A Precise Relationship Between the Deductive Power of Forward and Backward Strictness Analysis

Marc Neuberger
Prateek Mishra

Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400
{mjn, mishra}@sbcs.sunysb.edu

1 Introduction

The past decade has seen extensive research into the topic of strictness analysis and a rich family of analysis algorithms have been described in the literature. Most of the algorithms described in the literature fall into one of two classes: *forward analysis* [Mycroft80], based on abstract interpretation, and *backward analysis* [WH87], based on projections. Forward and backward analysis are also useful outside the context of strictness analysis, and have been applied to a wide range of problems such as binding-time analysis, sharing analysis and escape analysis [BjeHol89, JonLeM89, GomSes91, HunSan91, ParGol91].

This work is concerned with the relationship between the *deductive* power of forward and backward strictness analysis. We provide a precise and formal characterization of the relative power of these two analysis methods, when used for the strictness analysis of first-order functional programs over flat domains. Our main theorem is as follows: forward strictness analysis will determine that program P has property D *iff* backward strictness analysis determines that program P has equivalent property D'. To our knowledge, this result is the first of its kind.

Our results provide a foundation for a *comparative* study of the forward and backward analysis. Thus, by showing that the two analysis methods have equal deductive power, we provide a basis for the study of the *relative efficiency* of the two methods. Furthermore, our results allow characterizations of the deductive power of one analysis method to be applied to the other method. For example, in previous work [SMR91], we have given a precise and formal *semantic* characterization of the deductive power of forward strictness analysis. This characterization takes the form of showing that forward strictness analysis derives precisely those strictness properties for program P which do not depend upon any constant occurring in an evaluation of P. From our current results, we can conclude that this pleasant and important property holds for backwards strictness analysis as well.

1.1 Overview of Results

In Section 4, we describe an inference system that formalizes backwards analysis and provides a general description of projection-based reasoning about programs. Any specific backwards projection analysis algorithm may be seen an instance of this inference system by making an appropriate choice of specific projections. As our interest lies with strictness analysis, we choose our language of projections to be **ID**, **STR** and **FAIL** as these projections can describe all simple strictness properties of functions.

A major innovation in our inference system is the inclusion of a *disjunction* operator over projection properties. This allows us to better express strictness properties of the **if-then-else** construct by supporting statements that express *dependence* between variables. In backward analysis, we begin with a statement about the program or expression as a whole (for example the "context" in which it occurs), and derive statements about its free variables. Previous formalizations of projection analysis [WH87, Hughes85, Kamin90, HL91] have been restricted to deriving statements about the variables that are *independent* of each other. Thus, in these frameworks, expression **if** x **then** y **else** z is strict in x, but as it is neither strict in y, nor in z, we can reach no conclusions about them. In contrast,

1992 ACM LISP & F.P.-6/92/CA
© 1992 ACM 0-89791-483-X/92/0006/0127...$1.50

in our framework we can use the disjunction operator to conclude that the expression is *either* strict in y *or* strict in z. Observe that as forward analysis can express disjunctive properties, the frameworks described in [WH87, Hughes85, Kamin90, HL91] do not have the same deductive power as the corresponding forward analysis methods.

For our main theorem, we show that our inference system, restricted to the projections **ID**, **STR** and **FAIL**, has exactly the same power as forward strictness analysis. Below we briefly outline the proof which is fully described in Appendix B.

The key step in the proof is the discovery of a mapping ϕ that connects strictness properties in forward and backward analysis by mapping projection-based strictness properties to boolean functions. We show that ϕ respects the semantics of the two systems, that is, that the set of functions which have projection-based strictness property D is equal to the set of functions whose abstractions approximate $\phi(D)$, i.e. that D "means the same thing" as $\phi(D)$. For the final step, we show that our inference system derives a property D for expression e with respect to variables $\vec{x}$, *if and only if* forward analysis derives boolean function $\phi(D)$ for $\lambda\vec{x}.e$:

$$\vdash e[\vec{x}] \in D \iff \mathcal{E}^{\#}[\![e]\!]\varphi = \phi(D)$$

where $\mathcal{E}^{\#}$ is the standard abstraction function [Mycroft80] in forward analysis. Thus we have equivalence of the two systems.

1.2 Related Work

John Hughes [Hughes85] first proposed the use of backward analysis for reasoning about function strictness on structures. Subsequently, Wadler and Hughes [WH87] used projections to formalize backward analysis and described an algorithm for computing a variety of strictness properties using projections. Wadler and Hughes note that their treatment of conditional expressions has the limitations we have described above, but their work does not address whether these limitations are an *inherent* aspect of the use of projections. Our results demonstrate that this problem can be addressed by incorporating a disjunction operator into the analysis system. Wadler and Hughes do not offer any comparison of the deductive power of their system with forward strictness analysis.

Burn [Burn90] compares the expressiveness of projections versus abstraction in terms of sets of functions definable using the two techniques. His main result is a characterization of a class of projection properties that correspond to abstractions. Kamin [Kamin90] has further characterized projections that cannot be expressed as finite abstractions. These works do not discuss the relative *deductive* power of the two techniques.

Hughes [Hughes90] describes a family of analysis problems as instances of backward analysis. He claims that backward analysis is more efficient than forward analysis. However, it appears that the greater efficiency of backward analysis is due, at least in part, to the relative weakness of its deductive power. Future work could study the relative efficiency of forward and backward analysis using systems of equal deductive power.

The work of Hughes and Launchbury [HL91] is similar in spirit to our work. They describe an inference system for reasoning with projections and use it as a means for comparing backward with forward analysis. Their main result is the observation that, for certain problems, backward analysis might involve less search that forward analysis. Their work does not explicitly address the issue of the relationship between the deductive power of the two techniques.

1.3 Conclusion

Forward and backward analysis are important tools that have been widely used to design a variety of analysis algorithms. There is, as yet, little comparative study of the strengths and weaknesses of the two techniques. As a consequence compiler designers have chosen one or the other technique based on their intuition or by examination of a few examples. By demonstrating that forward and backward strictness analysis have equal deductive power, our work provides a first step towards the systematic comparative study of the two analysis techniques. Such a study would provide unambiguous guidance for compiler designers, based on principles and fact in place of intuitions and examples.

2 A Simple First-Order Functional Programming Language

Figure 1 shows the abstract syntax for the language we are studying. Its semantics are described in figure 2. We assume that we are given some domain of values *Val*. We are not concerned with the details of *Val*. In order for the **if-then-else** structure to be useful, however, *Val* should contain values for **true** and **false** .

The meaning of a program will be a mapping for defined function symbols (*FNam*) to functions over *Val*.

Syntactic Domains

x, x_i	$\in$	*Var*	(variables)	e, e_i	$\in$	*Exp*	(expressions)
c, c_i	$\in$	*Const*	(value constants)	d, d_i	$\in$	*Decl*	(declarations)
k, k_i	$\in$	*Prim*	(primitive functions)	p, p_i	$\in$	*Prog*	(programs)
f, f_i	$\in$	*FNam*	(function names)				

Syntactic Equations

Exp	::=	x	(variables)
	|	c	(constants)
	|	$f(e_i, ..., e_n)$	(application)
	|	$k(e_i, ..., e_n)$	(primitive application)
	|	**if** e_1 **then** e_2 **else** e_3	(if-then-else)
Decl	::=	$f(\vec{x}) = e$	(function declaration)
Prog	::=	$d_1, ..., d_n$	(program)

Figure 1: Abstract Syntax

3 Strictness Analysis Using Projections

Recall that a *projection* $\alpha : A \to A$ is a idempotent function which is less defined than the identity function.

