Static Extensivity Analysis for λ-Definable Functions Over Lattices

Hyunjun EO

Division of Computer Science
Korea Advanced Institute of Science and Technology
373-1 Guseong-dong Yuseong-gu, Daejeon 305-701, Korea
`poisson@ropas.kaist.ac.kr`

Kwangkeun YI

School of Computer Science and Engineering
Seoul National University
San 56-1 Sillim-dong Gwanak-gu, Seoul 151-742, Korea
`kwang@ropas.snu.ac.kr`

Kwang-Moo CHOE

Division of Computer Science
Korea Advanced Institute of Science and Technology
373-1 Guseong-dong Yuseong-gu, Daejeon 305-701, Korea
`choe@cs.kaist.ac.kr`

Received 23 April 2004

Abstract We employ a static analysis to examine the extensivity ($\forall x : x \leq f(x)$) of functions defined over lattices in a λ-calculus augmented with lattice operations. The need for such a verification procedure has arisen in our work on a generator system (called Zoo) of static program-analyzers. The input to Zoo is a static analysis specification that consists of finite-height lattice definitions and function definitions over the lattices. Once the extensivity of the functions is ascertained, the generated analyzer is guaranteed to terminate. The extensivity analysis consists of a sound syntax-driven deductive rules whose satisfiability check is done by a constraint solving procedure.

Keywords Static analysis, Functional static-analysis specification, Function extensivity, Deductive system, Constraints solving

§1 Motivation

We present a static analysis to check the extensivity ($\forall x : x \leq f(x)$) of functions defined over lattices in a λ-calculus augmented with constants, branching, least upper bounds, greatest lower bounds and recursive definitions. Our motivation comes from our work on a program-analyzer generator system.

We have been involved in a project to build a program-analyzer generator (named "Zoo"[15]). One of the program analysis frameworks that Zoo supports is abstract interpretation[2, 3]. Its user (analysis designer) defines an abstract interpreter in a specification language. Zoo then compiles the input specification into an executable analyzer (encoded in C and ML) which, given an input program to analyze, derives a set of data-flow equations and solves them by fixpoint iterations.

Our goal is to have the Zoo system not blindly generate an executable analyzer without verifying that the input specification qualifies for a static analysis. (This is analogous that the Yacc checks whether the input context-free-grammar specification has no ambiguity.) One qualification to check is that the input specification must define a terminating analysis. The Zoo system should be able to automatically check if the user-specified abstract interpreter defines a terminating analysis.

Such a termination check of an input abstract interpreter can be done by checking the extensivity of given functions in the input specification. The extensivity is a sufficient condition for the generated analyzer to always terminate. An abstract interpreter consists of lattice definitions and definitions of functions over the lattices. Once it is known that the functions are extensive ($\forall x : x \leq f(x)$), the generated analyzers are guaranteed to terminate because the function f is defined to range over a finite-height lattice. The generated analyzer computes the fixpoint sequence $\perp, f(\perp), f(f(\perp)), \cdots$ ($\perp$ is the least element of the f's domain lattice) and the extensivity of f guarantees that the sequence is an increasing chain whose length is limited because we only take a finite-height lattice as the domain of function f. By realizing such an extensivity check by yet again a static analysis, Zoo can statically estimate the extensivity of the input functions and consequently reject analyzer specifications that are possibly not extensive.

Existing results on function's property check, for example, in learning theory[14, 6, 7, 13], have turned out hardly usable in our case. They are restricted to boolean lattices and concern functions of type $\{0, 1\}^n \rightarrow \{0, 1\}$. Though finite distributive lattices can be embedded in a product of the boolean lattices[12], our Zoo system also supports non-distributive lattices which are prevalent in static program analysis. Also, their property-check algorithms are probabilistic, and so are allowed to err with some small probability. In our generalized case, where domains and lattices are not restricted to boolean nor distributive, finding a tight bound on this probability of mistakes seems a formidable job. Besides, only functions *in extenso* have been considered thus far, whereas we also have access to the definitions. This makes the problem amenable to static analysis. Our approach is to design a static analysis which reliably ensures the function's extensivity property with a reasonable accuracy.

In a disjunctive combination with the previous work[11] on the static monotonicity analysis(checking $\forall x \leq y : f(x) \leq f(y)$), our extensivity analysis enlarges the set of input specifications accepted by Zoo. This is because the extensivity and the monotonicity are incomparable properties and either of them guarantees the termination of the generated analyzers that are defined over finite-height lattices. For example, "$\lambda x.\mathit{if}\ x \sqsubseteq c\ \mathit{then}\ \bot\ \mathit{else}\ c$" is monotonic but not extensive, while "$\lambda x.\mathit{if}\ x = c\ \mathit{then}\ c\ \mathit{else}\ \top$" is extensive but not monotonic. Yet, each of the fixpoint-iteration sequence of the two functions reaches a fixpoint in a finite time: the former to $\bot$ (the least element of the lattice) in one iteration and the latter to $\top$ (the greatest element of the lattice) in two iterations.

§2 Setting

Let L be a lattice. A function $f : L \longrightarrow L$ is extensive (respectively reductive) if and only if for all x, we have $x \leq f(x)$ (respectively $f(x) \leq x$). If a function is both extensive and reductive, it is the identical function. For functions from product lattices (multiple arguments), we can analogously define extensivity and reductivity with respect to the ith component (ith argument).

Our goal is to design a static procedure that can certify whether a function between two lattices is extensive or not. The source language, which is the input specification language of the Zoo system, is a strongly-typed, statically-scoped, functional eager-evaluation language. The core of the language for specifying analysis functions is as follows:

Expression	e	$::=$	c	constant (lattice point)
		$\mid$	x	variable
		$\mid$	$\lambda x.e$	function
		$\mid$	$Y\lambda x.e$	recursive function
		$\mid$	$e\ e$	application
		$\mid$	$e \sqcup_L e$	least upper bound (join)
		$\mid$	$e \sqcap_L e$	greatest lower bound (meet)
		$\mid$	$\mathit{if}\ e \sqsubseteq_L e\ ?\ e\ :\ e$	branching
Type	τ	$::=$	L	user-defined lattice
		$\mid$	$\tau \rightarrow \tau$	function type

Values in this language are either lattice elements or functions over lattices. Hence a type τ is either a lattice L or a function $\tau \rightarrow \tau$ between types. c is a constant expression denoting a lattice element. The *if* expression branches, as usual, depending on whether the conditional partial-order relation holds or not. The usual evaluation rule for $e \hookrightarrow v$(expression e computes v) is in Figure 1. Notation $\{v/x\}e$ denotes the result from substituting v for free variable x in e. (In the actual specification language[15], user-definable lattices include product lattices, powerset lattices, function lattices, and lattices with user-defined orders.)

Throughout this paper we assume that every variable is uniquely named and a unique mono-type is associated to each expression.

