODUCTION

Mycroft [1980] pioneered the concept of strictness analysis as a technique for optimizing lazy functional languages by transforming call-by-need to call-by-value. His results have had major impact on techniques for compilation, optimization and parallel evaluation of lazy functional languages.Mycroft's work was based on abstract interpretation of first-order functions defined over flat domains. Subsequently, there has been much research into developing strictness analysis techniques for higher-order functions [Burn et al. 1985; Hudak and Young 1986; Kuo and Mishra 1989]

Work partially supported by NSF grants CCR-8706973, CCR- 8805734 and CCR-9102159. A preliminary version of this paper was presented in POPL '91.
Authors' addresses: R. Sekar, Bellcore, Room 1J–226R, 445 South Street, Morristown, NJ 07960; email: sekar@bellcore.com; P. Mishra and I.V. Ramakrishnan, Department of Computer Science, SUNY at Stony Brook, NY 11794; email: {mishra,ram}@cs.sunysb.edu.

and non-flat domains [Hughes and Wadler 1987; Hall and Wise 1987; Sekar et al. 1990; Wadler 1987].

It is well-known that no strictness analysis method can deduce all the strictness properties of a function. For example, consider the function

$$F(x, y) = if\ P(x)\ then\ y\ else\ 0$$

where $P(x)$ is always true for all values of x. Observe that F is strict in y, but this fact is impossible to detect uniformly. Therefore, any strictness analysis method must be *incomplete*, i.e., fail to report some strictness properties. This incompleteness may arise from a variety of sources such as inability to represent all possible strictness properties, inability to accurately analyze function composition or recursion, etc. Such inaccuracy leads to some "information loss" in any strictness analysis method. Despite a decade of research into strictness analysis methods, the important problem of discovering a *precise* and *formal* characterization of this loss of information has remained open. Such results, which we call *characterization theorems*, are important because they provide an unequivocal basis for comparing different strictness analysis methods. They also enable us to pinpoint weaknesses of existing methods, improve them and aid us in discovering new and more powerful methods.

1.1 Main Results

In this paper we present the first characterization theorems for Mycroft's method and its natural generalization to non-flat domains called *ee*-analysis [Sekar et al. 1990]. Our main theorem is:
Mycroft's method will deduce a strictness property for program P iff the deduction is independent of any constant appearing in any evaluation of P.
For instance, in the function F above, deducing that F is strict in y is based on the fact that $P(x)$ always evaluates to the constant *true* and thus Mycroft's method cannot deduce this.

In Mycroft's method all constants are mapped to a single value in the abstract domain. This obviously must contribute to loss of information. However there are several other steps in the abstraction that might also contribute to information loss: the choice of abstract representations for the primitive functions, the use of composition and fix point iteration to analyze user-defined functions, etc. Our contribution is to show that the information loss in Mycroft's method is *only* due to the mapping of all constants to a single value; all other abstraction steps are lossless. An important consequence of our result is that *any* method that uniformly ignores constants cannot be any more powerful than Mycroft's method.

1.2 Overview of Approach

To state our characterization theorem formally and prove it, we needed to devise new technical ideas and proof techniques. First, we specify a small set of equational axioms (called *E-equations*) that make all constant terms *interconvertible*, thereby capturing the information loss incurred by mapping all constants to the same value. The E-equations induce an equivalence relation $=_E$ (called E-equivalence) on programs, given by $P =_E Q$ iff P can be obtained from Q by "changing its constant parts". Based on this notion of E-equivalence of programs, our main theorem can be formally stated as: *Mycroft's (and our ee-analysis) method will deduce strictness property φ of program P iff φ is a strictness property shared by* every *program $Q =_E P$.*

For illustration, the following two programs are E-equivalent since one can be obtained from the other by changing *true* to *false* or vice-versa.

$$F_1(x, y) = if\ true\ then\ x\ else\ y$$
$$F_2(x, y) = if\ false\ then\ x\ else\ y$$

Note that F_1 is strict in x, whereas F_2 is strict in y. However, neither of these properties are shared by both programs. Thus these two properties will not be inferred by Mycroft's method. On the other hand, the property $x \vee y$ *is* shared by both and hence will be inferred by Mycroft's method.

For the *only if* part of the characterization theorem we show that if $P =_E Q$ then we compute the same strictness properties for P and Q. For establishing the *if* part we need to show that for any strictness property φ of P that is *not* inferred by Mycroft's method, there exists a $Q =_E P$ for which φ is *not* a strictness property. The construction of such a Q is quite complex and cumbersome. Therefore we develop a new proof technique called E-rewriting that precisely formalizes the notion of performing reductions with any $Q =_E P$. E-rewriting extends the standard notion of rewriting to permit the use of equational steps interspersed with standard reduction steps from P. Thus constants can be interchanged at any point in the reduction sequence. Finally, we show that there is no loss of information due to composition and recursion, other than that captured by E-rewriting.

E-equations are a syntactic characterization of information loss and E-rewriting provides an *algorithm independent* proof technique for characterizing the power of strictness analysis. (By this we mean that if we want to know whether an analysis procedure will compute a strictness property, we need not know its computation procedure at all. We need only check if the strictness property is preserved if the program is modified as permitted by the E-axioms, e.g., changing constants in Mycrofts and *ee*-analysis method.) It can be used to answer questions of completeness of such methods with respect to classes of programs. In contrast, it appears difficult to deal with such questions using the setup of abstract interpretation. Section 5 describes in detail our completeness results based on E-rewriting.

The techniques developed in this paper provide a general principle for establishing similar characterization theorems for many other analysis methods. As a demonstration of the generality of our technique, we have been able to give a characterization theorem for another variation of Mycroft's method called *dd*-analysis. *dd*-analysis is an enhancement of Mycroft's method to reason about head normal forms in non-flat domains. Among other applications, *dd*-analysis has been shown to be a valuable tool for repairing violations of strong sequentiality, a concept pioneered by Huet and Levy [1979] for lazy evaluation of equational programs.

Since the preliminary version of this paper appeared in POPL '91, much research [?; ?] has taken place in the area of completeness of abstract-interpretation-based analyses. In Section 6 we provide a detailed comparison between our approach and the approaches proposed in these works.

The rest of this paper is organized as follows: Section 2 is an overview of the

ee-analysis method. Technical details of E-rewriting and the main theorem appear in section 3. In section 4 we give a characterization theorem for dd-analysis. Implications and significance of our results appear in section 5 together with remarks on its generality. Finally relationship with other research is dicussed in Section 6.

2. PRELIMINARIES

The language we consider is a constructor-based term rewriting system (TRS). Such a system consists of a set of oriented equations (called *rewrite rules*) of the form $l \rightarrow r$ where the left-hand side (lhs) l and right-hand side (rhs) r are terms that may contain variables. The lhs must be *linear*, i.e., no variable in them can occur more than once. Furthermore, the nonvariable symbols are partitioned into two disjoint sets – the set of function symbols $\mathcal{F}$, and the set of constructor symbols $\mathcal{C}$, such that the outermost symbol of every lhs is in $\mathcal{F}$ while the rest are in $\mathcal{C}$. First-order programs written in functional languages, as well as the recursion equations studied by Mycroft, follow this discipline.

By $t[x \leftarrow v]$ we refer to the term obtained from t by substituting the variable x with the term v. If u is a term with n variables $x_1, \ldots, x_n$ then $u[t_1, \ldots, t_n]$ is shorthand for $u[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$. A *substitution* β maps variables to terms. An *instance* $t\beta$ of a term t is obtained by replacing x by $\beta(x)$ for each variable x in t. A *redex* in term u is an occurrence of an instance of an lhs. A term t is in *normal form* (NF) if it contains no redexes.

By $t \rightarrow u$ we mean that u is obtained by replacing a redex $l\beta$ in t by $r\beta$, where $[l \rightarrow r]$ is a rewrite rule. The reflexive and transitive closure of $\rightarrow$ is denoted by $\rightarrow^*$. We use the notations $t \downarrow \mathrm{NF}$ and $t \uparrow \mathrm{NF}$ to denote respectively that t can or cannot be reduced to a term in NF. A term t is in *head normal form* (HNF) if there is no redex t' such that $t \rightarrow^* t'$.

2.1 Overview of ee-analysis

Our characterization theorem is based on ee-analysis which is a strictness analysis method for non-flat domains. This analysis tells us which subterms of a term are to be normalized in order to normalize the entire term. We review this method and remark that it reduces to Mycroft's method for programs on flat domains. Therefore our results for ee-analysis directly apply to Mycroft's method as well. First we define strictness property of a term t as follows.