DEFINITION 1 Let α_i be projections, $\alpha_i \in A_i \to A_i$, $\forall 1 \leq i \leq n$, and β be a projection, $\beta \in B \to B$. An n-ary function $f : A_1 * ... * A_n \to B$ has the *basic strictness property* $\beta \Longrightarrow [\alpha_1, ..., \alpha_n]$ *at point* $(a_1, ..., a_n) \in A_1 * ... * A_n$ if

$$\beta(f(a_1, ..., a_n)) = \beta(f(\alpha_1(a_1), ..., \alpha_n(a_n))).$$

If f has the property at all $(a_1, ..., a_n) \in A_1 * ... * A_n$, then we say that f has the property $\beta \Longrightarrow [\alpha_1, ..., \alpha_n]$. If f has the property $\beta \Longrightarrow [\alpha_1, ..., \alpha_n]$, we write $f \in \beta \Longrightarrow [\alpha_1, ..., \alpha_n]$. We also use $\beta \Longrightarrow [\alpha_1, ..., \alpha_n]$ to denote the set of f which have the property. We use ν, ν_i to denote arbitrary basic strictness properties.

DEFINITION 2 Let $C = \{\nu_1, ..., \nu_m\}$ be a set of strictness properties. We say that a function f has *basic property set* C at point $(a_1, ..., a_n)$, if for all i, f has ν_i at point $(a_1, ..., a_n)$. We will sometimes write C as $\nu_1 \wedge ... \wedge \nu_m$, or, more compactly, $\bigwedge_{i=1}^{c} \nu_i$.

DEFINITION 3 Let $D = \{C_1, ..., C_m\}$ be a set of basic property sets. We say that a function f has the *disjunctive strictness property* D if

$\forall (a_1, ..., a_n) \in A_1 * ... * A_n$, $\exists C_i \in D$, such that
f has basic property set C_i at point $(a_1, ..., a_n)$.

We will sometimes write $D = \{C_1, ..., C_m\}$ as $D = C_1 \vee ... \vee C_m$, or, more compactly, $\bigvee_{i=1}^{d} C_i$ or $\bigvee_{i=1}^{d} \bigwedge_{j=1}^{c_i} \nu_{ij}$, to emphasize the disjunctive nature of the definition. We use DSP_S to denote the set of basic strictness properties over the projection set S.

In order to characterize simple strictness ($f\perp = \perp$) with projections, it is necessary to lift the domains and functions with an additional element $\lightning$, pronounced "abort", where $\lightning \sqsubset \perp$. We will be working with lifted domains for values, but will still denote the lifted domain by *Val*. Functions will be lifted so that for all $f \in Func, f\lightning = \lightning$.

DEFINITION 4 We say that a projection α is *strict* if $\alpha(\perp) = \lightning$.

The idea of strict projections is that they represent a level of demand which includes simple strictness. The simplest strict projection is **STR** defined as

$$\begin{aligned} \mathbf{STR}(\lightning) &= \lightning \\ \mathbf{STR}(\perp) &= \lightning \\ \mathbf{STR}(x) &= x, \quad x \sqsupset \perp \end{aligned}$$

Observe that a function f is strict if and only if $f \in \mathbf{STR} \Longrightarrow \mathbf{STR}$. We use *NS* to denote the set of non-strict projections ($\{\alpha \mid \alpha(\perp) \neq \lightning\}$).

DEFINITION 5 A strictness property D is *satisfiable* if $f \in D$ for some f. Otherwise it is *unsatisfiable*.

Semantic Domains

$$
\begin{array}{rcll}
Val & & & \text{(Values)} \\
Func & = & Val^n \to Val & \text{(Functions)} \\
Env & = & \mathbb{N} \to Val & \text{(Variable Environment)} \\
FEnv & = & FNam \to Func & \text{(Function Environment)} \\
PEnv & = & Prim \to Func & \text{(Primitive Environment)}
\end{array}
$$

Semantic Functions

$$
\begin{array}{rcl}
\mathcal{C} & : & Const \to Val \\
\mathcal{K} & : & PEnv \\
\mathcal{E} & : & Exp \to FEnv \to Env \to Val
\end{array}
\qquad
\begin{array}{rcl}
\mathcal{D} & : & Prog \to FEnv \to FEnv \\
\mathcal{P} & : & Prog \to FEnv
\end{array}
$$

Semantic Equations

$$
\begin{array}{rcl}
\mathcal{E}[\![x_i]\!]\varphi\rho & = & \rho_i \\
\mathcal{E}[\![c]\!]\varphi\rho & = & \mathcal{C}[\![c]\!] \\
\mathcal{E}[\![k(e_1,...,e_n)]\!]\varphi\rho & = & \mathcal{K}[\![k]\!](\mathcal{E}[\![e_1]\!]\varphi\rho, ..., \mathcal{E}[\![e_n]\!]\varphi\rho) \\
\mathcal{E}[\![f(e_1,...,e_n)]\!]\varphi\rho & = & \varphi[\![f]\!](\mathcal{E}[\![e_1]\!]\varphi\rho, ..., \mathcal{E}[\![e_n]\!]\varphi\rho) \\
\mathcal{E}[\![\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3]\!]\varphi\rho & = & \begin{cases} \mathcal{E}[\![e_2]\!]\varphi\rho, & \text{if } \mathcal{E}[\![e_1]\!]\varphi\rho = \textbf{true} \\ \mathcal{E}[\![e_3]\!]\varphi\rho, & \text{if } \mathcal{E}[\![e_1]\!]\varphi\rho = \textbf{false} \end{cases}
\end{array}
$$

$$
\mathcal{D}\left[\!\!\left[\begin{array}{rcl} f_1(\vec{x_1}) & = & e_1 \\ & \vdots & \\ f_n(\vec{x_n}) & = & e_n \end{array} \right]\!\!\right] \varphi[\![f_i]\!] = \mathcal{E}[\![e_i]\!]\varphi
$$

$$
\mathcal{P}[\![p]\!] = \bigsqcup_i ((\mathcal{D}[\![p]\!])^i([f_i \mapsto \bot]))
$$

Figure 2: Language Semantics

We define the set $Prop_n$ to be the set of all satisfiable basic strictness properties for a function of arity n:

$$
Prop_n = \left\{ \beta \Longrightarrow [\alpha_1, ..., \alpha_n] \;\middle|\; \begin{array}{l} \beta \not\in NS, \text{ or} \\ \forall i,\ \alpha_i \in NS \end{array} \right\}
$$

We will need some way in which to combine two demands to yield a new demand. Suppose we have that $f \in \beta \Longrightarrow [\alpha_1, \alpha_2]$. Let $g = \lambda x.fxx$. We should be able to find an α for which $g \in \beta \Longrightarrow [\alpha]$. Following [WH87], we define an operator &, and say that $\alpha = \alpha_1 \& \alpha_2$. This operator is defined as follows:

$$
(\alpha_1 \& \alpha_2)d = \begin{cases} \lightning & \text{if } \alpha_1(d) = \lightning, \\ & \text{or } \alpha_2(d) = \lightning, \\ \alpha_1(d) \sqcup \alpha_2(d), & \text{otherwise.} \end{cases}
$$

Let's consider some simple examples using just the projections **ID** and **STR**. This allows us to talk about simple strictness of a function. The identity projection, **ID**, represents no information. Thus, any function has the properties $\textbf{ID} \Longrightarrow \textbf{ID}$ and $\textbf{STR} \Longrightarrow \textbf{ID}$.