§3 Extensivity Checking by a Deductive System

Given an expression e of the core language, our extensivity check will conservatively determine whether the operation

$$(x_1, \cdots, x_n) \mapsto e$$

is identical, extensive, reductive, or unknown with respect to each of its free variables $x_1, \cdots, x_n$.

Example 3.1

"$(x \sqcap_L c) \sqcup_L y$" is at least as large as y hence extensive for y but unknown for x. "$\mathit{if}\ x \sqsubseteq_L c\ ?\ \top_L\ :\ x \sqcup_L y$" is at least as large as both x and y hence extensive for both x and y.

$$\textit{Value} \qquad v \ ::= \ c \mid x \mid \lambda x.e \mid Y v$$

$$\frac{}{c \hookrightarrow c} \quad \frac{}{x \hookrightarrow x}$$

$$\frac{}{\lambda x.e \hookrightarrow \lambda x.e} \quad \frac{}{Y\lambda x.e \hookrightarrow Y\lambda x.e}$$

$$\frac{e_1 \hookrightarrow \lambda x.e \quad e_2 \hookrightarrow v' \quad \{v'/x\}e \hookrightarrow v''}{e_1\ e_2 \hookrightarrow v''} \qquad \frac{e_1 \hookrightarrow Yv \quad e_2 \hookrightarrow v' \quad v(Yv) \hookrightarrow \lambda x.e \quad \{v'/x\}e \hookrightarrow v''}{e_1\ e_2 \hookrightarrow v''}$$

$$\frac{e_1 \hookrightarrow c_1 \quad e_2 \hookrightarrow c_2}{e_1 \sqcup_L e_2 \hookrightarrow c_1 \sqcup_L c_2} \quad \frac{e_1 \hookrightarrow c_1 \quad e_2 \hookrightarrow c_2}{e_1 \sqcap_L e_2 \hookrightarrow c_1 \sqcap_L c_2}$$

$$\frac{e_1 \hookrightarrow c_1 \quad e_2 \hookrightarrow c_2 \quad v_1 \sqsubseteq_L v_2 \quad e_3 \hookrightarrow v_3}{\textit{if } e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4 \hookrightarrow v_3} \quad \frac{e_1 \hookrightarrow c_1 \quad e_2 \hookrightarrow c_2 \quad v_1 \not\sqsubseteq_L v_2 \quad e_4 \hookrightarrow v_4}{\textit{if } e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4 \hookrightarrow v_4}$$

Fig. 1 Evaluation rules

We present the checking procedure as an inference system for judgments of the form

$$\Gamma \vdash e : \mathsf{xb}.$$

The judgments should be read as "under assumption Γ, expression e has extensivity behavior xb".

Extensivity assumption Γ and behavior xb range over the following domains:

Definition 3.1

$$\begin{array}{rcllll}
\Gamma & \in & \mathit{XE} & = & \mathit{Var} \stackrel{\text{fin}}{\to} \mathit{XB} & \text{extensivity assumption} \\
\mathsf{xb} & \in & \mathit{XB} & = & X + \mathit{XB} \times \mathit{XB} & \text{extensivity behavior} \\
\mathsf{x} & \in & X & = & \sum_{\tau \in \mathit{Type}} (\mathit{Var}_L \stackrel{\text{fin}}{\to} T) & \text{non-functional extensivity} \\
\mathsf{xb} \to \mathsf{xb} & \in & & & \mathit{XB} \times \mathit{XB} & \text{functional extensivity} \\
t & \in & T & = & \{\bot_T, 0, +, -, \top_T\} & \text{extensivity token} \\
x, y, z & \in & \mathit{Var} & & & \text{variables} \\
 & & \mathit{Var}_L & & & \text{variables of lattice } L
\end{array}$$

The extensivity assumption

$$\Gamma \in Var \stackrel{\text{fin}}{\to} XB$$

is a finite map from variables to extensivity behaviors XB. The extensivity behavior xb is either a pair

$$\mathsf{xb} \to \mathsf{xb} \in XB \times XB$$

of extensivity behaviors for function-typed expressions, or a map

$$\mathsf{x} \in X = \sum_{L \in Type} (Var_L \stackrel{\text{fin}}{\to} T)$$

from variables of lattice L to extensivity tokens. This typefulness of XB (constructively implied by the typefulness of X) is eligible because every expression is uniquely typed and its extensivity is, by definition, in terms of variables of the same type.

For $\mathsf{xb} \in XB$ or $x \in Var$, we sometimes write $\mathsf{xb}_{type(\mathsf{xb})}$ or $x_{type(x)}$ to expose their types. For extensivity behavior of lattice L, we simply write $\{x_1^{t_1}, \cdots, x_n^{t_n}\}$ to mean

$$\{x_1 \mapsto t_1, \cdots, x_n \mapsto t_n\} \cup \{y \mapsto \top_T \mid y \in Var_L \setminus \{x_1, \cdots, x_n\}\}.$$

We write $\{s\}e$ for the expression resulting from substituting s's images for the free variables in e:

Definition 3.2
$\{s\}e = \{s(x_1)/x_1\} \cdots \{s(x_n)/x_n\}e$ where $FV(e) = \{x_1, \cdots, x_n\}$.

Definition 3.3
The extensivity behavior xb has the following meaning $[\![\mathsf{xb}]\!]$:

$$\begin{array}{rcl}
[\![\mathsf{x}]\!] & = & \cap\{[\![x^t]\!] \mid \mathsf{x}(x) = t\} \\
[\![\mathsf{xb}_1 \to \mathsf{xb}_2]\!] & = & \{e \mid \forall e_1 \in [\![\mathsf{xb}_1]\!] : e\, e_1 \in [\![\mathsf{xb}_2]\!]\}
\end{array}$$

where

$$\begin{array}{rcl}
[\![x^0]\!] & = & \{e \mid x \in FV(e) \ \wedge\ (\forall s : \{s\}e \hookrightarrow v \text{ implies } s(x) = v)\} \\
[\![x^+]\!] & = & \{e \mid x \in FV(e) \ \wedge\ (\forall s : \{s\}e \hookrightarrow v \text{ implies } s(x) \sqsubseteq_L v)\} \\
[\![x^-]\!] & = & \{e \mid x \in FV(e) \ \wedge\ (\forall s : \{s\}e \hookrightarrow v \text{ implies } v \sqsubseteq_L s(x))\} \\
[\![x^{\top_T}]\!] & = & \text{all expressions} \\
[\![x^{\perp_T}]\!] & = & \emptyset
\end{array}$$

Note the extensivity tokens T form a lifted diamond-shaped lattice:

$$\begin{array}{ccc} & \top_T & \\ / & & \backslash \\ + & & - \\ \backslash & & / \\ & 0 & \\ & | & \\ & \bot_T & \end{array} \qquad \bot_T \sqsubseteq_T 0 \sqsubseteq_T + \sqsubseteq_T \top_T \quad \text{and} \quad \bot_T \sqsubseteq_T 0 \sqsubseteq_T - \sqsubseteq_T \top_T.$$