Definition (ee-Strictness). Let t be a term, $\varphi = (x_{11} \wedge \cdots \wedge x_{1r_1}) \vee (x_{21} \wedge \cdots \wedge x_{2r_2}) \vee \cdots \vee (x_{n1} \wedge \cdots \wedge x_{nr_n})$, with $x_{ij} \in \mathcal{V}(t)$. Then φ is an ee-strictness property of t iff, in every reduction sequence that normalizes any instance $t\beta$ of t, all the terms in one of the sets $\{\beta(x_{11}), ..., \beta(x_{1r_1})\}, ..., \{\beta(x_{n1}), ..., \beta(x_{nr_n})\}$ are normalized.

For instance, the ee-strictness of *if* is $(x_1 \wedge x_2) \vee (x_1 \wedge x_3)$, which means that in order to normalize $if(t_1, t_2, t_3)$, we have to normalize either t_1 and t_2 or t_1 and t_3.

We use the notation t^{ee} to denote the ee-strictness of a term t. The strictness of arbitrary terms is computed by *composing* the strictness of constructors and function symbols using the following rule.

Definition (*Rule of Substitution*). Let t be a term, $x_1, ..., x_n$ be the variables in t and β be a substitution for these variables. Then

$$[t\beta]^{ee} = t^{ee}[x_i \leftarrow [\beta(x_i)]^{ee}].$$

To illustrate this rule, consider the term $t = f(x', x'')$ and the substitution $\beta(x') = x, \beta(x'') = g(y, z)$. The rhs of the above rule becomes $f^{ee}[x' \leftarrow x, x'' \leftarrow g^{ee}(y, z)]$. Suppose that $f^{ee}(x', x'') = x' \vee x''$, and $g^{ee}(y, z) = y \vee z$. Then the rhs would simplify to yield the strictness of $t\beta = f(x, g(y, z))$ as $x \vee (y \vee z)$.

Now we proceed to the equations to compute x^{ee}, c^{ee}, and f^{ee}, so that, together with the above rule of substitution, we can compute strictness of arbitrary terms. For the variable case, its substitution is to be normalized, and hence $x^{ee} = x$. For the case $c(x_1, ..., x_n)$, the NF of its instance $c(s_1, ..., s_n)$ is obtained only by normalizing each of $s_1, ..., s_n$. Hence c^{ee} is the conjunction of the x_i. Note that, by this definition, if the arity of c is zero then $c^{ee} = true$. This discussion yields the following two equations:

$$\begin{aligned} x^{ee} &= x && (1)\\ [c(x_1, ..., x_n)]^{ee} &= x_1 \wedge \cdots \wedge x_n && (2) \end{aligned}$$

For the case when the root symbol is a function symbol f, assume that f is defined by the rules:

$$\begin{aligned} f(t_1^1, ..., t_1^n) &\rightarrow e_1\\ \vdots\; &\;\vdots\; \vdots\\ f(t_k^1, ..., t_k^n) &\rightarrow e_k \end{aligned}$$

Observe that any instance $f(s_1, ..., s_n)$ (of $f(x_1, ..., x_n)$) can be rewritten into only one of $e_1, ..., e_k$. Therefore f^{ee} is a disjunction of e_j^{ee}, for $j = 1, ..., k$. However, f^{ee} is a formula on $x_1, ..., x_n$, and these variables differ from those in e_j (which are actually variables in $t_j^1, ..., t_j^n$). Hence we need a function g_j that projects the demand on the variables in $t_j^1, ..., t_j^n$ into a demand on $x_1, ..., x_n$. This leads us to the following equation for defining f^{ee}.

$$[f(x_1, ..., x_n)]^{ee} = \bigvee_{j=1}^{k} g_j(e_j^{ee}) \qquad (3)$$

The equations to define g_j are as follows. These equations are to be applied in the order in which they are given. In this definition, φ and ψ stand for arbitrary boolean formulae in disjunctive normal form (DNF), and φ with a subscript stands for a conjunction.

$$\begin{aligned} g_j(\psi \vee \varphi) &= g_j(\psi) \vee g_j(\varphi) && (4)\\ g_j(\varphi_1) &= \left(\bigwedge_{i \mid t_j^i \text{ contains no variable}} x_i \right) \wedge g_j(\varphi_1) && (5)\\ g_j(x_{i1} \wedge \cdots \wedge x_{ir_i} \wedge \varphi_1) &= x_i \wedge g_j(\varphi_1) \qquad \text{if } \mathcal{V}(t_j^i) = \{x_{i1}, ..., x_{ir_i}\} && (6)\\ g_j(\varphi_1) &= true, \text{ otherwise.} && (7) \end{aligned}$$

The justification for the definition of g_j is as follows. Eq. (4) simply "pushes" the g_j operations into individual conjunctions. For Eqs. (5) through (7), note that if $f(s_1, ..., s_n)$ is rewritten by the jth equation then it must be the case that $s_i = t^i_j \beta$, for some β. If t^i_j contains variables $x_{i1}, ..., x_{ir_i}$, and all these are e-demanded to normalize e_j, then s_i must be normalized. This is because all nonvariable symbols in t^i_j are constructors, and so s_i will be in NF if each of $\beta(x_{i1}), ..., \beta(x_{ir_i})$ are in NF. Hence x_i appears in $g_j(e^{ee}_j)$ if $x_{i1} \wedge \cdots \wedge x_{ir_i}$ appears in e^{ee}_j (and hence (6)). By the same argument, if t^i_j consists entirely of constructors (no variables), s_i must be normalized to match it, and hence x_i is e-demanded (and hence (5)). Otherwise we cannot conclude an e-demand on x_i, and thus it does not appear in f^{ee} (and hence (7)).

The above equations directly compute f^{ee} if f is nonrecursive. In case of recursive functions we use the following fixpoint iteration procedure. Let $f_1, ..., f_m$ denote all functions in the program. For $i = 1, ..., m$, define

$$
\begin{aligned}
f_i^{ee,0} &= \bigwedge_{j=1}^{arity(f_i)} x_j \\
f_i^{ee,r+1} &= \bigvee_{j=1}^{k_i} g_j\left(e_j^{ee}[f_l^{ee} \leftarrow f_l^{ee,r}]\right), \text{ for } l = 1, ..., m
\end{aligned}
$$

where k_i is the number of rules defining f_i. The limit of the sequence $f^{ee,0}, f^{ee,1}, ...$ is defined to be f^{ee}_i.

We illustrate ee-analysis using the following program:

Example 1.
$$
\begin{aligned}
f(nil, x, y) &\rightarrow x + y \\
f(x : nil, y, z) &\rightarrow f(nil, y, x) \\
f(u : w, y, z) &\rightarrow f(w, y, z)
\end{aligned}
$$

We begin with $f^{ee,0} = 0$. Note that $e_1^{ee,0} = x \wedge y$ (by definition of $+^{ee}$) and $g_1(x \wedge y) = x_1 \wedge g_1(x \wedge y)$ (by (5)), which can then be simplified to $x_1 \wedge x_2 \wedge x_3$ using 2 applications of (6). It can be easily seen that $e_2^{ee,0} = e_3^{ee,0} = 0$ and so $f^{ee,1} = x_1 \wedge x_2 \wedge x_3$. Substituting this, we get $g_1(e_1^{ee,1}) = x_1 \wedge x_2 \wedge x_3$. Observe that $e_2^{ee,1} = x \wedge y$ and so $g_2(e_2^{ee,1}) = x_1 \wedge g_2(y) = x_1 \wedge x_2$ (by two applications of (6)). Also note $g_3(e_3^{ee,1}) = g_3(w \wedge y \wedge z)$ which can be reduced to $x_2 \wedge x_3 \wedge g_3(w)$ (two applications of (6)) and then to $x_2 \wedge x_3$ (by 7). Therefore $f^{ee,2} = x_2 \wedge (x_1 \vee x_3)$. Following the same procedure, we arrive at $f^{ee,3} = x_2 = f^{ee,4}$. Thus, f is ee-strict only in its second argument.

To derive Mycroft's method from ee-analysis, observe that each function in his recursion equation notation is defined using a single rewrite rule. Therefore the disjunction in rule (3) is unnecessary. Furthermore, since these equations have no constructors on the lhs (no pattern matching), it can be easily shown that the g_j's reduce to identity function. Finally, note that some of the primitive functions such as the conditional, used by Mycroft can be defined using pattern matching. The strictness properties derived by our analysis for such functions are exactly those assumed by Mycroft.