Consider the definition

$$
f(x, y) = x + y
$$

Since f is strict in both x and y, we have $f \in \textbf{STR} \Longrightarrow [\textbf{STR}, \textbf{STR}]$. Now consider the definition

$$
f(x, y) = y
$$

Because f does not use x, so we cannot have $f \in \textbf{STR} \Longrightarrow [\textbf{STR}, \textbf{STR}]$. We have only $f \in \textbf{STR} \Longrightarrow [\textbf{ID}, \textbf{STR}]$. Suppose we have the following definition:

$$
\begin{array}{rcl}
f(x, y) & = & \textbf{if } x = 0 \textbf{ then} \\
& & y \\
& & \textbf{else} \\
& & y
\end{array}
$$

First of all, **if** is strict in the first argument and $=$ is strict in both its arguments, so f is strict in x. Further, since y is returned in both cases, f is strict in y, so $f \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}]$.

Let's try to abstract **if-then-else**:

$$\begin{aligned} g(x,y,z) &= \mathbf{if}\ x\ \mathbf{then} \\ &\quad\ y \\ &\quad\ \mathbf{else} \\ &\quad\ z \\ f(x,y) &= g(x=0,y,y) \end{aligned}$$

We would still like to be able to find $f \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}]$. Unfortunately we can't do this using basic strictness properties, for what property can we assign to g ? Certainly, g is strict in its first argument, so we have $g \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{ID}, \mathbf{ID}]$, but this is inadequate to find the strictness of f. We have neither $g \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}, \mathbf{ID}]$ nor $g \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{ID}, \mathbf{STR}]$. The first case fails to hold for $g(\mathbf{false}, \perp, x)$, the second, for $g(\mathbf{true}, x, \perp)$. Similarly, we can't have $g \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}, \mathbf{STR}]$, as both of the above examples serve as counterexamples to this assertion. Thus, the best we can do for g is $g \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{ID}, \mathbf{ID}]$, a fact inadequate to yield the desired information about f.

In order to remedy this problem, we use disjunctive strictness properties. Disjunction allows us to express the fact that g is strict in one of the two arms without specifying which one. Thus, we can say $g \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}, \mathbf{ID}] \vee \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{ID}, \mathbf{STR}]$. Now if we consider $g(x=0,y,y)$, we can see that in either case, the expression is strict in y, so we can get $f \in \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}]$.

4 An Analysis System

We now present an inference system for reasoning about strictness properties of programs. It is assumed that the user of the rules provides a set of projections which is closed under &, and complete sets of properties for the primitives. If the system is to be used for practical puposes, the set of projections needs to be finite. This insures that the property sets derived are finite.

The system derives disjunctive strictness properties. By their nature, individual disjunctive properties are capable of expressing all of the valid properties of a function. Thus, rather than having several proof trees for a given expression each of which yields one of the expression's properties, we have one proof tree which yields a single disjunctive strictness property which contains all of the information which the system can derive from the starting assumptions. This is a result of the fact the disjunctive strictness propreties are very much like boolean expressions, and two can be conjoined into a single property.

In describing the rules, we will use the notation

$$e[x_1, ..., x_n] \in D.$$

This means that if we regard e as a function from the variables $x_1, ..., x_n$, $(\vec{x})$, to a value, it has strictness property D. Roughly speaking, though abtraction is not part of our language

$$(\lambda \vec{x}.e) \in D.$$

In the rules, n will always represent the length of the $\vec{x}$ variable vector.

In order to simplify the presentation of the application rule we use an operator $\Psi_{m,n}$ defined in figure 3.

This operator handles the application of an m-ary function to m expressions, and computes a strictness property with respect to n free variables. The first argument it takes is a simple property of the function. The rest are simple properties of the actual parameters to the function. Note that the α_i occur in two places, and so must match. In order to insure that this operator is only used when such a match occurs, we define a predicate operator Θ, also shown in figure 3.

For an $A \in \mathit{FNam} \rightarrow \mathit{DSP}$, $B \in \mathit{Prim} \rightarrow \mathit{DSP}$, $e \in \mathit{Exp}$, $\vec{x}$ an n-vector of Var's, and $D \in \mathit{DSP}$, our system derives statements of the form:

$$A, B \vdash e[\vec{x}] \in D$$

where

$$D = \bigvee_i \bigwedge_j \beta_{ij} \Longrightarrow [\alpha_{ij1}, ..., \alpha_{ijn}]$$

meaning that with function strictness assumptions A, and primitive function strictness assumptions B, e has property D with respect to the variables $\vec{x}$.

We present the inference rules in figure 4.

The (const) rule states that for a constant, we have the conjunction of all rules which give non-strict contexts to the variables. This is clearly correct, since a constant has no free variables, so that any free variable x can be replaced by any $\alpha(x)$, provided that $\alpha(x)$ doesn't yield $\lightning$. If $\alpha(x)$ gave abort, then we would derive incorrect results, for example, for $f(x) = 1$, we could find $f \in \mathbf{STR} \Longrightarrow \mathbf{STR}$, which is clearly incorrect.

The (var) rule is just slightly different from the (const) rule, recognizing that a variable expression x has one free variable, namely x. Thus we restrict the context of x to be the context in which the expression appears. Thus, for example, for $f(x) = x$, we only derive properties of the form $\alpha \Longrightarrow \alpha$.

$$\Psi_{m,n} \quad : \quad Prop_m * {Prop_n}^m \rightarrow Prop_n$$
$$\Theta_{n,n} \quad : \quad Prop_m * {Prop_n}^m \rightarrow Bool$$

$$\Psi_{m,n}\begin{pmatrix} \beta \Longrightarrow [\alpha_1, ..., \alpha_m], \\ \alpha_1 \Longrightarrow [\delta_{11}, ..., \delta_{1n}], \\ \vdots \\ \alpha_m \Longrightarrow [\delta_{m1}, ..., \delta_{mn}] \end{pmatrix} = \beta \Longrightarrow [\&_i\{\delta_{i1}\}, ..., \&_i\{\delta_{in}\}]$$

$$\Theta_{m,n}\begin{pmatrix} \beta \Longrightarrow [\alpha_1, ..., \alpha_m], \\ \gamma_1 \Longrightarrow [\delta_{11}, ..., \delta_{1n}], \\ \vdots \\ \gamma_m \Longrightarrow [\delta_{m1}, ..., \delta_{mn}] \end{pmatrix} = \forall 1 \leq i \leq m,\ \alpha_i = \gamma_i$$

Figure 3: Definitions of Ψ and Θ

The (app) rule is where all of the interesting work happens. The basic idea is this: We have a disjunctive strictness property for the function being applied. We also have disjunctive strictness properties for each of the actual parameters relative to the variables of interest ($\vec{x}$). The result is a disjunction, the disjuncts of which are formed by choosing a disjunct from each of the function property and the actual parameter properties. For each result disjunct, all of the choices of basic strictness properties from the source disjuncts are considered. Each of these compatible sets gives a basic strictness property to the result through the use of the $\Psi_{m,n}$ operator. So far the manipulation is purely a logical one, and follows from the fact that the conjunction and disjunction in the properties correspond to the logical operations conjunction and disjunction.