The partial order in XB follows from the meanings of XB elements:

Definition 3.4
The order in X is point-wise:

$$\mathsf{x}_L \sqsubseteq_X \mathsf{x}'_L \;\; \mathit{iff} \;\; \forall x \in \mathit{Var}_L : \mathsf{x}(x) \sqsubseteq_X \mathsf{x}'(x)$$

and the order in $XB \times XB$ is contra-variant on the first component:

$$\mathsf{xb}_1 \rightarrow \mathsf{xb}_2 \sqsubseteq_{XB} \mathsf{xb}'_1 \rightarrow \mathsf{xb}'_2 \;\; \mathit{iff} \;\; \mathsf{xb}'_1 \sqsubseteq_{XB} \mathsf{xb}_1 \;\wedge\; \mathsf{xb}_2 \sqsubseteq_{XB} \mathsf{xb}'_2.$$

Now we present the extensivity checking rules $\Gamma \vdash e : \mathsf{xb}$. For constant expression, we have three cases, $\bot_L$, $\top_L$, and other constant c:

$$\frac{}{\Gamma \vdash \bot_L : \{x \mapsto - \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}}$$

$$\frac{}{\Gamma \vdash \top_L : \{x \mapsto + \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}}$$

$$\frac{}{\Gamma \vdash c_L : \{x \mapsto \top_T \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}}$$

The extensivity of a variable is assumed in the environment:

$$\frac{\Gamma(x) = \mathsf{xb}}{\Gamma \vdash x : \mathsf{xb}}$$

The extensivity of the least upper bound is compositional:

$$\frac{\Gamma \vdash e_1 : \mathsf{xb}_1 \quad \Gamma \vdash e_2 : \mathsf{xb}_2}{\Gamma \vdash e_1 \sqcup_L e_2 : \mathsf{xb}_1 \oplus \mathsf{xb}_2}$$

The e_1 and e_2 evaluate into elements of lattice L. Hence $\forall i \in \{1, 2\} : \mathsf{xb}_i \in X_L$. The $\oplus$ operation over X is point-wise:

$$\mathsf{x}_L \oplus \mathsf{x}'_L \quad = \quad \{x \mapsto (\mathsf{x}(x) \oplus \mathsf{x}'(x)) \mid x \in \mathit{Var}_L\}$$

The commutative $\oplus$ operation for the extensivity tokens is determined following the effect of moving-up($\sqcup_L$) operation:

$\oplus$	0	$+$	$-$	$\top_T$
0	0	$+$	0	$+$
$+$	$+$	$+$	$+$	$+$
$-$	0	$+$	$-$	$\top_T$
$\top_T$	$+$	$+$	$\top_T$	$\top_T$

Let's consider the extensivity of $e_1 \sqcup_L e_2$ for x.

- If e_1 and e_2 are identical to x then $e_1 \sqcup_L e_2$ is also identical to x. Hence $0 \oplus 0 = 0$.
- If e_1 is identical to x and e_2 is extensive for x ($x \sqsubseteq_L e_2$) then $e_1 \sqcup_L e_2$ is extensive for x ($x \sqsubseteq_L e_1 \sqcup_L e_2$). Hence $0 \oplus + = +$.
- If e_1 is identical to x and e_2 is reductive for x ($e_2 \sqsubseteq_L x$) then $e_1 \sqcup_L e_2$ is identical to x. Hence $0 \oplus - = 0$.
- If e_1 is identical to x and e_2 is unknown for x then $e_1 \sqcup_L e_2$ is extensive for x ($x \sqsubseteq_L e_1 \sqcup_L e_2$). Hence $0 \oplus \top_T = +$.
- If e_1 is extensive for x ($x \sqsubseteq_L e_1$) then for every e_2, $e_1 \sqcup_L e_2$ is extensive for x ($x \sqsubseteq_L e_1 \sqcup_L e_2$). Hence $+ \oplus \bullet = +$.
- If e_1 and e_2 are reductive for x ($e_1 \sqsubseteq_L x$, $e_2 \sqsubseteq_L x$) then $e_1 \sqcup_L e_2$ is also reductive for x ($e_1 \sqcup_L e_2 \sqsubseteq_L x$). Hence $- \oplus - = -$.
- If e_1 is reductive for x ($e_1 \sqsubseteq_L x$) and e_2 is unknown for x then $e_1 \sqcup_L e_2$ is unknown for x. Hence $- \oplus \top_T = \top_T$.
- If e_1 and e_2 are unknown for x then so does $e_1 \sqcup_L e_2$. Hence $\top_T \oplus \top_T = \top_T$.

Similarly, we use the $\ominus$ operation for the extensivity of the greatest lower bound:

$$\frac{\Gamma \vdash e_1 : \mathsf{xb}_1 \quad \Gamma \vdash e_2 : \mathsf{xb}_2}{\Gamma \vdash e_1 \sqcap_L e_2 : \mathsf{xb}_1 \ominus \mathsf{xb}_2}$$

The $\ominus$ operation is point-wise, just as the $\oplus$ case. For $i \in \{1, 2\}$, the type of e_i is a lattice L, i.e., $\mathsf{xb}_i \in X_L$.

$$\mathsf{x}_L \ominus \mathsf{x}'_L \quad = \quad \{x \mapsto (\mathsf{x}(x) \ominus \mathsf{x}'(x)) \mid x \in \mathit{Var}_L\}$$

The commutative $\ominus$ operation for the extensivity tokens is dual to $\oplus$; 0 is opposite to $\top_T$, and $+$ is opposite to $-$.

$\ominus$	0	$+$	$-$	$\top_T$
0	0	0	$-$	$-$
$+$	0	$+$	$-$	$\top_T$
$-$	$-$	$-$	$-$	$-$
$\top_T$	$-$	$\top_T$	$-$	$\top_T$

The rule for lambda expressions is similar to standard typing. The extensivity behaviors of the argument and the result are determined. Note that the extensivity of the result (xb_2) can be weaker ($\mathsf{xb}_2' \sqsubseteq_{XB} \mathsf{xb}_2$) than that of the body (xb_2'). This relaxation makes the rule less restrictive; without this relaxation we would have to reject analysis specifications in which two functions of varying extensivities are called in the same application. Lastly, because the extensivity of a function is determined in the calling context of it, the extensivity of the function should be independent of the formal parameter:

$$\frac{\Gamma + x : \mathsf{xb}_1 \vdash e : \mathsf{xb}_2' \quad \mathsf{xb}_2' \sqsubseteq_{XB} \mathsf{xb}_2}{\Gamma \vdash \lambda x.e : \mathsf{xb}_1 \to \mathsf{xb}_2}$$

The rule for a recursive function requires that the function name and its body have the same extensivity:

$$\frac{\Gamma \vdash \lambda x.e : \mathsf{xb} \to \mathsf{xb}}{\Gamma \vdash Y \lambda x.e : \mathsf{xb}}$$

The rule for application has nothing particular.

$$\frac{\Gamma \vdash e_1 : \mathsf{xb}_1 \to \mathsf{xb}_2 \quad \Gamma \vdash e_2 : \mathsf{xb}_1' \quad \mathsf{xb}_1' \sqsubseteq_{XB} \mathsf{xb}_1}{\Gamma \vdash e_1\ e_2 : \mathsf{xb}_2}$$

Note that the extensivity of function's argument (xb_1) can be weaker ($\mathsf{xb}_1' \sqsubseteq_{XB} \mathsf{xb}_1$) than that of the actual argument (xb_1'). This relaxation makes the rule less restrictive; without this relaxation we would have to reject analysis specifications in which the arguments of varying extensivities are passed to the same function.