3. CHARACTERIZATION THEOREM FOR EE-ANALYSIS

3.1 Alternative Definition of Strictness Property

We start this section with an alternative definition of strictness property that is used in our proofs. For this definition, we need to relate substitutions and truth assignments as follows. (In the following, we use the notations $t \downarrow \mathrm{NF}$ and $t \uparrow \mathrm{NF}$ to denote respectively that t can or cannot be reduced to a term in NF.) Let $\mathcal{A}$ be a (truth assignment) function assigning $true$ or $false$ to $x_1, ..., x_n$ and σ be a (substitution) function assigning terms to $x_1, ..., x_n$. We say $\sigma \leq_c \mathcal{A}$ iff $\forall 1 \leq i \leq n$ $[\sigma(x_i) \downarrow \mathrm{NF} \Rightarrow \mathcal{A}(x_i)]$. In other words, $\mathcal{A}$ assigns $true$ to a variable whenever σ substitutes a term possessing NF in its place. Similarly, $\sigma \geq_c \mathcal{A}$ means that $\forall i, \mathcal{A}(x_i) \Rightarrow \sigma(x_i) \downarrow \mathrm{NF}$.

Definition (*Strictness Property II*). φ is a strictness property of t iff $\forall \mathcal{A}\ \neg\mathcal{A}(\varphi) \Rightarrow \forall \sigma \leq_c \mathcal{A}, t\sigma \uparrow \mathrm{NF}$.

Note that this definition is a compact version of Mycroft's original definition (pp.275 in [Mycroft 1980]) of strictness property. Now we show that the two definitions of strictness property are equivalent.

Proof of equivalence of Strictness Properties I and II. To show that definition 1 implies definition 3, we proceed as follows. Let $\mathcal{A}(\varphi)$ be false. Then $\mathcal{A}$ must assign false to one literal in each conjunct in the DNF of φ. Therefore every $\sigma \leq_c \mathcal{A}$, must substitute a term with no NF for one literal in every conjunct. From the contrapositive of definition 1, the consequent $t\sigma \uparrow \mathrm{NF}$ of definition 3 follows. To show that definition 3 implies 1, assume that the antecedent of the contrapositive of definition 1 holds (i.e., $\forall j\ \exists i\ \beta(x_{ij}) \uparrow \mathrm{NF}$). Now let $\mathcal{A}(x_{ij}) = true$ iff $\beta(x_{ij}) \downarrow \mathrm{NF}$. Note that $\mathcal{A}(\varphi) = false$, $\beta \leq_c \mathcal{A}$ and hence by definition 3, $t\beta \uparrow \mathrm{NF}$. □

3.2 Maximal Strictness Property

The set of strictness properties forms a lattice and hence every term t has a maximal strictness property, which is defined as follows.

Definition (*Maximal Strictness Property*).
A strictness property φ of t is said to be maximal if either of the following (equivalent) conditions hold.
1. If ψ is a strictness property for t then $\varphi \Rightarrow \psi$.
2. $\forall \mathcal{A}\ \mathcal{A}(\varphi) \Rightarrow \exists \sigma \leq_c \mathcal{A},\ t\sigma \downarrow \mathrm{NF}$.

In the following example, x_2, $x_2 \vee x_1$ and $x_1 \vee x_2 \vee x_3$ are all strictness properties of f whereas x_2 is its maximal strictness property.

Example 2.

$$\begin{aligned} f(nil, x, y) &\rightarrow x + y \\ f(x : nil, y, z) &\rightarrow f(nil, y, x) \\ f(u : w, y, z) &\rightarrow f(w, y, z) \end{aligned}$$

Proof of equivalence of the conditions in Definition 4. Henceforth we assume that φ is in DNF. To show that (2) implies the (contrapositive) of (1), pick any arbitrary ψ. If $\psi = \varphi$ then (1) holds. Otherwise, we can find an $\mathcal{A}$ such that $\mathcal{A}(\varphi) = true$ and $\mathcal{A}(\psi) = false$. For this $\mathcal{A}$, there is a $\sigma \leq_c \mathcal{A}$, such that $t\sigma \downarrow \mathrm{NF}$ (by 2). Note that $t\sigma \downarrow \mathrm{NF}$ although $\mathcal{A}(\psi) = false$ and hence ψ is not a strictness property of t.

Now we show that $\neg 2 \Rightarrow \neg 1$ (i.e., $1 \Rightarrow 2$). Assuming $\neg 2$, there is an $\mathcal{A}$ such that $\mathcal{A}(\varphi) = true$ and $\forall \sigma \leq_c \mathcal{A},\ t\sigma \uparrow \mathrm{NF}$. Since $\mathcal{A}(\varphi) = true$, there is a conjunct φ_1 of φ such that $\mathcal{A}(\varphi_1) = true$. Define $\mathcal{A}'(x_i) = true$ iff x_i is in φ_1. Note $\forall \sigma[\sigma \leq_c \mathcal{A}' \Rightarrow \sigma \leq_c \mathcal{A},]$ and so $\forall \sigma \leq_c \mathcal{A}'\ t\sigma \uparrow \mathrm{NF}$.

Let φ_2 be the disjunction of all the remaining clauses in φ other than φ_1. Let $\psi = \varphi_2 \vee [\bigvee_{x|x \in t, x \notin \varphi_1} (\varphi_1 \wedge x_i)]$. Note $\psi \Rightarrow \varphi$. We now show that ψ is a strictness property of t, thus contradicting 1. To do this, consider any $\bar{\mathcal{A}}$ such that $\bar{\mathcal{A}}(\psi) = false$. If $\bar{\mathcal{A}}(\varphi)$ is also false, then since φ is a strictness property, we have $\forall \sigma \leq_c \bar{\mathcal{A}}\ \ t\sigma \uparrow \mathrm{NF}$. Otherwise, (i.e., $\bar{\mathcal{A}}(\psi) = false, \bar{\mathcal{A}}(\varphi) = true$) it must be the case that $\bar{\mathcal{A}}(\varphi_1) = true$ and $\varphi_1 \wedge x = false$ (for all $x \notin \varphi_1$, x in t). This means that $\bar{\mathcal{A}}(x) = true$ iff $x \in \varphi_1$ and hence $\bar{\mathcal{A}} = \mathcal{A}'$. Since we already know that $\forall \sigma \leq_c \mathcal{A}'\ t\sigma \uparrow \mathrm{NF}$, we have established that ψ is a strictness property of t. □

3.3 E-axioms and E-Rewriting

We introduce the following equations (henceforth called E-axioms) to capture the notion that we cannot distinguish between constants in ee-analysis.

$$x = c(c_1, ..., c_i, x, c_{i+1}, ..., c_n), \quad \forall c, c_1, ..., c_n \in \mathcal{C} \tag{8}$$
$$c = c', \quad \forall c, c' \in \mathcal{C} \tag{9}$$

Note that a term $c(c_1, ..., c_i, \bar{c}, ..., c_n)$ consisting entirely of constructors can be built from $\bar{c}$ using equation (8). By repeating this process, we can build any such constructor term containing $\bar{c}$ starting with $\bar{c}$. All such terms are equal. Since any two zeroary constructors are equal by equation (9), any two constructor terms are made equal by these equations.

We say $t =_E s$ whenever t can be obtained from s using the above two equations. This relation can be naturally lifted to rewrite rules and programs as follows: $l \to r =_E l' \to r'$ iff $l =_E l'$ and $r =_E r'$. For programs P and Q, $P =_E Q$ iff for every rule $l \to r$ in P there is a rule $l' \to r' =_E l \to r$ in Q and vice versa. Henceforth P and Q stand for programs.

Finally we define E-derivation $t_0 \to t_1 \to \cdots \to t_n$ where each t_i is obtained from t_{i-1} by applying one of the rewrite rules in the program or one of the above equations. We call the reduction step a P-step (denoted $t_i \to_P t_{i+1}$) in the former case and an E-step (denoted $t_i \to_E t_{i+1}$) in the latter case. If it is a P or an E-step then we denote it as $t_i \to_{(P,E)} t_{i+1}$. Note that in an E-step equation (8) can be applied either right-to-left, denoted ($8r$), or left-to-right, denoted ($8l$). We say that a term t is in E-normal form iff it contains no function symbols and use the notation $s \downarrow E\text{-NF}$ to denote that s possesses an E-NF.