The $\Psi_{m,n}$ operator is basically an application operator. Its operation is to collect the individual contexts in which the variables occur in each actual argument and, using the & operator, generate a suitable context for each variable for the whole application. The correctness of using & to join the contexts is argued in [WH87], and is not hard to see by reviewing the definitions.

The (rec) rule allows derivation of properties for a whole program. Note the assumption that $A(f)$ is satisfiable. This is necessary because assuming unsatisfiable properties for a function can yield a result claiming that some function actually has an unsatisfiable property, a contradiction.

5 Comparison of Inference System with Abstract Interpretation

In this section we will consider an instance of the inference system in which we consider only the projections **ID**, **STR**, and **FAIL**. These three projections allow us to talk about simple strictness as Mycroft's system does.

5.1 Abstract Interpretation

We start by presenting an abstract interpretation similar to Mycroft's. We define abstraction map(s), $\mathcal{A}_n$ in figure 5. This map tells us exactly when a function is strict in a given argument or set of arguments. In other words, for a given program p and n-ary function f defined in the program, $\mathcal{P}[\![p]\!][\![f]\!](a_1, ..., a_n) = \perp$ exactly when $\mathcal{A}_n(\mathcal{P}[\![p]\!][\![f]\!])(\mathcal{A}_0(a_1), ..., \mathcal{A}_0(a_n)) = 0$. Of course, since termination is, in general, undecidable, we cannot compute $\mathcal{A}_n(\mathcal{P}[\![p]\!][\![f]\!])$. We therefore define a computable approximation to this abstraction, $\mathcal{P}^\#$. The following relationship holds between $\mathcal{P}$, $\mathcal{P}^\#$, and $\mathcal{A}_n$:

$$\mathcal{A}_n(\mathcal{P}[\![p]\!][\![f]\!]) \sqsubseteq \mathcal{P}^\#[\![p]\!][\![f]\!]$$

for all p, f. Thus, $\mathcal{P}^\#$ may not always tell us when a function fails to terminate.

5.2 Relative Power of the Two Systems

Note that with only the three projections **ID**, **STR** and **FAIL**, the only basic properties which are of any interest are of the form $\mathbf{STR} \Longrightarrow [\alpha_1, ..., \alpha_n]$. The only satisfiable property of the form $\mathbf{ID} \Longrightarrow [\alpha_1, ..., \alpha_n]$,

$$\text{(const)}\quad \frac{}{A, B \vdash c[\vec{x}] \in \bigwedge \left\{ \begin{array}{ll} & \{\beta \Longrightarrow [\alpha_1, ..., \alpha_n] \mid \forall 1 \leq i \leq n,\ \alpha_i \in NS\} \\ \cup & \{\mathbf{ID} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}]\} \\ \cup & \{\mathbf{FAIL} \Longrightarrow [\mathbf{FAIL}, ..., \mathbf{FAIL}]\} \end{array} \right\}}$$

$$\text{(var)}\quad \frac{}{A, B \vdash x_i[\vec{x}] \in \bigwedge \left\{ \begin{array}{ll} & \{\beta \Longrightarrow [\alpha_1, ..., \alpha_{i-1}, \beta, \alpha_{i+1}, ..., \alpha_n] \mid \forall 1 \leq i \leq n,\ \alpha_i \in NS\} \\ \cup & \{\mathbf{ID} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}]\} \\ \cup & \{\mathbf{FAIL} \Longrightarrow [\mathbf{FAIL}, ..., \mathbf{FAIL}]\} \end{array} \right\}}$$

$$\text{(app)}\quad \frac{\begin{array}{c} A(f) = D = \bigvee_{i=1}^{d} \bigwedge_{j=1}^{c_i} \nu_{0ij}, f \in FNam \quad \text{or} \quad B(f) = D = \bigvee_{i=1}^{d} \bigwedge_{j=1}^{c_i} \nu_{0ij}, f \in Prim \\ A, B \vdash e_1[\vec{x}] \in D_1 = \bigvee_{i=1}^{d_1} \bigwedge_{j=1}^{c_{1i}} \nu_{1ij} \\ \vdots \\ A, B \vdash e_m[\vec{x}] \in D_m = \bigvee_{i=1}^{d_m} \bigwedge_{j=1}^{c_{mi}} \nu_{mij} \end{array}}{\begin{array}{l} A, B \vdash f(e_1, ..., e_m)[\vec{x}] \in \bigvee_{i=1}^{d} \bigvee_{k_1=1}^{d_1} \cdots \bigvee_{k_m=1}^{d_m} \bigwedge_{(j,j_1,...,j_m) \in S} \Psi_{m,n}(\nu_{0ij}, \nu_{1k_1j_1}, ..., \nu_{mk_mj_m}) \\ \quad \text{where } S = \{(j, j_1, ..., j_m) \mid \Theta(\nu_{0ij}, \nu_{1k_1j_1}, ..., \nu_{mk_mj_m})\} \end{array}}$$

$$\text{(rec)}\quad \frac{\begin{array}{c} \forall f, A(f) \text{ is satisfiable} \\ \forall 1 \leq i \leq m,\ A, B \vdash e_i[\vec{x_i}] \in A(f_i) \end{array}}{B \vdash \left\{ \begin{array}{lcl} f_1(\vec{x_1}) & = & e_1 \\ & \vdots & \\ f_m(\vec{x_m}) & = & e_m \end{array} \right\} : A}$$

Figure 4: Inference Rules

is $\mathbf{ID} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}]$, which is held by all functions. For any function, we have $\mathbf{FAIL} \Longrightarrow [\mathbf{FAIL}, ..., \mathbf{FAIL}]$ which is the strongest property of the form $\mathbf{FAIL} \Longrightarrow ...$, so there are no interesting properties of that form. Note that any expressible property is equivalent to some property of the form:

$$\begin{array}{ll} & \left(\begin{array}{ll} & \mathbf{STR} \Longrightarrow [\alpha_{11}, ..., \alpha_{1n}] \\ \wedge & \mathbf{ID} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}] \\ \wedge & \mathbf{FAIL} \Longrightarrow [\mathbf{FAIL}, ..., \mathbf{FAIL}] \end{array} \right) \\ \vee & \vdots \\ \vee & \left(\begin{array}{ll} & \mathbf{STR} \Longrightarrow [\alpha_{d1}, ..., \alpha_{dn}] \\ \wedge & \mathbf{ID} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}] \\ \wedge & \mathbf{FAIL} \Longrightarrow [\mathbf{FAIL}, ..., \mathbf{FAIL}] \end{array} \right) \end{array}$$

So for $\bigvee_{i=1}^{d} \bigwedge_{j=1}^{3} \beta_{ij} \Longrightarrow [\alpha_{ij1}, ..., \alpha_{ijn}]$, we have, for all i, that $\beta_{i1} = \mathbf{STR}$, $\beta_{i2} = \mathbf{ID}$, $\beta_{i3} = \mathbf{FAIL}$, $\alpha_{i2k} = \mathbf{ID}$, and $\alpha_{i3k} = \mathbf{FAIL}$. We call properties written in this form *canonical* properties. It can be shown that if the assumptions (A, B) in the inference rules give properties in canonical form, then the rules will only yield properties in canonical form. Thus we can restrict our attention to canonical properties.