The extensivity of the if-expression should subsume those of the two branches:

$$\frac{\Gamma \vdash e_3 : \mathsf{xb}_3 \quad \Gamma \vdash e_4 : \mathsf{xb}_4}{\Gamma \vdash \mathit{if}\ e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4 : \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4}$$

For example, if e_3 is extensive and e_4 is reductive, the result is unknown ($\top_{XB}$).

If we know an order relation between variables, we can further refine the extensivity of expressions. For example, suppose that $x \sqsubseteq_L y$ holds for the two free variables x and y of e and that the extensivity of expression e is $\{y^+\}$. Then we can refine $\{y^+\}$ into $\{x^+, y^+\}$ because x is always less than or equal to y.

We can gather such order relations from conditional expressions. Let's consider checking the extensivity of the true-branch e_3 of *if* $e_1 \sqsubseteq_L e_2 \mathrel{?} e_3 : e_4$. Suppose that e_1 is extensive for x ($x \sqsubseteq_L e_1$) and e_2 is reductive for y ($e_2 \sqsubseteq_L y$). Then we can get $x \sqsubseteq_L y$ because $e_1 \sqsubseteq_L e_2$ holds in the true-branch e_3.

We maintain such order relations in assumption Δ. Order assumption Δ ranges over the following domain:

Definition 3.5

$$\Delta \in 2^{Var \times Var} \quad \text{order assumption.}$$

When $(x, y) \in \Delta$, we write $x \sqsubseteq_L y \in \Delta$.

The judgments for extensivity checking are now changed:

$$\Delta, \Gamma \vdash e : \mathsf{xb}.$$

Following two rules, (REF$-$) and (REF$+$), refines the extensivity of expression e with order assumption Δ:

$$\frac{x \sqsubseteq_L y \in \Delta \qquad \Delta, \Gamma \vdash e : \mathsf{xb} \qquad \mathsf{xb} \sqsubseteq_{XB} \{x^-\}}{\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{y^-\}} \text{ (REF}-\text{)}$$

$$\frac{x \sqsubseteq_L y \in \Delta \qquad \Delta, \Gamma \vdash e : \mathsf{xb} \qquad \mathsf{xb} \sqsubseteq_{XB} \{y^+\}}{\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{x^+\}} \text{ (REF}+\text{)}$$

If $x \sqsubseteq_L y \in \Delta$ and e is reductive for x ($\Delta, \Gamma \vdash e : \mathsf{xb}$ and $\mathsf{xb} \sqsubseteq_{XB} \{x^-\}$), e is also reductive for y ($\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{y^-\}$ and $\mathsf{xb} \sqcap_{XB} \{y^-\} \sqsubseteq_{XB} \{y^-\}$). Dually, if $x \sqsubseteq_L y \in \Delta$ and e is extensive for y, e is also extensive for x.

The starting point for the refinement is an if-expression. From the extensivities of the conditional, we get some order relations and refine the true-branch with them:

$$\frac{\begin{array}{c} \Delta, \Gamma \vdash e_1 : \mathsf{xb}_1 \qquad \Delta, \Gamma \vdash e_2 : \mathsf{xb}_2 \\ \Delta \cup \{x \sqsubseteq_L y \mid \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\}, \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\}\}, \Gamma \vdash e_3 : \mathsf{xb}_3 \\ \Delta, \Gamma \vdash e_4 : \mathsf{xb}_4 \end{array}}{\Delta, \Gamma \vdash \textit{if } e_1 \sqsubseteq_L e_2 \mathrel{?} e_3 : e_4 : \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4} \text{ (IF)}$$

Figure 2 shows every rule of our extensivity checking system. Other rules, except for (REF$-$), (REF$+$) and (IF), just propagate Δ so that at every

$\boxed{\Delta, \Gamma \vdash e : \mathsf{xb}}$ Under order assumption Δ and extensivity assumption Γ, expression e has extensivity behavior xb.

$$\frac{}{\Delta, \Gamma \vdash \bot_L : \{x \mapsto - \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}}\ \text{(CON-1)}$$

$$\frac{}{\Delta, \Gamma \vdash \top_L : \{x \mapsto + \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}}\ \text{(CON-2)}$$

$$\frac{}{\Delta, \Gamma \vdash c_L : \{x \mapsto \top_T \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}}\ \text{(CON-3)}$$

$$\frac{\Gamma(x) = \mathsf{xb}}{\Delta, \Gamma \vdash x : \mathsf{xb}}\ \text{(VAR)}$$

$$\frac{\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1 \quad \Delta, \Gamma \vdash e_2 : \mathsf{xb}_2}{\Delta, \Gamma \vdash e_1 \sqcup_L e_2 : \mathsf{xb}_1 \oplus \mathsf{xb}_2}\ \text{(LUB)}$$

$$\frac{\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1 \quad \Delta, \Gamma \vdash e_2 : \mathsf{xb}_2}{\Delta, \Gamma \vdash e_1 \sqcap_L e_2 : \mathsf{xb}_1 \ominus \mathsf{xb}_2}\ \text{(GLB)}$$

$$\frac{\Delta, \Gamma + x : \mathsf{xb}_1 \vdash e : \mathsf{xb}_2' \quad \mathsf{xb}_2' \sqsubseteq_{XB} \mathsf{xb}_2}{\Delta, \Gamma \vdash \lambda x.e : \mathsf{xb}_1 \to \mathsf{xb}_2}\ \text{(LAM)}$$

$$\frac{\Delta, \Gamma \vdash \lambda x.e : \mathsf{xb} \to \mathsf{xb}}{\Delta, \Gamma \vdash Y \lambda x.e : \mathsf{xb}}\ \text{(REC)}$$

$$\frac{\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1 \to \mathsf{xb}_2 \quad \Delta, \Gamma \vdash e_2 : \mathsf{xb}_1' \quad \mathsf{xb}_1' \sqsubseteq_{XB} \mathsf{xb}_1}{\Delta, \Gamma \vdash e_1\ e_2 : \mathsf{xb}_2}\ \text{(APP)}$$