As noted in the introduction, E-rewriting formalizes the notion that constants can be interchanged at any point in the reduction sequence and therefore we achieve the effect of performing reductions w.r.t. to any $Q =_E P$. Formally,

THEOREM 1. *$t \to^*_{(P,E)} s \Leftrightarrow \exists t' =_E t, s' =_E s, Q =_E P$ such that $t' \to^*_Q s'$*

PROOF. By definition of E-equivalence of programs, a Q-reduction step $l'\sigma \to r'\sigma$ (here $l' \to r' \in Q$) is equivalent to a (P,E)-reduction sequence $l'\sigma \to_E^* l\sigma \to_P r\sigma \to_E^* r'\sigma$ (where $l \to r \in P$). Therefore every Q-derivation can be viewed as a special type of (P,E)-derivation and hence the proof in ($\Leftarrow$) direction. For the proof in the other direction, we need to transform an arbitrary (P,E)-derivation sequence into one of this special type. To do such a transformation requires rearranging the order in which the E-steps and P-steps are performed. It is straightforward to rearrange a sequence consisting of an E-step followed by a P-step (or P-step followed by a E-step) into one consisting of zero or more P-steps followed by zero or more E-steps (or zero or more E-steps followed by zero or more P-steps). However, simply repeating this process may create arbitrarily many new E or P-steps and so this process of rearrangement may not terminate. Therefore a sophisticated analysis of the interaction between the E and P-steps is necessary to construct the desired reduction sequence. We first present an outline of such a construction below:

1. Let $t_k \to t_{k+1}$ be the first application of (8r) in the sequence $t \to_{(P,E)}^* s$. We show that either this reduction and the one immediately preceding it can be interchanged, or that the preceding equation uses a program rule $l \to r$ and the effect of performing these two reductions can be achieved by performing one reduction using a rule $l \to r'$ (here $r =_E r'$).

2. We repeat step 1 till the reduction using (8r) rule "disappears" or is moved ahead of all other reductions.

3. Repeat step 2 on the next application of (8r) rule and so on. At this point, we have a reduction sequence which can be partitioned into two parts. The initial part $t \to^* \bar{t}$ contains a sequence of reductions using (8r) rules alone. The latter part contains reductions using (8l), (9) and rules of the form $l \to r'$, where $l \to r$ is a rule in P and $r =_E r'$. Let $t' = \bar{t}$.

4. In a manner analogous to the steps above, we "push" all applications of rules $l \to r'$ (where $l \to r$ is a rule in P and $r =_E r'$) ahead of all applications of (8l) and (9) steps.

Now we have constructed the reduction sequence $t \to_{(8r)}^* t' \to_Q^* s' \to_{(8l,9)}^* s$. Note that Q consists only of rules $l' \to r'$ that are E-equivalent to some rule $l \to r$ in P and thus $Q =_E P$. We now fill in the details of the proof by providing the details of each of the above four proof steps:

Step 1 Let $t_{k+1} \to t_{k+2}$ be the first reduction step that uses the (8l) rule and let $l \to r$ be the rule used in the reduction $t_k \to t_{k+1}$ immediately preceding it. Noting that t_{k+1} must be of the form $u[r\sigma]$, we have the following cases depending upon whether the (8l) rule is applied inside u (case 1a) or inside σ (case 1b) or inside r (case 1c).

Case 1a: Let $u = u_1[u_2]$ where u_1 is a term at whose root the reduction using (8l) is performed. Now, the following diagram illustrates how the order of these reductions can be interchanged.

$$\begin{tikzcd}
t_k = u_1[u_2[l\sigma]] \arrow[r, "l \rightarrow r"] \arrow[d, "(8l)"'] & u_1[u_2[r\sigma]] = t_{k+1} \arrow[d, "(8l)"] \\
u_1[c(c_1, ..., c_i, u_2[l\sigma], ..., c_n)] \arrow[r, "l \rightarrow r"] & u_1[c(c_1, ..., c_i, u_2[r\sigma], ..., c_n)] = t_{k+2}
\end{tikzcd}$$

<u>*Case 1b:*</u> In this case, (8l) must have been applied inside some subterm $s = \sigma(x)$ for a variable x in r and must have rewritten s into some s'. Thus $t_k = u[l[x \leftarrow s, ...]] \rightarrow u[r[x \leftarrow s, ...]] = t_{k+1}$. Let $s_1, ..., s_n$ be the other subterms that are substitutions for x (i.e., the variable x occurs $n+1$ times in r). So t_{k+1} is of the form $\bar{u}[s, s_1, ..., s_r]$ and $t_{k+2} = \bar{u}[s', s_1, ..., s_n]$. We can interchange the order of these reductions as follows: $t_k = u[l[x \leftarrow s, ...]] \rightarrow_{(8l)} u[l[x \leftarrow s', ...]] \rightarrow_{l \rightarrow r} u[r[x \leftarrow s']] = \bar{u}[s', s'_1, ...] = t'_{k+1}$. Now t'_{k+1} can be reduced to t_{k+2} by n applications of (8r) rules to reduce each s'_i back to s_i.

<u>*Case 1c:*</u> Note that in this case $l \rightarrow r$ cannot itself be a (8l) rule since the first application of (8l) rule was in $t_{k+1} \rightarrow t_{k+2}$. It cannot be a (8r) rule since in that case $r = x$ and we cannot apply (8l) rule within r. If it is the (9) rule, then we have the following diagram.

$$\begin{tikzcd}
t_k = v[c'] \arrow[r, "(9)"] \arrow[d, "(8l)"'] & v[c''] = t_{k+1} \arrow[d, "(8l)"] \\
v[c(c_1, ..., c_i, c', ..., c_k)] \arrow[r, "(9)"] & t_{k+2} = v[c(c_1, ..., c_i, c'', ..., c_k)]
\end{tikzcd}$$

Otherwise, if it is a program rule, we have the following diagram.

$$\begin{tikzcd}
t_k = v[l\sigma] \arrow[r, "l \rightarrow r"] \arrow[dr, "l \rightarrow r'"'] & v[r\sigma] \arrow[d, "(8l)"] \\
 & v[r'\sigma]
\end{tikzcd}$$

This completes the proof of step 1.

<u>**Step 4**</u> Let $t_{k+1} \rightarrow t_{k+2}$ be the *first* application of a program rule $l \rightarrow r$. Therefore $t_{k+1} = u[l\sigma]$ and we again have three cases to consider depending upon whether the preceding reduction (using (8r) or (9) was performed within u (case 4a) or within σ (case 4b) or within l (case 4c).

<u>*Case 4a:*</u> Note that this case is not possible if the rule applied in the step

$t_k \to t_{k+1}$ was a (9) rule and so it must have been a (8r) rule. The following diagram illustrates how the two reductions can be interchanged.

$$\begin{tikzcd}
t_k = u[c(c_1, \ldots, c_i, s[l\sigma], \ldots, c_n)] \arrow[r, "(8r)"] \arrow[d, "l \to r"'] & u[s[l\sigma]] = t_{k+1} \arrow[d, "l \to r"] \\
u[c(c_1, \ldots, c_i, s[r\sigma], \ldots, c_n)] \arrow[r, "(8r)"] & u[s[r\sigma]] = t_{k+2}
\end{tikzcd}$$

Case 4b: In this case, the (8r) or (9) rule must have been applied within some subterm s_1 and must have rewritten into s'_1, where s'_1 is a substitution for a variable x in l. Assume that x occurs n times in r and let $s'_2, \ldots, s'_n$ denote the substitutions for these occurrences in t_{k+2}. So $t_{k+2} = \bar{u}[s'_1, \ldots, s'_n]$. We can now interchange the order of these reductions as follows:
$t_k = u[l[x \leftarrow s_1, \ldots]] \to_{l \to r} u[r[x \leftarrow s_1, \ldots]] = t'_{k+1} = \bar{u}[s_1, \ldots, s_n]$. We can now apply rule (8r) on subterms $s_1, \ldots, s_n$ to reduce t'_{k+1} into t_{k+2}.

Case 4c: The following diagram shows how we can get another program rule $l' \to r =_E l \to r$ that achieves the effect of both these reductions.

$$\begin{tikzcd}
t_k = u[l'\sigma] \arrow[r, "(8r)"] \arrow[dr, "l' \to r"'] & u[l\sigma] = t_{k+1} \arrow[d, "l \to r"] \\
 & u[r\sigma] = t_{k+2}
\end{tikzcd}$$

3.4 Shared Strictness Property and Main Theorem

We now require notation to denote strictness properties of a term w.r.t. different programs. We denote the strictness property φ of t w.r.t. P as $\varphi(P)$, where P may be dropped when it is irrelevant or obvious from context.