Rather than always saying

$$\begin{array}{ll} & \mathbf{STR} \Longrightarrow [\alpha_{11}, ..., \alpha_{1n}] \\ \wedge & \mathbf{ID} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}] \\ \wedge & \mathbf{FAIL} \Longrightarrow [\mathbf{FAIL}, ..., \mathbf{FAIL}] \end{array}$$

we will abbreviate this by $\mathbf{STR} \Longrightarrow [\alpha_{11}, ..., \alpha_{1n}]$, leaving the **ID** and **FAIL** cases understood.

Our aim is to prove that our inference system, using only the projections **ID**, **STR** and **FAIL**, has exactly the same power as the abstract interpretation we have

$$
\begin{array}{rcll}
Bool & = & \{0,1\}, & 0 \sqsubseteq 1
\end{array}
$$

$$
\begin{array}{rcll}
\mathcal{A}_0 & : & Val \rightarrow Bool & \\
\mathcal{A}_0(\bot) & = & 0 & \\
\mathcal{A}_0(v) & = & 1, & v \neq \bot
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{A}_n & : & (Val^n \rightarrow Val) \rightarrow (Bool^n \rightarrow Bool) \\
\mathcal{A}_n(f)(b_1,...,b_n) & = & \left\{ \begin{array}{ll} 0 & \text{if } \forall v_1 ... v_n,\ (\forall 1 \leq i \leq n,\ \mathcal{A}_0(v_i) \sqsubseteq b_i) \Rightarrow \mathcal{A}_0(f(v_1,...,v_n)) = 0 \\ 1 & \text{otherwise.} \end{array} \right.
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{E}^\#[\![c]\!]\varphi\rho & = & 1 \\
\mathcal{E}^\#[\![x_i]\!]\varphi\rho & = & \rho_i \\
\mathcal{E}^\#[\![k(e_1,...,e_n)]\!]\varphi\rho & = & (\mathcal{K}^\#[\![k]\!])(\mathcal{E}^\#[\![e_1]\!]\varphi\rho,...,\mathcal{E}^\#[\![e_n]\!]\varphi\rho) \\
\mathcal{E}^\#[\![f(e_1,...,e_n)]\!]\varphi\rho & = & (\varphi[\![f]\!])(\mathcal{E}^\#[\![e_1]\!]\varphi\rho,...,\mathcal{E}^\#[\![e_n]\!]\varphi\rho)
\end{array}
$$

$$
\mathcal{D}^\#\left[\!\!\left[\begin{array}{rcl} f_1(\vec{x_1}) & = & e_1 \\ & \vdots & \\ f_n(\vec{x_n}) & = & e_n \end{array} \right]\!\!\right] \varphi[\![f_i]\!] \quad = \quad \mathcal{E}^\#[\![e_i]\!]\varphi
$$

$$
\mathcal{P}^\#[\![p]\!] \quad = \quad \bigsqcup_i((\mathcal{D}^\#[\![p]\!])^i([f_i \mapsto \bot]))
$$

Figure 5: Abstract Interpretation

presented. We begin by demonstrating a mapping ϕ from disjunctive strictness properties to boolean functions. We then show that this mapping respects the semantics of the two systems, that is, that the set of functions which have disjunctive strictness property D is the same set of functions whose abstractions approximate $\phi(D)$, i.e. $\{f \mid f \in D\} = \{f \mid \mathcal{A}_n(f) \sqsubseteq \phi(D)\}$. This tells us that D "means the same thing" as $\phi(D)$. Once we have this, we will show that when our inference system derives a property D for a function in a program, the abstract interpretation ($\mathcal{E}^\#$) will derive $\phi(D)$. Thus we will have equivalence of the two systems.

We define an operator, $\overline{\alpha}$, which gives a boolean function for each of our projections.

$$
\begin{array}{rcl}
\overline{\mathbf{ID}}(x) & = & 1 \\
\overline{\mathbf{STR}}(x) & = & x \\
\overline{\mathbf{FAIL}}(x) & = & 0
\end{array}
$$

We define $\phi : DSP \rightarrow Bool^n \rightarrow Bool$ in figure 6.

Taking an interesting property of **if-then-else** for example,

$$
\begin{array}{l}
\phi\left(\begin{array}{ll} & \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{STR}, \mathbf{ID}] \\ \vee & \mathbf{STR} \Longrightarrow [\mathbf{STR}, \mathbf{ID}, \mathbf{STR}] \end{array} \right) \\
\quad = \quad \lambda v_1 v_2 v_3.\ \begin{array}{ll} & \overline{\mathbf{STR}}(v_1) \wedge \overline{\mathbf{STR}}(v_2) \wedge \overline{\mathbf{ID}}(v_3) \\ \vee & \overline{\mathbf{STR}}(v_1) \wedge \overline{\mathbf{ID}}(v_2) \wedge \overline{\mathbf{STR}}(v_3) \end{array} \\
\quad = \quad \lambda v_1 v_2 v_3.\ (v_1 \wedge v_2 \wedge 1) \vee (v_1 \wedge 1 \wedge v_3) \\
\quad = \quad \lambda v_1 v_2 v_3.\ (v_1 \wedge v_2) \vee (v_1 \wedge v_3).
\end{array}
$$

This being equivalent by distributivity to Mycroft's definition of $IF^\#$, $IF^\#(p,x,y) = p \wedge (x \vee y)$.

PROPOSITION 1 *Let* $D \in DSP$,

$$
D = \left(\begin{array}{ll} & \mathbf{STR} \Longrightarrow [\alpha_{11},...,\alpha_{1n}] \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{21},...,\alpha_{2n}] \\ \vee & \ldots \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{m1},...,\alpha_{mn}] \end{array} \right)
$$

Then $\{f \mid f \in D\} = \{f \mid \mathcal{A}_n(f) \sqsubseteq \phi(D)\}$.

$$\phi\begin{pmatrix} & \mathbf{STR} \Longrightarrow [\alpha_{11},...,\alpha_{1n}] \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{21},...,\alpha_{2n}] \\ \vee & ... \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{m1},...,\alpha_{mn}] \end{pmatrix} = \lambda\vec{b}.\begin{pmatrix} & \overline{\alpha_{11}}(b_1) \wedge ... \wedge \overline{\alpha_{1n}}(b_n) \\ \vee & \overline{\alpha_{21}}(b_1) \wedge ... \wedge \overline{\alpha_{2n}}(b_n) \\ \vee & ... \\ \vee & \overline{\alpha_{m1}}(b_1) \wedge ... \wedge \overline{\alpha_{mn}}(b_n) \end{pmatrix}$$

or, more compactly

$$\phi(\bigvee_{i=1}^{d}(\mathbf{STR} \Longrightarrow [\alpha_{i1},...,\alpha_{in}])) = \lambda\vec{b}.\bigvee_{i=1}^{d}\bigwedge_{j=1}^{n}\overline{\alpha_{ij}}(b_j)$$

Figure 6: Definition of ϕ

Now we prove the equivalence of expression analysis. After that, we will prove the equivalence of program analysis, which basically follows from the expression analysis result.