$$\frac{x \sqsubseteq_L y \in \Delta \qquad \Delta, \Gamma \vdash e : \mathsf{xb} \qquad \mathsf{xb} \sqsubseteq_{XB} \{x^-\}}{\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{y^-\}}\ \text{(REF}-\text{)}$$

$$\frac{x \sqsubseteq_L y \in \Delta \qquad \Delta, \Gamma \vdash e : \mathsf{xb} \qquad \mathsf{xb} \sqsubseteq_{XB} \{y^+\}}{\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{x^+\}}\ \text{(REF+)}$$

$$\frac{\begin{array}{c}\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1 \qquad \Delta, \Gamma \vdash e_2 : \mathsf{xb}_2 \\ \Delta \cup \{x \sqsubseteq_L y \mid \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\}, \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\}\}, \Gamma \vdash e_3 : \mathsf{xb}_3 \\ \Delta, \Gamma \vdash e_4 : \mathsf{xb}_4\end{array}}{\Delta, \Gamma \vdash \mathit{if}\ e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4 : \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4}\ \text{(IF)}$$

Fig. 2 Extensivity Checking System

expression refinement rules, (REF$-$) and (REF$+$), can be applied. In the rest of our presentation, we simply write $\Gamma \vdash e : \mathsf{xb}$ to mean $\emptyset, \Gamma \vdash e : \mathsf{xb}$.

Example 3.2
Expression $e_{3.2} = (x \sqcap_L c_L) \sqcup_L y$ under assumptions $x : \{x^0\}$ and $y : \{y^0\}$ has extensivity $\{x^{\top_T}, y^+\}$:

$$\dfrac{\dfrac{\begin{array}{c} x : \{x^0\}, y : \{y^0\} \vdash x : \{x^0\} \text{ (VAR)} \\ x : \{x^0\}, y : \{y^0\} \vdash c_L : \top_{X_L} \text{ (CON-3)} \end{array}}{x : \{x^0\}, y : \{y^0\} \vdash x \sqcap_L c_L : \{x^0\} \ominus \top_{X_L}}\text{ (GLB)} \qquad x : \{x^0\}, y : \{y^0\} \vdash y : \{y^0\}}{x : \{x^0\}, y : \{y^0\} \vdash (x \sqcap_L c) \sqcup_L y : \{x^{\top_T}, y^+\}}\text{ (LUB)}$$

The extensivity of $e_{3.2}$ for x is unknown ($x^{\top_T}$). For y, the extensivity of $e_{3.2}$ is from y^0 in assumption to y^+ in result, hence $e_{3.2}$ is extensive for y.

Example 3.3
As an expression where different variables are bound to the same value, consider $e_{3.3} = (\lambda y.(\lambda f.f\ y)(\lambda x.y))\ z$. Its extensivity under assumption $z : \{z^0\}$ is $\{z^0\}$. That is, the extensivity of $e_{3.3}$ for z is from $\{z^0\}$ to $\{z^0\}$, hence $e_{3.3}$ is extensive for z:

$$\dfrac{z : \{z^0\} \vdash z : \{z^0\} \qquad \dfrac{\dfrac{A}{z : \{z^0\}, y : \{z^0\} \vdash (\lambda f.f\ y)(\lambda x.y) : \{z^0\}}\text{ (APP)}}{z : \{z^0\} \vdash (\lambda y.(\lambda f.f\ y)(\lambda x.y)) : \{z^0\} \to \{z^0\}}\text{ (LAM)}}{z : \{z^0\} \vdash (\lambda y.(\lambda f.f\ y)(\lambda x.y))\ z : \{z^0\}}\text{ (APP)}$$

where sub-proof-trees in A are

$$\dfrac{\dfrac{\vdots}{z : \{z^0\}, y : \{z^0\}, f : \{z^0\} \to \{z^0\} \vdash f\ y : \{z^0\}}\text{ (APP)}}{z : \{z^0\}, y : \{z^0\} \vdash \lambda f.f\ y : (\{z^0\} \to \{z^0\}) \to \{z^0\}}\text{ (LAM)}$$

and

$$\dfrac{z : \{z^0\}, y : \{z^0\}, x : \{z^0\} \vdash y : \{z^0\}}{z : \{z^0\}, y : \{z^0\} \vdash \lambda x.y : \{z^0\} \to \{z^0\}}\text{ (LAM)}$$

Example 3.4
As an example that a function is called at different site, consider $e_{3.4} = (\lambda f.(\lambda z.fy)(fx))(\lambda w_L.w)$. Its extensivity under assumption $x : \{x^0\}$ and $y : \{y^0\}$ is $\top_{X_L}$, hence the ex-

tensivity of $e_{3.4}$ for x and y are unknown, even though in reality it is extensive for y. This accuracy loss is because our analysis is mono-variant. The smallest functional extensivity for f that allows both "$f\ x$" ($\{x^0\} \to \{x^0\}$) and "$f\ y$" ($\{y^0\} \to \{y^0\}$) is found $\top_{X_L} \to \top_{X_L}$ by the (LAM) rule.

Example 3.5

As an example of the refinement, consider $e_{3.5} = \mathit{if}\ \top_L \sqsubseteq_L x\ ?\ y\ :\ x$ under $\Delta = \{y \sqsubseteq_L x\}$ and $\Gamma = \{x : \{x^0\}, y : \{y^0\}\}$. The extensivity of $e_{3.5}$ is $\{x^-, y^+\}$, hence reductive for x and extensive for y:

$$\frac{\Delta, \Gamma \vdash \top_L : \{x^+, y^+\} \quad \Delta, \Gamma \vdash x : \{x^0\} \quad A \quad B}{\Delta, \Gamma \vdash \mathit{if}\ \top_L \sqsubseteq_L x\ ?\ y\ :\ x : \{x^-, y^+\}}\ \text{(IF)}$$

where the sub-proof-tree in A is

$$\frac{y \sqsubseteq_L x \in \Delta \cup \Delta' \quad \Delta \cup \Delta', \Gamma \vdash y : \{y^0\} \quad \{y^0\} \sqsubseteq_{XB} \{y^-\}}{\Delta \cup \Delta', \Gamma \vdash y : \{x^-, y^0\}}\ \text{(REF}-\text{)},$$

$\Delta' = \{y \sqsubseteq_L x\}$ because $\{x^+, y^+\} \sqsubseteq_{XB} \{y^+\}$ and $\{x^0\} \sqsubseteq_{XB} \{x^-\}$, and the sub-proof-tree in B is

$$\frac{y \sqsubseteq_L x \in \Delta \quad \Delta, \Gamma \vdash x : \{x^0\} \quad \{x^0\} \sqsubseteq_{XB} \{x^+\}}{\Delta, \Gamma \vdash x : \{x^0, y^+\}}\ \text{(REF+)}.$$