Definition (Shared Strictness Property).
φ is a shared strictness property of t w.r.t. P iff either of the following (equivalent) conditions hold:
1. φ is a strictness property of every term $s =_E t$ w.r.t every program $Q =_E P$.
2. $\forall \mathcal{A} \neg \mathcal{A}(\varphi) \Rightarrow \forall \sigma \leq_c \mathcal{A}, t\sigma \uparrow E$-NF.

Equivalence of the above two conditions follow directly from theorem 1. Now we are ready to present our main theorem.

THEOREM MAIN THEOREM. *$t^{ee}(P) \Rightarrow \psi$ iff ψ is a shared strictness property of t w.r.t P.*

The main part of the proof effort is in establishing the if part of the above theorem. The proof of only if part is fairly easy and is based on the concept of strictness properties of two terms s and t being syntactically identical (denoted $s^{ee} \equiv t^{ee}$), i.e., these two formulas are identical even without expanding out the strictness property of some subterms in s and t. Consider for example, the terms $f(x)$ and $f(c(x))$, where $f \in \mathcal{F}$ and $c \in \mathcal{C}$. Note $[f(c(x))]^{ee} \equiv (f(x_1))^{ee}[x_1 \leftarrow [c(x)]^{ee}] \equiv (f(x_1))^{ee}(x_1 \leftarrow x) \equiv [f(x)]^{ee}$.

Lemma 3. *(a)* $\forall s, t,\ \ s =_E t \Rightarrow s^{ee} \equiv t^{ee}$.
(b) $\forall P, Q, \forall t,\ \ P =_E Q \Rightarrow t^{ee}(P) \equiv t^{ee}(Q)$.

Proof. For part (a), we show that s^{ee} and t^{ee} are syntactically identical whenever s can be obtained from t by one application of an E-axiom. By induction on number of applications of E-axioms required to get t from s, part (a) of the lemma follows. Let $t = t'[x \leftarrow l\sigma]$ and $s = t'[x \leftarrow r\sigma]$ where $l = r$ is the E-axiom applied. Note that $t^{ee} \equiv t'^{ee}[x \leftarrow (l\sigma)^{ee}]$ and $s^{ee} \equiv t'^{ee}[x \leftarrow (r\sigma)^{ee}]$. Observe that for both E-axioms, $(l\sigma)^{ee} \equiv (r\sigma)^{ee}$ and so $s^{ee} \equiv t^{ee}$.

For part (b), it suffices to show that the expressions obtained for t^{ee} (using the rule of substitution and rules (1), (2) and (3) in section 2.1) are syntactically identical for programs P and Q. Since the expression for t^{ee} are obtained from f^{ee} and the rule of substitution, $t^{ee}(P)$ and $t^{ee}(Q)$ will be identical whenever $f^{ee}(P)$ and $f^{ee}(Q)$ are identical for all functions f. Let $l_1 \to r_1, ..., l_n \to r_n$ and $l'_1 \to r'_1, ..., l'_m \to r'_m$ be the rules defining f in P and Q respectively. By definition, $f^{ee}(P) = \bigvee_{j=1}^{n} g_j(r_j^{ee}(P))$. Note that $\forall j$ there is an $l'_i \to r'_i =_E l_j \to r_j$ in Q. By part (a) of this theorem, $r_j^{ee} \equiv r_i'^{ee}$. Also observe that the equations in E do not change the variables in $t_1^j, ..., t_n^j$ (recall $l_j = f(t_1^j, ..., t_n^j)$) and so g_j and g'_i are identical. Furthermore, note that for each $l_j \to r_j \in P$ there is an $l'_i \to r'_i \in Q$ and hence each disjunct in the expression for $f^{ee}(P)$ also appears in $f^{ee}(Q)$. By interchanging P and Q in the above argument, we see that the converse of this also holds and therefore $f^{ee}(P) \equiv f^{ee}(Q)$. □

Thus we have established the only if part of our main theorem. The proof of the if part is quite difficult. We first present an outline of our approach to this part of the proof and then present a detailed proof.

3.5 Approach to Proof of Main Theorem

Our proof strategy is to show that ee-analysis infers the maximal shared strictness property. In other words, if $t^{ee}(P) = \varphi$ then we wish to show (cf. definition 4):

$$\forall \mathcal{A}\ [\mathcal{A}(\varphi) \Rightarrow \exists \sigma \leq_c \mathcal{A},\ \ t\sigma \downarrow E\text{-NF}] \tag{10}$$

We call a strictness property that satisfies (10) as an E-maximal strictness property (abbreviated EMSP). For the purposes of proof we choose to show that $t^{ee}(P) = \varphi$ satisfies the stronger property:

$$\forall \mathcal{A}\ [\mathcal{A}(\varphi) \Rightarrow \forall \sigma \geq_c \mathcal{A},\ \ t\sigma \downarrow E\text{-NF}] \tag{11}$$

Note that stronger property (11) implies (10). We now show that (11) also implies the if part of theorem 2. Assume (11) and that ψ is a shared strictness property. By (11), if $\mathcal{A}(\varphi = t^{ee}(P)) = true$ then $\forall \sigma \geq_c \mathcal{A},\ \ t\sigma \downarrow E$-NF. From the contrapositive of the second condition in definition 5, it follows that $\mathcal{A}(\psi) = true$ and hence we have $t^{ee}(P) \Rightarrow \psi$.

For the proof of (11) we need the concept of *depth* of a reduction sequence $t\sigma = t_0 \rightarrow^* t_n$ w.r.t. t. Label all occurrences of function symbols in t (exclude function symbols within σ) with 0. Let t_{i+1} be obtained from t_i using the rule $l \rightarrow r$. If the root of the redex is labelled (say, with j), then label all occurrences of function symbols in r with $j+1$ before rewrite. If the root of the redex is not labelled then proceed as usual. Then the depth of $t_0 \rightarrow \cdots \rightarrow t_n$ is the maximum among all the labels assigned to any function symbol in any of $t_1, ..., t_n$. We use the notation $s \downarrow_{k,t} E$-NF to denote that s can be E-normalized using a reduction sequence of depth $\leq k$ w.r.t. t; analogously, $s \downarrow_{k,t}$ NF can be similarly defined.

Now recall the fix point computational method used in ee-analysis. Just as f^{ee} specifies all the arguments that must have an NF whenever $f(s_1, \ldots, s_n) \downarrow$ NF, we remark that $f^{ee,k}$ specifies all the arguments that must have an NF whenever $f(s_1, \ldots, s_n) \downarrow_{k,f(x_1,\ldots,x_n)}$ NF. In Example 2, if $f(s_1, s_2, s_3) \downarrow_{1,f(x_1,x_2,x_3)}$ NF then s_1, s_2, s_3 must have NF; if $f(s_1, s_2, s_3) \downarrow_{2,f(x_1,x_2,x_3)}$ NF then s_1 and one of s_2 or s_3 must have NF as so on. The above remark implies that $f^{ee,k}$ are maximal in some sense and motivates the following definition of a *k-maximal property* φ of t.

$$\forall \mathcal{A}\ [\mathcal{A}(\varphi) \Rightarrow \forall \sigma \geq_c \mathcal{A},\ \ t\sigma \downarrow_{k,t} E\text{-NF}] \tag{12}$$

Observe that property (12) implies (11). We show that $t^{ee,k}$ is a k-maximal property. Then, by soundness of ee-analysis, t^{ee} is a strictness property of t that satisfies (11) and hence is an EMSP.

3.6 Proof of Main Theorem

We first show that the rule of substitution preserves k-maximal properties.

Lemma 4. *Let $t_1, ..., t_n$ be arbitrary terms and $\varphi_1, ..., \varphi_n$ be the respective k-maximal properties. Also let ψ be a k-maximal property of $d(x_1, ..., x_n)$, where $d \in \mathcal{C} \cup \mathcal{F}$. Then $\overline{\psi} = \psi[x_i \leftarrow \varphi_i]$ is a k-maximal property of $d(t_1, ..., t_n)$.*

Proof. The theorem is proved by showing in three steps that

$$\forall \bar{\mathcal{A}}[\bar{\mathcal{A}}(\bar{\psi}) \Rightarrow \forall \sigma \geq_c \bar{\mathcal{A}}\ \ d(t_1, ..., t_n)\sigma \downarrow_{k,d(t_1,\ldots,t_n)} E\text{-NF}]$$

1. Given $\bar{\mathcal{A}}$ we construct an $\mathcal{A}$ such that $\mathcal{A}(\psi) = \bar{\mathcal{A}}(\bar{\psi})$.