THEOREM 1 *Let $A : FNam \to DSP$, $B : Prim \to DSP$, and $\varphi : FNam \to Bool^n \to Bool$. Assume that for all f, $A(f)$ is in canonical form, and that for all k, $B(k)$ is in canonical form. Further, assume that $\phi(B(k)) = \mathcal{K}^\#[\![k]\!]$, and that $\phi(A(f)) = \varphi[\![f]\!]$. Then for any expression e, and any disjunctive strictness property*

$$D = \begin{pmatrix} & \mathbf{STR} \Longrightarrow [\alpha_{11},...,\alpha_{1n}] \\ \vee & ... \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{m1},...,\alpha_{mn}] \end{pmatrix}$$

our inference system derives

$$A, B \vdash e[\vec{x}] \in D$$

if and only if

$$\mathcal{E}^\#[\![e]\!]\varphi = \phi(D).$$

Now we need to prove that our inference system can generate the same results for programs that abstract interpretation does. This is a fairly simple result of theorem 1.

THEOREM 2 *Let p be some program. Assume that $\phi(B(k)) = \mathcal{K}^\#[\![k]\!]$ for all primitives k. Then if $\mathcal{P}^\#[\![p]\!] = \phi(A)$, inference system II can derive $B \vdash p : A$.*

PROOF: We have, from the definition of $\mathcal{P}^\#$, that $\mathcal{D}^\#[\![p]\!](\phi(A)) = \phi(A)$ meaning that for all i, $\mathcal{E}^\#[\![e_i]\!](\phi(A)) = \phi(A(f_i))$. But from theorem 1, we have then that $A, B \vdash e_i \in A(f_i)$. From the (rec) rule then, we immediately have $B \vdash p : A$.

□

References

[BjeHol89] Bror Bjerner and Sören Holmström, A Compositional Approach to Time Analysis of First Order Lazy Functional Programs. In *Functional Programming Languages and Computer Architecture*, pp. 157-165. London, September 1989.

[Burn90] G. Burn, A Relationship Between Abstract Interpretation and Projection Analysis. In *The 17th Annual Conference on Principles of Programming Languages*, pp. 151-156. San Francisco, January 1990.

[GomSes91] Carsten K. Gomard and Peter Sestoft, Globalization and Live Variables. In *Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pp. 166-177. New Haven, June 1991.

[Hughes85] J. Hughes, Strictness Detection in Non-Flat Domains. In *Proceedings of the Workshop on Programs as Data Objects* (Copenhagen), H. Ganzinger and N. Jones, eds. Springer-Verlag LNCS 217, 1985.

[Hughes87] J. Hughes, *Backwards Analysis of Functional Programs.* University of Glasgow research report CSC/87/R3, March 1987.

[Hughes90] J. Hughes, Compile-Time Analysis of Functional Programs. In *Research Topics in Functdional Programming*, D. Turner, ed. Addison-Wesley, 1990.

[HL91] J. Hughes and John Launchbury, *Towards Relating Forwards and Backwards Analyses.* Draft Manuscript Glasgow University, 1991.

[HunSan91] Sebastian Hunt and David Sands, Binding Time Analysis: A New PERspective. In *Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pp. 154-165. New Haven, June 1991.

[JonLeM89] Simon B. Jones and Daniel Le Métayer, Compile-time Garbage Collection by Sharing Analysis. In *Functional Programming Languages and Computer Architecture*, pp. 54-74. London, September 1989.

[Kamin90] S. Kamin, *Head Strictness is not an Abstract Property.* Draft Manuscript, University of Illinois at Urbana-Champaign, 1991.

[Mycroft80] Alan Mycroft, The Theory and Practice of Transforming Call-By-Need into Call-By-Value. In *Proceedings of the 4th International Symposium on Programming*, pp. 269-281. Springer-Verlag LNCS 83, 1980.

[ParGol91] Young Gil Park and Benjamin Goldberg, Reference Escape Analysis: Optimizing Reference Counting Based on the Lifetime of References. In *Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pp. 166-177. New Haven, June 1991.

[SMR91] R.C. Sekar, P. Mishra, I.V. Ramakrishnan, On the Power and Limitation of Strictness Analysis Based on Abstract Interpretation, in *The 18th Annual Conference on Principles of Programming Languages*, pp. 37-48. Orlando, 1991.

[WH87] P. Wadler and J. Hughes, Projections for Strictness Analysis. In G. Kahn, editor, *Proceedings of the Functional Programming Languages and Computer Architecture Conference*, pp. 385-407. Springer-Verlag LNCS 274, September 1987.

A Proof of Proposition 1

$\subseteq$ Assume $f \in D$. We need to show that $\mathcal{A}_n(f) \sqsubseteq \phi(D)$. In other words, if $\phi(D)(\vec{b}) = 0$, then $\mathcal{A}_n(f)(\vec{b}) = 0$. Assume the contrary, that for some $\vec{b}$, $\phi(D)(\vec{b}) = 0$, but $\mathcal{A}_n(f)(\vec{b}) = 1$. Then from the definition of $\mathcal{A}_n$, for some $\vec{v}$, $\forall 1 \leq i \leq n$, $\mathcal{A}_0(v_i) \sqsubseteq b_i$, but $\mathcal{A}_0(f(\vec{v})) \neq 0$, i.e. $f(\vec{v}) \sqsupseteq \perp$. Fix this $\vec{v}$.

Since $f \in D$, we must have for some $1 \leq i \leq m$, that

$$\mathbf{STR}(f(\vec{v})) = \mathbf{STR}(f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_1))).$$

Fix i. Now, since $f(\vec{v}) \neq \perp$, we must have $\mathbf{STR}(f(\vec{v})) \sqsupseteq \perp$. So we also have

$$\mathbf{STR}(f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_1))) \sqsupseteq \perp.$$

Then $f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_1)) \sqsupseteq \perp$, since $\mathbf{STR} \sqsubset \mathbf{ID}$. This means that for all j, $\alpha_{ij}(v_j) \neq \lightning$, since, if for some j, $\alpha_{ij}(v_j) = \lightning$, then $f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_1)) = \lightning$. First, we know that α_{ij} is not **FAIL**, since $\mathbf{FAIL}(v_j) = \lightning$. Now, since α_{ij} is either **STR** or **ID**, this means that $\alpha_{ij}(v_j) = v_j$ for all j.