However, without refinement, the extensivity of $e_{3.5}$ is $\{x^{\top_T}, y^{\top_T}\}$ because $\{y^0\} \sqcup_{XB} \{x^0\} = \{x^{\top_T}, y^{\top_T}\}$:

$$\frac{\cdots \quad \Delta \cup \Delta', \Gamma \vdash y : \{y^0\} \quad \Delta, \Gamma \vdash x : \{x^0\}}{\Delta, \Gamma \vdash \mathit{if}\ \top_L \sqsubseteq_L x\ ?\ y\ :\ x : \{x^{\top_T}, y^{\top_T}\}}\ \text{(IF)}$$

§4 Soundness

Definition 4.1

Let $(s, v) \models \mathsf{xb}$ be the minimal relation between extensivities xb, values v (lattice elements or functions), and value environments $s \in \mathit{Var} \xrightarrow{\text{fin}} \mathit{Value}$ that satisfies:

$$
\begin{array}{rcll}
(s,v) & \models & \mathsf{x}_L & \textit{iff} \quad \forall x \in \mathit{Var}_L : (s,v) \models x^{\mathsf{x}_L(x)} \\
(s,v) & \models & x^{+} & \textit{iff} \quad x \in \mathit{dom}(s) \ \wedge \ s(x) \sqsubseteq_L v \\
(s,v) & \models & x^{-} & \textit{iff} \quad x \in \mathit{dom}(s) \ \wedge \ v \sqsubseteq_L s(x) \\
(s,v) & \models & x^{0} & \textit{iff} \quad x \in \mathit{dom}(s) \ \wedge \ v = s(x) \\
(s,v) & \models & x^{\top_T} & \textit{iff} \quad \mathit{true} \\
(s,\lambda x.e) & \models & \mathsf{xb}_1 \rightarrow \mathsf{xb}_2 & \textit{iff} \quad \forall v_1 \in \mathit{Value} : ((s,v_1) \models \mathsf{xb}_1) \ \wedge \ (\{v_1/x\}e \hookrightarrow v) \\
 & & & \qquad \text{imply } (s + x : v_1, v) \models \mathsf{xb}_2 \\
(s,Yv) & \models & \mathsf{xb} & \textit{iff} \quad (s,v) \models \mathsf{xb} \rightarrow \mathsf{xb}
\end{array}
$$

We write $s \models \Gamma$ when the value environment s is consistent with the extensivity assumption Γ:

Definition 4.2
$s \models \Gamma$ iff $\mathit{dom}(\Gamma) = \mathit{dom}(s)$ and $\forall x \in \mathit{dom}(\Gamma) : (s, s(x)) \models \Gamma(x)$.

We write $s \models \Delta$ when the value environment s is consistent with the order assumption Δ:

Definition 4.3
$s \models \Delta$ iff $\forall x, y \in \mathit{Var}_L : x \sqsubseteq_L y \in \Delta \Rightarrow (x, y \in \mathit{dom}(s) \ \wedge \ s(x) \sqsubseteq_L s(y))$.

Lemma 4.1
If $(s,v) \models \mathsf{xb}$ and $\mathsf{xb} \sqsubseteq_{XB} \mathsf{xb}'$ then $(s,v) \models \mathsf{xb}'$.

Proof Obvious from the definition of $\models$ and the partial order $\sqsubseteq_{XB}$ in XB. ■

Lemma 4.2
If $(s,v) \models \mathsf{xb}_\tau$ then $\forall x \notin \mathit{dom}(s), \forall v' \in \mathit{Value} : (s + x : v', v) \models \mathsf{xb}_\tau$

Proof By the definition of $(s,v) \models \mathsf{xb}$. ■

Lemma 4.3
If $(s,v) \models \mathsf{xb}_1$ and $(s,v) \models \mathsf{xb}_2$ then $(s,v) \models \mathsf{xb}_1 \sqcap_{XB} \mathsf{xb}_2$.

Proof By the definition of $(s,v) \models \mathsf{xb}$. ■

Lemma 4.4
If $\Delta, \Gamma \vdash e : \mathsf{xb}$ and $\Delta \subseteq \Delta'$ then $\Delta', \Gamma \vdash e : \mathsf{xb}$

Proof We prove the lemma by induction on the proof tree size. For (REF$-$)

and (REF+), if $x \sqsubseteq_L y \in \Delta$ then $x \sqsubseteq_L y \in \Delta'$, and thus the lemma holds by induction. For other rules, induction is trivial because they do not use Δ. ■

Finally we are ready to state the correctness result:

Theorem 4.1
If $\Delta, \Gamma \vdash e : \mathsf{xb}$ then $s \models \Gamma$, $s \models \Delta$, and $\{s\}e \hookrightarrow v$ imply $(s, v) \models \mathsf{xb}$.

Proof We proceed by induction on the proof tree size.

Case (VAR) $\Delta, \Gamma \vdash x : \Gamma(x)$.
By definition, $(s, \{s\}x) \models \Gamma(x)$ for $s \models \Gamma$ because $\{s\}x = s(x)$.

Case (CON-1) $\Delta, \Gamma \vdash \bot_L : \{x \mapsto - \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$ or (CON-2) $\Delta, \Gamma \vdash \top_L : \{x \mapsto + \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$.
Obviously, $\bot_L \sqsubseteq_L s(x)$ and $s(x) \sqsubseteq_L \top_L$ for $x \in \mathit{dom}(s) = \mathit{dom}(\Gamma)$.

Case (CON-3) $\Delta, \Gamma \vdash c_L : \{x \mapsto \top_T \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$.
Obviously, $(s, c) \models x^{\top_T}$ for $x \in \mathit{dom}(s) = \mathit{dom}(\Gamma)$.

Case (LAM) $\Delta, \Gamma \vdash \lambda x.e : \mathsf{xb}_1 \to \mathsf{xb}_2$.
By definition, $\Delta, \Gamma + x : \mathsf{xb}_1 \vdash e : \mathsf{xb}'_2$ and $\mathsf{xb}'_2 \sqsubseteq_{XB} \mathsf{xb}_2$. Let $s \models \Gamma$ and $s \models \Delta$. By definition $\{s\}\lambda x.e = \lambda x.\{s\}e \hookrightarrow \lambda x.\{s\}e$, because $x \notin \mathit{dom}(s)$. We must show: $(s, \lambda x.\{s\}e) \models \mathsf{xb}_1 \to \mathsf{xb}_2$, that is, $(s, v_1) \models \mathsf{xb}_1$ and $\{v_1/x\}\{s\}e \hookrightarrow v_2$ imply $(s, v_2) \models \mathsf{xb}_2$. From $s \models \Gamma$ and $(s, v_1) \models \mathsf{xb}_1$, $s + x : v_1 \models \Gamma + x : \mathsf{xb}_1$. $s + x : v_1 \models \Delta$ because $s \models \Delta$ and $x \notin \mathit{dom}(s)$. $\{v_1/x\}\{s\}e \hookrightarrow v_2$ is $\{s + x : v_1\}e \hookrightarrow v_2$. Hence, by induction hypothesis, $(s + x : v_1, v_2) \models \mathsf{xb}'_2$, which implies $(s + x : v_1, v_2) \models \mathsf{xb}_2$ (by Lemma 4.1).