2. Given a $\sigma \geq_c \bar{\mathcal{A}}$, we construct a β such that $d(t_1, ..., t_n)\sigma$ can be reduced to $d(x_1, ..., x_n)\beta$ by an E-derivation whose depth w.r.t. $d(t_1, ..., t_n)$ is $\leq k$. We also show that $\beta \geq_c \mathcal{A}$.

3. Since ψ is a k-maximal property of $d(x_1, ..., x_n)$ and since $\mathcal{A}(\psi) = true$, for every $\beta' \geq_c \mathcal{A}$, there exists an E-derivation of depth $\leq k$ (w.r.t. $d(x_1, ..., x_n)$) to normalize $d(x_1, ..., x_n)\beta'$. We select a suitable β' and from such a derivation for $d(x_1, ..., x_n)\beta'$, we construct another E-derivation to normalize $d(x_1, ..., x_n)\beta$ in a depth $\leq k$ w.r.t. $d(t_1, ..., t_n)$.

Step 1 Let $\mathcal{A}$ be such that $\mathcal{A}(x_j) = \bar{\mathcal{A}}(\varphi_j)$. To show that $\mathcal{A}(\psi) = \bar{\mathcal{A}}(\bar{\psi})$, let $\psi_1, ..., \psi_m$ be all the conjuncts in the DNF of ψ. Then $\bar{\psi} = \bar{\psi}_1 \vee \cdots \vee \bar{\psi}_m$ where $\bar{\psi}_i = \psi_i[x_j \leftarrow \varphi_j]$. Note that $\bar{\mathcal{A}}(\bar{\psi})$ is true iff $\exists \bar{\psi}_i$ that is true. But then $\mathcal{A}(\psi_i) = true$ iff $\bar{\mathcal{A}}(\bar{\psi}_i) = true$ (by definition of $\mathcal{A}$).

Step 2 Let $\sigma \geq_c \bar{\mathcal{A}}$. Since φ_j is a k-maximal property for t_j, there exists an E-derivation (of depth $\leq k$ w.r.t. t_j) that normalizes $t_j\sigma$ whenever $\bar{\mathcal{A}}(\varphi_j) = true$. Let $NF(t_j\sigma)$ denote the normal form reached by such a derivation. Now let $\beta(x_j) = NF(t_j\sigma)$ whenever $\bar{\mathcal{A}}(\varphi_j) = true$ and $\beta(x_j) = t_j\sigma$ otherwise. By concatenating E-derivations to normalize each $t_j\sigma$ such that $\bar{\mathcal{A}}(\varphi_j) = true$, we obtain an E-derivation Δ that reduces $d(t_1, ..., t_n)\sigma$ to $d(x_1, ..., x_n)\beta$. Observe, by definition of depth, that this derivation is of depth $\leq k$ w.r.t. $d(t_1, ..., t_n)$. Furthermore, note that $\beta(x_j)$ is in NF whenever $\bar{\mathcal{A}}(\varphi_j) = \mathcal{A}(x_j) = true$, and so $\beta \geq_c \mathcal{A}$.
Step 3 Let β' be such that $\beta'(x_j) = \beta(x_j)$ whenever $\mathcal{A}(x_j) = true$ and $\beta'(x_j) = \perp$[1] otherwise. Notice $\beta' \geq_c \mathcal{A}$ and $\mathcal{A}(\psi) = true$. Therefore there exists an E-derivation Γ' that normalizes $d(x_1, ..., x_n)\beta'$ within a depth of k with respect to $d(x_1, \ldots, x_n)$. Observe that in Γ' all subterms that do not possess a HNF must be discarded. Therefore all reductions performed within $\beta'(x_j)$ for x_j such that $\mathcal{A}(x_j) = false$ are irrelevant and can be removed from Γ' to obtain a new derivation Γ. Now, observe that whenever $\mathcal{A}(x_j) = false$, $\beta'(x_j)$ is discarded by Γ without ever reducing any redex inside it and that $\beta'(x_j) = \beta(x_j)$ for other j. Therefore Γ can be used to normalize $d(x_1, ..., x_n)\beta$ in a depth $\leq k$ w.r.t. $d(x_1, ..., x_n)$.

Now consider the E-derivation $\bar{\Gamma}$ obtained by concatenating Δ (from Step 2) and Γ. Note that no function symbol in any t_j such that $\mathcal{A}(x_j) = true$ appears in $d(x_1, ..., x_n)\beta$ and so the maximum label assigned to any of these function symbols in $\bar{\Gamma}$ is $\leq k$ w.r.t $d(t_1, t_2, \ldots, t_n)$. Also observe that the labels of function symbols in t_j for those j such that $\mathcal{A}(x_j) = false$, remain zero since no redex with these function symbols is ever reduced in Δ or Γ. Furthermore, since the label of d in $d(x_1, ..., x_n)\sigma$ is zero and since depth of Γ w.r.t. $d(x_1, ..., x_n)$ is $\leq k$, its label never increases beyond k. Therefore the depth of $\bar{\Gamma}$ is $\leq k$ w.r.t. $d(t_1, ..., t_n)$. $\square$

Now we show that equation 3 (in section 2.1) computes $(k+1)$-maximal property of f, given the k-maximal properties of the e_j's.

LEMMA 5. *Let φ_j's be respective k-maximal properties of e_j's. Then $\varphi = \bigvee_{j=1}^{k} g_j(\varphi_j)$ is a $(k+1)$-maximal property of $f(x_1, ..., x_n)$.*

PROOF. This theorem is proved in three steps by showing that

$$\forall\mathcal{A}[\mathcal{A}(\varphi) \Rightarrow \forall\sigma \geq_c \mathcal{A} \;\; t\sigma\downarrow_{k+1, f(x_1,\ldots,x_n)} E\text{-NF}]$$

1. Given an $\mathcal{A}$ such that $\mathcal{A}(\varphi) = true$, we select one rule $f(t_1, ..., t_n) \rightarrow e_j$ and construct an $s =_E f(x_1, ..., x_n)$ such that $\forall\sigma \geq_c \mathcal{A}$ an instance $s\sigma$ can be reduced into an instance $e_j\beta$.

2. We show that $\beta \geq_c \mathcal{A}'$ for some $\mathcal{A}'$ such that $\mathcal{A}'(\varphi_j) = true$.

3. Finally, we construct an E-derivation to normalize $f(x_1, ..., x_n)$ in depth $\leq k+1$ (w.r.t. $f(x_1, ..., x_n)$) based on the the derivation sequence to reduce $f(x_1, ..., x_n)\sigma$ into $e_j\beta$ and the fact that φ_j is a k-maximal property of e_j.

Step 1 Given any $\mathcal{A}$ such that $\mathcal{A}(\varphi) = true$, there is a j such that $\mathcal{A}(g_j(\varphi_j)) = true$. Let ψ be a conjunct in the DNF of $g_j(\varphi_j)$ that is made true by $\mathcal{A}$. By equation (4) (in section 2.1) defining the g_j's, there is a conjunct ψ' in the DNF of φ_j such that

[1] $\perp$ stands for any term lacking head normal form.

$\psi = g_j(\psi')$. Now we let $s = f(s_1, ..., s_n)$, where each s_i is further specified by one of the following two cases.

Case 1. x_i is in ψ: $s_i = x_i$

Case 2. x_i is not in ψ: Let $x_{i1}, ..., x_{ir_i}$ be the variables in t_i. Then, by equations 6 (in section 2.1), $\exists x_{il}$ $(1 \leq l \leq r_i)$ that is not in ψ' or otherwise $x_i \in \psi$. Let $s_i = t_i[x_{il} \leftarrow x_i, x_{im} = c\ , \forall m \neq l]$ where c is a constructor. It can be easily seen that $s_i =_E x_i$ and so $s =_E f(x_1, x_2, \ldots, x_n)$.

Observe that $\mathcal{A}(x_i)$ is true for x_i in ψ and since $\sigma \geq_c \mathcal{A}$, $\sigma(x_i)$ can be normalized into some term u. Since u consists of only constructors, it can be reduced into an instance $t_i[x_{im} \leftarrow c]$ (for every variable x_{im} in t_i and for some constructor c) using E-reductions. Thus there is an E-derivation that reduces $s_i\sigma$ into $t_i[x_{im} \leftarrow c]$ for $x_i \in \psi$. Also note that for $x_i \notin \psi$, s_i itself is an instance of t_i and therefore $s_i\sigma$ is an instance of t_i. By concatenating such E-derivations for each x_i in ψ we obtain an E-derivation Δ that reduces $s\sigma$ into an instance $f(t_1, ..., t_n)\beta$. This can now be reduced using the rule $f(t_1, ..., t_n) \rightarrow e_j$ into $e_j\beta$.