We have that $\phi(D)(\vec{b}) = 0$. Then from the definition of ϕ, for all $1 \leq k \leq m$, we must have

$$\overline{\alpha_{k1}}(b_1) \wedge ... \wedge \overline{\alpha_{kn}}(b_n) = 0.$$

Specifically, we must have

$$\overline{\alpha_{i1}}(b_1) \wedge ... \wedge \overline{\alpha_{in}}(b_n) = 0.$$

This means that, for some $1 \leq j \leq n$, $\overline{\alpha_{ij}}(b_j) = 0$. But since $\overline{\mathbf{ID}}(b_j) = 1$, we must have $\alpha_{ij} = \mathbf{STR}$ (we already ruled out any α_{ij} being **FAIL**). Thus, $\overline{\mathbf{STR}}(b_j) = 0$, which implies that $b_j = 0$. Now, by assumption, $\mathcal{A}_0(v_j) \sqsubseteq b_j$, so $\mathcal{A}_0(v_j) = 0$, implying that $v_j = \perp$. But then $\alpha_{ij}(v_j) = \mathbf{STR}(\perp) = \lightning \neq \perp = v_j$, contradicting the conclusion of the previous paragraph.

$\supseteq$ Assume $\mathcal{A}_n(f) \sqsubseteq \phi(D)$. If $\phi(D)(\vec{b}) = 0$, then $\mathcal{A}_n(f)(\vec{b}) = 0$. We need to show that $f \in D$. Assume the contrary. We will show that there exists a $\vec{b}$ for which $\phi(D)(\vec{b}) = 0$, but $\mathcal{A}_n(f)(\vec{b}) = 1$, deriving a contradiction.

Since we don't have $f \in D$, there is some $\vec{v}$ such that for all $1 \leq i \leq m$,

$$\mathbf{STR}(f(\vec{v})) \neq \mathbf{STR}(f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_n))).$$

Fix $\vec{v}$. For this to be the case, for each $1 \leq i \leq m$, there must be some $1 \leq j_i \leq n$, for which $\alpha_{ij_i}(v_{j_i}) \neq v_{j_i}$. Clearly, either α_{ij_i} must be **STR**, and v_{j_i} must be $\perp$, or α_{ij_i} must be **FAIL**. Further, $f(\vec{v}) \sqsupseteq \perp$, for otherwise we would have to have $f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_n)) \sqsubseteq \perp$, and then $\mathbf{STR}(f(\vec{v})) = \mathbf{STR}(f(\alpha_{i1}(v_1), ..., \alpha_{in}(v_n))$ would hold since $\mathbf{STR}(x) = \lightning$, if $x \sqsubseteq \perp$.

Let $\vec{b}$ be defined by $\forall 1 \leq k \leq n, b_k = \mathcal{A}_0(v_k)$. Now

$$\phi(D)(\vec{b}) = \begin{array}{ll} & \alpha_{11}(\mathcal{A}_0(v_1)) \wedge ... \wedge \alpha_{1n}(\mathcal{A}_0(v_n)) \\ & \vdots \\ \vee & \alpha_{m1}(\mathcal{A}_0(v_1)) \wedge ... \wedge \alpha_{mn}(\mathcal{A}_0(v_n)) \end{array}$$

Recall that for all $1 \leq i \leq m$, there is some $1 \leq j_i \leq n$, for which either $\alpha_{ij_i} = \mathbf{STR}$, and $v_{j_i} = \perp$ or $\alpha_{ij_i} = \mathbf{FAIL}$. But then for all $1 \leq i \leq m$,

$$\overline{\alpha_{ij_i}}(\mathcal{A}_0(v_{j_i})) = \overline{\mathbf{STR}}(\mathcal{A}_0(\perp)) = \overline{\mathbf{STR}}(0) = 0, \text{ or}$$
$$\overline{\alpha_{ij_i}}(\mathcal{A}_0(v_{j_i})) = \overline{\mathbf{FAIL}}(\mathcal{A}_0(v_{j_i})) = 0$$

So $\phi(D)(\vec{b}) = 0 \vee ... \vee 0 = 0$.

Now it remains to show that $\mathcal{A}_n(f)(\vec{b}) = 1$, giving a contradiction. From the definition of $\mathcal{A}_n$, we only need to show a $\vec{v}$ for which $\mathcal{A}_0(v_i) \sqsubseteq b_i$, but $\mathcal{A}_0(f(\vec{v})) = 1$. This is simply $\vec{v}$ that we fixed above. We defined $b_i = \mathcal{A}_0(v_i)$, so we certainly have $\mathcal{A}_0(v_i) \sqsubseteq b_i$. And we showed above that $f(\vec{v}) \neq \perp$, so $\mathcal{A}_0(f(\vec{v})) = 1$.

□

B Proof of Main Theorem

PROOF: By induction on the structure of e.

(const) Since we are limited to **ID**, **STR** and **FAIL**, $NS = \{\mathbf{ID}\}$, since **STR** and **FAIL** are strict. So the only statement we can derive is

$$A, B \vdash c[\vec{x}] \in \mathbf{STR} \Longrightarrow [\mathbf{ID}, ..., \mathbf{ID}].$$

So $\phi(D)(\vec{b}) = 1 \wedge 1 \wedge ... \wedge 1 = 1$. From the definition of $\mathcal{E}^\#$, we have $\mathcal{E}^\#[\![c]\!]\varphi(\vec{b}) = 1$.

(var) Again, since $NS = \{\mathbf{ID}\}$, the only statement we can derive is

$$A, B \vdash x_i[\vec{x}] \in \mathbf{STR} \Longrightarrow [\mathbf{ID}, ..., \mathbf{STR}, ..., \mathbf{ID}].$$

From this we have $\phi(D)(\vec{b}) = 1 \wedge ... \wedge \overline{\mathbf{STR}}(b_i) \wedge ... \wedge 1 = b_i$. From the definition of $\mathcal{E}^\#$, we have $\mathcal{E}^\#[\![x_i]\!]\varphi(\vec{b}) = b_i$.

(app) By assumption, we have that $\phi(A(f)) = \varphi[\![f]\!]$, and by induction we have, for all i, that $\phi(D_i) = \mathcal{E}^\#[\![e_i]\!]\varphi$. So we have that

$$\begin{aligned}&\mathcal{E}^\#[\![f(e_1, ..., e_m)]\!]\varphi(\vec{b})\\ &= (\varphi[\![f]\!])(\mathcal{E}^\#[\![e_1]\!]\varphi(\vec{b}), ..., \mathcal{E}^\#[\![e_m]\!]\varphi(\vec{b}))\\ &= \phi(A(f))(\phi(D_1)(\vec{b}), ..., \phi(D_m)(\vec{b}))\end{aligned}$$

Let

$$D = \begin{pmatrix} & \mathbf{STR} \Longrightarrow [\alpha_{11}, ..., \alpha_{1m}] \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{21}, ..., \alpha_{2m}] \\ \vee & ... \\ \vee & \mathbf{STR} \Longrightarrow [\alpha_{d1}, ..., \alpha_{dm}] \end{pmatrix}$$

And for each D_j, let

$$D_j = \begin{pmatrix} & \mathbf{STR} \Longrightarrow [\gamma_{j11}, ..., \gamma_{j1n}] \\ \vee & \mathbf{STR} \Longrightarrow [\gamma_{j21}, ..., \gamma_{j2n}] \\ \vee & ... \\ \vee & \mathbf{STR} \Longrightarrow [\gamma_{jd_j1}, ..., \gamma_{jd_jn}] \end{pmatrix}$$