Step 2 Let $\mathcal{A}'$ be defined by $\mathcal{A}'(x_{im}) = true$ iff $x_{im} \in \psi'$. Clearly $\mathcal{A}'(\psi') = true$ and since ψ' is a conjunct in φ_j, $\mathcal{A}'(\varphi_j) = true$. We consider the cases 1 and 2 (of step 1) to show that $\beta \geq_c \mathcal{A}'$, or equivalently that $\beta(x_{im}) \downarrow E$-NF whenever x_{im} appears in ψ'. In case 1, every x_{im} in t_i is in ψ' and the substitution for every one of them is c and so is in E-NF. In the second case also, the substitution for every variable x_{im} that appears in ψ' is c. Therefore $\beta(x_{im}) \downarrow E$-NF in both cases.

Step 3 Since φ_j is a k-maximal property of e_j and $\mathcal{A}'(\varphi_j) = true$, and since $\beta \geq_c \mathcal{A}'$, there is an E-derivation Γ' of depth $\leq k$ w.r.t. e_j that normalizes $e_j\beta$. Consider the E-derivation obtained by concatenating Δ, the reduction $f(t_1, ..., t_n)\beta \rightarrow e_j\beta$ and Γ'. This derivation normalizes $f(x_1, ..., x_n)\sigma$ and is of depth $\leq k + 1$ w.r.t. $f(x_1, ..., x_n)$ because in the step $f(t_1, t_2, \ldots, t_n)\beta \rightarrow e_j\beta$ the label of all function symbols in e_j is one and these labels increase by at most k in Γ'. □

LEMMA 6. *$t^{ee,k}$ is a k-maximal property of t.*

PROOF. By induction on k. For the basis, note that $f^{ee,0} = false$ and so 12 holds vacuously. Using lemma 4 and the definition of $t^{ee,k}$ in terms of $f^{ee,k}$, we can establish by simple induction on the structure of t that $t^{ee,0}$ is a 0-complete property of t.

For the induction step, assume that $e_j^{ee,k}$'s are the respective k-complete properties of e_j's. By lemma 5, we see that $f^{ee,k+1}$ is a $k+1$-complete property of f. As in the basis step, using the definition of $t^{ee,k+1}$ (in terms of $f^{ee,k+1}$) and lemma 4, we can show that $t^{ee,k+1}$ is a $k+1$-complete property of t. □

Now, by soundness of ee-analysis,

COROLLARY 7. *t^{ee} is EMSP of t.*

4. CHARACTERIZATION THEOREM FOR DD-ANALYSIS

To illustrate the generality of our technique we briefly outline its use in proving a similar characterization theorem for dd-analysis. As mentioned earlier, dd-analysis is an enhancement of Mycroft's method to deal with head normal forms in non-flat domains. In particular, a dd-strictness property of a function f specifies which arguments of the function are to be head normalized in order to obtain the head normal form of an application of f. Among other applications, it has proven valuable in repairing violations of strong sequentiality (a concept pioneered by Huet and Levy [1979]) in functional programs. The equations to compute t^{dd} are as follows (See [Sekar et al. 1990] for details).

$$x^{dd} = x \tag{13}$$

$$[c(x_1, ..., x_n)]^{dd} = 1 \tag{14}$$

$$[f(x_1, ..., x_n)]^{dd} = \bigvee_{j=1}^{k} g_j(e_j^{dd}) \tag{15}$$

The g_j's are defined as follows.

$$g_j(\varphi) = \left(\bigwedge_{\substack{i \mid t_i^j \text{ is a} \\ \text{nonvariable}}} x_i \right) \wedge g_j(\varphi) \tag{16}$$

$$g_j(\varphi_1 \vee \varphi_2) = g_j(\varphi_1) \vee g_j(\varphi_2) \tag{17}$$

$$g_j(\varphi_1 \wedge \varphi_2) = g_j(\varphi_1) \wedge g_j(\varphi_2) \tag{18}$$

$$g_j(x_{i1}) = x_i, \text{ if } t_i^j \text{ is a variable} \tag{19}$$

$$g_j(x_{il}) = 1, \text{ if } t_i^j \text{ is a nonvariable} \tag{20}$$

To obtain a characterization theorem, the first step is to specify E-axioms appropriate for dd-analysis.

$$c(t_1, \ldots, t_n) = c', \forall c, c' \in \mathcal{C} \tag{21}$$

The effect of this equation is to equate any two terms that are in head normal form.

In the remainder of this section, we specify the necessary changes to definitions and theorems (used in proving the characterization theorem for ee-analysis) so as to obtain a similar result for dd-analysis. In all definitions, theorems and lemmas we replace NF by HNF, E-NF by E-HNF and "term without normal form" by "term lacking head normal form".

We can prove theorem 1 with respect to the E-axioms for dd-analysis using essentially the same proof used for ee-analysis. In lemma 3, for part (a) note that dd of right-hand side is precisely the dd of the left-hand side in equation (21). For part (b) we only need to argue that g_j functions of equivalent programs remain unchanged. To do so, we only need consider equations (16), (19) and (20). Observe that these rules depend only on whether t_j^i is a variable or non-variable. But this property is left unchanged by the E-axiom.

Lemma 4 carries over without any changes in the arguments. For lemma 5, the following observations are necessary. In case 2 of step 1, we observe that $x \notin \psi$ implies that $x_{i1} \notin \psi'$ and $t_i = x_{i1}$. Therefore, in both cases s_i is identical to x_i. Also observe that whenever $A(x_i)$ is true for $x_i \in \psi$ and $\sigma \geq_c A$, then $\sigma(x_i)$ can be head normalized to some term u. As u is in HNF it can be E-reduced to $t_i[x_{i1} \leftarrow c]$. The remainder of the reasoning carries over unchanged. With these changes, we have the following result for dd-analysis as well.

THEOREM 8. *$t^{dd}(P) \Rightarrow \varphi$ iff φ is a shared (dd) strictness property for t.*

5. SIGNIFICANCE AND IMPLICATIONS

Strictness analysis methods such as that of Mycroft are often defined by a non-standard semantics obtained by re-interpreting programs over a simpler abstract domain. This definition is useful for computing strictness properties but not helpful in answering questions dealing with the power of the method in a manner *independent* of the computational procedure used. Such questions take the form: given a class of programs $\mathcal{P}$ is the method complete for this class? What is the information lost by the method? How can its power be enhanced? etc. In contrast, we give a proof technique to answer these questions in a computation independent manner. Specifically, E-axioms are a syntactic characterization of the information lost by the method and E-rewriting provides a powerful algorithm-independent tool for answering questions such as above. By this we mean that if we want to know whether an analysis procedure will compute a strictness property, we need not know its computation procedure at all. We need only check if the strictness property is preserved if the program is modified as permitted by the E-axioms (e.g., changing constants in Mycrofts and *ee*-analysis method). Thus, using our approach we can draw the following conclusions about Mycroft's and our method.

—The information loss in *ee*-analysis method is *only* due to the mapping of all constants to a single value; *all* other abstraction steps are lossless.

—Any method that uniformly ignores constants can be *no more* powerful than *ee*-analysis method. More precisely, if such a method computes the same strictness properties for the lhs and rhs of the E-axioms then it will not have any additional power. Therefore the only source of improvement is to enrich the analysis domain by adding more "points" to it.

—We show that Mycroft's and our methods are complete for *non-erasing* programs, wherein all the variables in the lhs of a rule appears on the rhs. There are many important functions that are non-erasing such as *append*, *reverse*, *matrix multiplication*, etc. Observe that if P is non-erasing, so is any $Q =_E P$. By our main theorem, *ee*-analysis method deduces all shared strictness properties and so it is complete for the class of non-erasing programs. *Note that this proof makes no reference to the computational procedure used in deducing strictness properties.*

—We can enhance the above class by permitting the use of *if-then-else* (an erasing function) in the following way: if-then-else must not be used to "potentially throw away" function arguments. For example, *if x then y else z* throws away one of y or z, whereas *if x then y*z else y/z* does not throw away any arguments. *ee*-analysis method is also complete for this class of programs as well.

—On the other hand, ee-analysis method is incomplete even for the simple class of non-recursive programs with linear rhs. Functions F_1 and F_2 (Section 1.2) illustrate the incompleteness.

Finally, the technique developed in this paper provides a *general principle* for establishing similar characterization theorems for other strictness analysis methods. This is further discussed below.