Then we have

$$\phi(D)(\vec{b}) = \bigvee_{i=1}^{d} \bigwedge_{j=1}^{m} \overline{\alpha_{ij}}(b_j),$$

and

$$\phi(D_j)(\vec{b}) = \bigvee_{k_j=1}^{d_j} \bigwedge_{l=1}^{n} \overline{\gamma_{jk_jl}}(b_l).$$

So we have that

$$\begin{aligned}&\phi(A(f))(\phi(D_1)(\vec{b}), ..., \phi(D_m)(\vec{b}))\\ &= \textstyle\bigvee_{i=1}^{d} \bigwedge_{j=1}^{m} \overline{\alpha_{ij}}(\phi(D_j)(\vec{b}))\\ &= \textstyle\bigvee_{i=1}^{d} \bigwedge_{j=1}^{m} \overline{\alpha_{ij}}(\bigvee_{k_j=1}^{d_j} \bigwedge_{l=1}^{n} \overline{\gamma_{jk_jl}}(b_l)).\end{aligned}$$

Here, i indexes the choice of disjunct in the function's property (D), j indexes the arguments to the function, k_j indexes the choice of disjunct from the jth expression's strictness property (D_j), and l indexes the variable number.

Because $\overline{\alpha_{ij}}$ is monotonic in *Bool*, we can move it inside the $\vee$ and $\wedge$ to yield

$$\bigvee_{i=1}^{d} \bigwedge_{j=1}^{m} \bigvee_{k_j=1}^{d_j} \bigwedge_{l=1}^{n} \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b_l)).$$

Now, by distributivity we can move the $\vee_{k_j}$ outside as follows:

$$\bigvee_{i=1}^{d} \bigvee_{k_1=1}^{d_1} ... \bigvee_{k_m=1}^{d_m} \bigwedge_{j=1}^{m} \bigwedge_{l=1}^{n} \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b_l)).$$

Let's work on the other side now. We can derive

$$A, B \vdash f(e_1, ..., e_m) \in D'$$

with

$$D' = \bigvee_{i=1}^{d} \bigvee_{k_1=1}^{d_1} ... \bigvee_{k_m=1}^{d_m} \bigwedge_{(j,j_1,...,j_m) \in S} \Psi_{m,n}(\vec{\nu})$$

where $\vec{\nu} = (\nu_{0ij}, \nu_{1k_1j_1}, ..., \nu_{mk_mj_m})$ and $S = \{(j, j_1, ..., j_m) \mid \Theta(\nu_{0ij}, \nu_{1k_1j_1}, ..., \nu_{mk_mj_m})\}$.

We are interested in what $\phi(D')$ is. Noting from the definition of ϕ that $\phi(\vee \wedge \nu) = \vee\phi(\wedge\nu)$, let's look at the conjunction

$$\phi(\bigwedge_{(j,j_1,\ldots,j_m)\in S} \Psi_{m,n}(\nu_{0ij}, \nu_{1k_1j_1}, \ldots, \nu_{mk_mj_m}))$$

Note that the conjunction will have only one conjunct of the form $\mathbf{STR} \Longrightarrow \ldots$. Recalling the definitions of $\Theta_{m,n}$ and $\Psi_{m,n}$, and knowing that ϕ only looks at properties of the form $\mathbf{STR} \Longrightarrow \ldots$, we are only concerned with the case where $j = 1$. Let $\nu_{0i1} = \mathbf{STR} \Longrightarrow [\alpha_{i1}, \ldots, \alpha_{im}]$. Having fixed $j = 1$, the j_l are fixed, since there is only one property per conjunct which has the form $\alpha_{ij} \Longrightarrow \ldots$.

Now

$$\begin{aligned}&\phi(\Psi_{m,n}(\nu_{0i1}, \nu_{1k_1j_1}, \ldots, \nu_{mk_mj_m}))\\ &\quad = \phi(\mathbf{STR} \Longrightarrow [\&_j\{\delta_{j1}\}, \ldots, \&_j\{\delta_{jn}\}])\end{aligned}$$

where

$$\delta_{jl} = \begin{cases} \gamma_{jk_jl}, & \alpha_{ij} = \mathbf{STR} \\ \mathbf{ID}, & \alpha_{ij} = \mathbf{ID} \\ \mathbf{FAIL}, & \alpha_{ij} = \mathbf{FAIL} \end{cases}.$$

Note that

$$\begin{aligned}&\overline{\delta_{jl}}(b) = \overline{\gamma_{jk_jl}}(b) = \overline{\mathbf{STR}}(\overline{\gamma_{jk_jl}}(b)) = \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b)), \text{or}\\ &\overline{\delta_{jl}}(b) = \overline{\mathbf{ID}}(b) = \overline{\mathbf{ID}}(\overline{\gamma_{jk_jl}}(b)) = \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b)), \text{or}\\ &\overline{\delta_{jl}}(b) = \overline{\mathbf{FAIL}}(b) = \overline{\mathbf{FAIL}}(\overline{\gamma_{jk_jl}}(b)) = \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b))\end{aligned}$$

so we have that $\overline{\delta_{jl}}(b) = \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b))$.

So $\phi(\Psi_{m,n}(\nu_{0i1}, \nu_{1k_1j_1}, \ldots, \nu_{mk_mj_m}))$ gives

$$\overline{\&_j\{\delta_{j1}\}}(b_1) \wedge \ldots \wedge \overline{\&_j\{\delta_{jn}\}} = \bigwedge_{l=1}^{n} \overline{\&_j\{\delta_{jl}\}}(b_l).$$

Note that for $\alpha, \beta \in \{\mathbf{ID}, \mathbf{STR}, \mathbf{FAIL}\}$,

$$\begin{aligned}\mathbf{ID\&ID} &= \mathbf{ID}\\ \mathbf{ID\&STR} &= \mathbf{STR}\\ \mathbf{ID\&FAIL} &= \mathbf{FAIL}\\ \mathbf{STR\&STR} &= \mathbf{STR}\\ \mathbf{STR\&FAIL} &= \mathbf{FAIL}\\ \mathbf{FAIL\&FAIL} &= \mathbf{FAIL}\end{aligned}$$

We can see case by case that from the definition of $\overline{\alpha}$, $\overline{(\alpha\&\beta)}(b) = \overline{\alpha}(b) \wedge \overline{\beta}(b)$. So we get

$$\bigwedge_{l=1}^{n} \bigwedge_{j=1}^{m} \overline{\delta_{jl}}(b_l)$$

and so replacing the definition of $\overline{\delta_{jl}}$, and swapping the $\wedge$'s:

$$\bigwedge_{j=1}^{m} \bigwedge_{l=1}^{n} \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b_l)).$$

Finally, we have

$$\bigvee_{i=1}^{d} \bigvee_{k_1=1}^{d_1} \ldots \bigvee_{k_m=1}^{d_m} \bigwedge_{j=1}^{m} \bigwedge_{l=1}^{n} \overline{\alpha_{ij}}(\overline{\gamma_{jk_jl}}(b_l)),$$

matching our result for $\mathcal{E}^{\#}[\![f(e_1, \ldots, e_m)]\!]\varphi(\vec{b})$.

□