5.1 Generality of Approach

In order to generalize our approach to deal with other methods for strictness analysis as well as extensions of our ee-analysis mentioned before, we first need to discover equational axioms to capture the loss of information in the analysis method. Discovering E-axioms that not only capture loss of information in an intuitive way but are also useful in proving completeness results, may require considerable ingenuity. For example, we leave as an open problem the discovery of such E-axioms for a method proposed by Wadler [1987] that reasons about lists using a four point domain.

The E-axioms induce an equivalence relation over classes of programs. It then remains to show that inferred strictness properties are related to equivalence classes of programs. The first step in this direction is to give a formal statement of the characterization theorem. In this direction, it is useful to develop the notion of E-rewriting and establish a result similar to theorem 1 which relates E-rewriting and performing reductions w.r.t. several programs. Such a result will make the statement of the characterization theorem and its proof cleaner and a lot less cumbersome. However, establishing such a theorem may be nontrivial. It appears that the only way the presence of E-axioms changes reduction sequences is in that E-axioms can make a non-redex into a redex or convert a redex for one lhs into a redex for another. Both these can be accomplished by allowing application of E-axioms only "around" redexes, suggesting that a result similar to theorem 1 will hold. If this theorem fails to hold, a characterization that explicitly refers to all programs E-equivalent to the given program can be given.

The next step is to show that the strictness properties computed are all shared strictness properties. This can be established using the analysis in a fairly straight forward manner if the E-axioms are correctly set up. The more difficult part of establishing the characterization theorem is to show that every shared strictness property (or equivalently, the maximal shared strictness property) is computed. The proof of this involves showing that each step in the analysis procedure (such as composition, function definition by pattern match, etc.) preserves maximal shared strictness properties. This part of the proof is simplified if we avoid pattern matching, but instead restrict ourselves to the language of recursion equations. In this case, we need only show that we have the maximal shared strictness properties of the base functions (such as if and arithmetic operators) and that the composition step loses no information other than that given by the E-axioms. Thus the only proof that needs to be extended is that of lemma 4.

6. RELATED WORK

Since the preliminary version of this paper appeared in POPL '91, much research [?; ?] has taken place in the area of completeness of abstract-interpretation-based analyses. ?] use a denotational approach to prove results similar to ours for the more general problem of abstract interpretation based analyses. Their proofs are based on a notion of *similarity semantics* that permits the semantics of values to be "confused." The confusion permitted is exactly that captured by the abstract domain. They show that abstract interpretation over linearly ordered domains for a first-order and a typed higher-order language is complete upto this notion of confusion.

Reddy and Kamin's proofs are based on denotational semantics. In particular, their completeness result is stated in terms of a similarity semantics that is derived directly from the structure of the abstract domain. This approach enables them establish a result with much wider applicability than ours. However, our approach has its own advantages that we mention below. First, the E-axioms provide a characterization that is not closely tied to the abstract domain itself. Moreover, the E-axioms provide a syntactic (or "localized") characterization of the information loss, whereas Reddy and Kamin's approach is based on replacing entire values (or expressions) with others that have the same similarity semantics. We believe that a syntactic characterization is more useful for drawing implications about the completeness of a method on classes of programs. For instance, it readily follows from our result that Mycroft's and ee-analysis are complete for non-erasing programs. In contrast, it is unclear how such simple arguments can be used to draw this conclusion based on the similarity semantics approach.

A second important difference between the results is as follows. Reddy and Kamin's result is based on the semantics-based formulation of abstract interpretation. Often, methods based on abstract intepretation do not make direct use of the semantics based formulation since it can lead to inefficiencies in practice. More efficient computational algorithms are hence developed for implementation purposes. (See, for instance, [Clack and Jones 1985; ?].) One must then address the issue of completeness of such algorithms. An operational approach such as ours can more easily deal with completeness issues at such algorithmic and implementation levels as well, in addition to addressing them at the formulation level. Indeed, our ee-analysis is specified in [Sekar et al. 1990] to provide the computational methods as well, and our completeness results thus apply to the actual implemtation of ee-analysis. It is this additional burden that makes our proofs quite detailed.

As a final remark on Reddy and Kamin's work, they note that their method does not extend to non-linearly-ordered domains. They remark that the basic problem is the possible loss of information in computing least upper bounds. It is interesting to note that our generalization of ee-analysis to arbitrary demands (as reported in [?]), produces results quite similar to abstract interpretation, but does not suffer from the above drawback when dealing with domains with lossy lub's. It is an interesting open problem whether demand analysis possesses completeness properties for all finite domains.

The results of ours and that of Reddy and Kamin enable us to characterize the power of an abstract intepretation in terms of information obtained by a concrete interpretation. While such a characterization can typically be useful to compare the power of different abstract interpretations, they may not answer the question of the relative power of two abstract interpretations with respect to the ability to deduce specific properties. For instance, Reddy and Kamin's result seems to suggest that the power of an abstract interpretation is determined simply by the structure of abstract domain. However, ?] gave an example where two abstract interpretations used different domains, but were able to identify the same strictness properties. To address this problem of more easily comparing abstract interpretations, Mycroft has proposed the concept of predicate-based abstract interpretation.

In this approach, in addition to the two components of interpretation (namely, a domain and the interpretation of functions), Mycroft adds a third component in the form of predicates that capture the program properties of interest. Another way to view the predicates is that they define a notion of observability. Mycroft then proceeds to set up a framework in which one can compare the power of two abstract interpretations with respect to this notion of observability.

ACKNOWLEDGMENTS

The authors would like to thank all the referees for their comments and suggestions, which have resulted in significantly improving the content as well as the presentation of this article.

REFERENCES

Augustsson, L. 1984. A compiler for lazy ML. In *ACM Conference on LISP and Functional Programming*. ACM, New York, 218–227.

Burn, G., Hankin, C., and Abramsky, S. 1985. Theory and practice of strictness analysis for higher order functions. In *Workshop on Programs as Data Objects*, N. Jones and H. Ganzinger Eds. Lecture Notes in Computer Science, vol 217. Springer–Verlag, New York.

Clack, C. and Jones, S. P. 1985. Strictness analysis — a practical approach. In *Functional Programming and Computer Architecture*. Lecture Notes in Computer Science, vol 20. Springer–Verlag, New York, 35–49.

Cousot, P. and Cousot, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In the *4th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 238–252.

Hall, C. and Wise, D. 1987. Compiling strictness into streams. In the *14th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 132–143.

Hudak, P. and Young, J. 1986. Higher order strictness analysis for untyped lambda calculus. In the *12th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 97–109.

Huet, G. and Levy, J. 1979. Computations in nonambiguous linear term rewriting systems. Tech. Rep. 359, IRIA, Le Chesney, France.

Hughes, R. and Wadler, P. 1987. Projections for strictness analysis. In *Functional Programming and Computer Architecture*. Lecture Notes in Computer Science, vol 27. Springer–Verlag, New York, 385–407.

Kaser, O., Pawagi, S., Ramakrishnan, C., Ramakrishnan, I., and Sekar, R. 1992. Fast parallel implementation of functional languages — the EQUALS experience. In *ACM Conference on Lisp and Functional Programming*. ACM, New York, 335–344.

Kuo, T. and Mishra, P. 1989. Strictness analysis: A type inference perspective. In *Conference on Functional Programming and Computer Architecture*. ACM, New York.

Mycroft, A. 1980. The theory and practice of transforming call-by-need into call-by-value. In the *International Symposium on Programming*. Lecture Notes in Computer Science, vol 8. Springer–Verlag, New York, 269–281.

Ramakrishnan, C., Ramakrishnan, I., and Sekar, R. 1995. A symbolic constraint-solving framework for analysis of logic programs. In *Partial Evaluation and Semantics-Based Program Manipulation*. ACM Press, New York, 12–23.

Sekar, R. and Ramakrishnan, I. 1993. Beyond strong sequentiality. *Inf. Comput. 104*, 1 (May), 78–109.

Sekar, R., Mishra, P., and Ramakrishnan, I. 1991. On the power and limitation of strictness analysis based on abstract interpretation. In the *18th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 37–48.

SEKAR, R., PAWAGI, S., AND RAMAKRISHNAN, I. 1990. Small domains spell fast strictness analysis. In the *17th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 169–183.

WADLER, P. 1987. Strictness analysis on non-flat domains (by abstract interpretation over finite domains). In *Abstract Interpretation of Declarative Languages*, S. Abramsky and C. Hankin Eds. Ellis Horwood, Chichester, U.K., 266–275.