Case (REC) $\Delta, \Gamma \vdash Y \lambda x.e : \mathsf{xb}$.
We have to show $(s, Y\lambda x.\{s\}e) \models \mathsf{xb}$ for $s \models \Gamma$. By definition, $\Delta, \Gamma \vdash \lambda x.e : \mathsf{xb} \to \mathsf{xb}$. By induction hypothesis, $(s, \lambda x.\{s\}e) \models \mathsf{xb} \to \mathsf{xb}$, which, by definition, implies $(s, Y\lambda x.\{s\}e) \models \mathsf{xb}$.

Case (APP) $\Delta, \Gamma \vdash e_1\, e_2 : \mathsf{xb}_2$.
By definition, $\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1 \to \mathsf{xb}_2$, $\Delta, \Gamma \vdash e_2 : \mathsf{xb}'_1$, and $\mathsf{xb}'_1 \sqsubseteq_{XB} \mathsf{xb}_1$. Let $s \models \Gamma$, $s \models \Delta$ and $(\{s\}e_1)\, (\{s\}e_2) \hookrightarrow v$. We have to show: $(s + x : v', v) \models \mathsf{xb}_2$ where $(s, v') \models \mathsf{xb}_1$ and $\{v/x\}e \hookrightarrow v$. There are two sub-cases where e_1 evaluates to a non-recursive function or a recursive function.

- When $\{s\}e_1$ evaluates to a non-recursive function, $(\{s\}e_1)\, (\{s\}e_2) \hookrightarrow v$

means

$$\{s\}e_1 \hookrightarrow \lambda x.e, \tag{1}$$
$$\{s\}e_2 \hookrightarrow v', \text{ and} \tag{2}$$
$$\{v'/x\}e \hookrightarrow v. \tag{3}$$

By induction hypothesis, respectively from (1) and (2),

$$(s, \lambda x.e) \models \mathsf{xb}_1 \to \mathsf{xb}_2 \tag{4}$$

and $(s, v') \models \mathsf{xb}_1'$ which by Lemma 4.1 is

$$(s, v') \models \mathsf{xb}_1. \tag{5}$$

Hence from (3), (4), and (5), $(s, v) \models \mathsf{xb}_2$ and in turn $(s + x : v', v) \models \mathsf{xb}_2$ (by Lemma 4.2).

- When e_1 evaluates to a recursive function, $(\{s\}e_1)\ (\{s\}e_2) \hookrightarrow v$ means

$$\{s\}e_1 \hookrightarrow Yv, \tag{6}$$
$$\{s\}e_2 \hookrightarrow v', \tag{7}$$
$$v(Yv) \hookrightarrow \lambda x.e, \text{ and} \tag{8}$$
$$\{v'/x\}e \hookrightarrow v. \tag{9}$$

By induction hypothesis, respectively from (6) and (7),

$$(s, Yv) \models \mathsf{xb}_1 \to \mathsf{xb}_2 \tag{10}$$

and $(s, v') \models \mathsf{xb}_1'$ which by Lemma 4.1 implies

$$(s, v') \models \mathsf{xb}_1. \tag{11}$$

By definition from (10), $(s, v) \models (\mathsf{xb}_1 \to \mathsf{xb}_2) \to (\mathsf{xb}_1 \to \mathsf{xb}_2)$. Thus from (10) and (8), $(s, \lambda x.e) \models \mathsf{xb}_1 \to \mathsf{xb}_2$, which from (11) and (9) implies $(s, v) \models \mathsf{xb}_2$ and in turn $(s + x : v', v) \models \mathsf{xb}_2$ (by Lemma 4.2).

Case (LUB) $\Delta, \Gamma \vdash e_1 \sqcup_L e_2 : \mathsf{xb}$.

By definition, $\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1$, $\Delta, \Gamma \vdash e_2 : \mathsf{xb}_2$, and $\mathsf{xb} = \mathsf{xb}_1 \oplus \mathsf{xb}_2$. Let $s \models \Gamma$, $s \models \Delta$, and $(\{s\}e_1) \sqcup_L (\{s\}e_2) \hookrightarrow v$. We have to show: $(s, v) \models \mathsf{xb}$. $(\{s\}e_1) \sqcup_L (\{s\}e_2) \hookrightarrow v$ means $\{s\}e_1 \hookrightarrow v_1$, $\{s\}e_2 \hookrightarrow v_2$, and $v = v_1 \sqcup_L v_2$. By induction hypothesis, $(s, v_1) \models \mathsf{xb}_1$, and $(s, v_2) \models \mathsf{xb}_2$. Because $v = v_1 \sqcup_L v_2$, $\mathsf{xb} = \mathsf{xb}_1 \oplus \mathsf{xb}_2$, and $\oplus$ is point-wise, it is sufficient to show: $\forall x \in \mathit{Var}_L \cap \mathit{dom}(s)$: $(s, v_1 \sqcup_L v_2) \models x^{\mathsf{xb}_1(x) \oplus \mathsf{xb}_2(x)}$. Note that if $x \notin \mathit{dom}(s)$ then $(s, v_1 \sqcup_L v_2) \models x^{\mathsf{xb}_1(x) \oplus \mathsf{xb}_2(x)}$ is always true because $\mathsf{xb}_1(x) \oplus \mathsf{xb}_2(x) = \top_T \oplus \top_T = \top_T$.

- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^0$, $v_1 = s(x)$ and $v_2 = s(x)$. Hence, $v_1 \sqcup_L v_2 = s(x)$, and it implies $(s, v_1 \sqcup_L v_2) \models x^0$ $(= x^{0 \oplus 0})$.
- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^+$, $v_1 = s(x)$ and $s(x) \sqsubseteq_L v_2$. Hence, $s(x) \sqsubseteq_L v_1 \sqcup_L v_2$, and it implies $(s, v_1 \sqcup_L v_2) \models x^+$ $(= x^{0 \oplus +})$.
- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^-$, $v_1 = s(x)$ and $v_2 \sqsubseteq_L s(x)$. Hence, $v_1 \sqcup_L v_2 = s(x)$, and it implies $(s, v_1 \sqcup_L v_2) \models x^0$ $(= x^{0 \oplus -})$.
- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^{\top_T}$, $v_1 = s(x)$ and v_2 can be any value. Hence, $v_1 \sqcup_L s(x) \sqsubseteq_L v_2$, and it implies $(s, v_1 \sqcup_L v_2) \models x^+$ $(= x^{0 \oplus \top_T})$.
- When $(s, v_1) \models x^+$ and $(s, v_2) \models x^+$, $s(x) \sqsubseteq_L v_1$ and $s(x) \sqsubseteq_L v_2$. Hence, $s(x) \sqsubseteq_L v_1 \sqcup_L v_2$, and it implies $(s, v_1 \sqcup_L v_2) \models x^+$ $(= x^{+ \oplus +})$.
- When $(s, v_1) \models x^+$ and $(s, v_2) \models x^-$, $s(x) \sqsubseteq_L v_1$ and $s(x) \sqsubseteq_L v_2$. Hence, $s(x) \sqsubseteq_L v_1 \sqcup_L v_2$, and it implies $(s, v_1 \sqcup_L v_2) \models x^+$ $(= x^{+ \oplus -})$.
- When $(s, v_1) \models x^+$ and $(s, v_2) \models x^{\top_T}$, $s(x) \sqsubseteq_L v_1$ and v_2 can be any value. Hence, $s(x) \sqsubseteq_L v_1 \sqcup_L v_2$, and it implies $(s, v_1 \sqcup_L v_2) \models x^+$ $(= x^{+ \oplus \top_T})$.
- When $(s, v_1) \models x^-$ and $(s, v_2) \models x^-$, $v_1 \sqsubseteq_L s(x)$ and $v_2 \sqsubseteq_L s(x)$. Hence, $v_1 \sqcup_L v_2 \sqsubseteq_L s(x)$, and it implies $(s, v_1 \sqcup_L v_2) \models x^-$ $(= x^{- \oplus -})$.
- When $(s, v_1) \models x^-$ and $(s, v_2) \models x^{\top_T}$, $v_1 \sqsubseteq_L s(x)$ and v_2 can be any value. Hence, $v_1 \sqcup_L v_2$ can be any value, and it implies $(s, v_1 \sqcup_L v_2) \models x^{\top_T}$ $(= x^{- \oplus \top_T})$.
- When $(s, v_1) \models x^{\top_T}$ and $(s, v_2) \models x^{\top_T}$, v_1 and v_2 can be any value. Hence, $v_1 \sqcup_L v_2$ can be any value, and it implies $(s, v_1 \sqcup_L v_2) \models x^{\top_T}$ $(= x^{\top_T \oplus \top_T})$.

Case (GLB) $\Delta, \Gamma \vdash e_1 \sqcap_L e_2 : \mathsf{xb}$.

By definition, $\Delta, \Gamma \vdash e_1 : \mathsf{xb}_1$, $\Delta, \Gamma \vdash e_2 : \mathsf{xb}_2$, and $\mathsf{xb} = \mathsf{xb}_1 \ominus \mathsf{xb}_2$. Let $s \models \Gamma$ and $(\{s\}e_1) \sqcap_L (\{s\}e_2) \hookrightarrow v$. We have to show: $(s, v) \models \mathsf{xb}$. $(\{s\}e_1) \sqcap_L (\{s\}e_2) \hookrightarrow v$ means $\{s\}e_1 \hookrightarrow v_1$, $\{s\}e_2 \hookrightarrow v_2$, and $v = v_1 \sqcap_L v_2$. By induction hypothesis, $(s, v_1) \models \mathsf{xb}_1$, and $(s, v_2) \models \mathsf{xb}_2$. Because $v = v_1 \sqcap_L v_2$, $\mathsf{xb} = \mathsf{xb}_1 \ominus \mathsf{xb}_2$, and $\ominus$ is point-wise, it is sufficient to show: $\forall x \in Var_L \cap dom(s) : (s, v_1 \sqcap_L v_2) \models x^{\mathsf{xb}_1(x) \ominus \mathsf{xb}_2(x)}$.

- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^0$, $v_1 = s(x)$ and $v_2 = s(x)$. Hence, $v_1 \sqcap_L v_2 = s(x)$, and it implies $(s, v_1 \sqcap_L v_2) \models x^0$ $(= x^{0 \ominus 0})$.
- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^+$, $v_1 = s(x)$ and $s(x) \sqsubseteq_L v_2$. Hence, $v_1 \sqcap_L v_2 = s(x)$, and it implies $(s, v_1 \sqcap_L v_2) \models x^0$ $(= x^{0 \ominus +})$.
- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^-$, $v_1 = s(x)$ and $v_2 \sqsubseteq_L s(x)$. Hence, $v_1 \sqcap_L v_2 \sqsubseteq_L s(x)$, and it implies $(s, v_1 \sqcap_L v_2) \models x^-$ $(= x^{0 \ominus -})$.
- When $(s, v_1) \models x^0$ and $(s, v_2) \models x^{\top_T}$, $v_1 = s(x)$ and v_2 can be any value.

Hence, $v_1 \sqcap_L v_2 \sqsubseteq_L s(x)$, and it implies $(s, v_1 \sqcap_L v_2) \models x^-$ $(= x^{0 \ominus \top_T})$.

- When $(s, v_1) \models x^+$ and $(s, v_2) \models x^+$, $s(x) \sqsubseteq_L v_1$ and $s(x) \sqsubseteq_L v_2$. Hence, $s(x) \sqsubseteq_L v_1 \sqcap_L v_2$, and it implies $(s, v_1 \sqcap_L v_2) \models x^+$ $(= x^{+\ominus+})$.
- When $(s, v_1) \models x^+$ and $(s, v_2) \models x^-$, $s(x) \sqsubseteq_L v_1$ and $s(x) \sqsubseteq_L v_2$. Hence, $s(x) \sqsubseteq_L v_1 \sqcap_L v_2$, and it implies $(s, v_1 \sqcap_L v_2) \models x^-$ $(= x^{+\ominus-})$.
- When $(s, v_1) \models x^+$ and $(s, v_2) \models x^{\top_T}$, $s(x) \sqsubseteq_L v_1$ and v_2 can be any value. Hence, $v_1 \sqcap_L v_2$ can be any value, and it implies $(s, v_1 \sqcap_L v_2) \models x^{\top_T}$ $(= x^{+\ominus\top_T})$.
- When $(s, v_1) \models x^-$ and $(s, v_2) \models x^-$, $v_1 \sqsubseteq_L s(x)$ and $v_2 \sqsubseteq_L s(x)$. Hence, $v_1 \sqcap_L v_2 \sqsubseteq_L s(x)$, and it implies $(s, v_1 \sqcap_L v_2) \models x^-$ $(= x^{-\ominus-})$.
- When $(s, v_1) \models x^-$ and $(s, v_2) \models x^{\top_T}$, $v_1 \sqsubseteq_L s(x)$ and v_2 can be any value. Hence, $v_1 \sqcap_L v_2 \sqsubseteq_L s(x)$, and it implies $(s, v_1 \sqcap_L v_2) \models x^-$ $(= x^{-\ominus\top_T})$.
- When $(s, v_1) \models x^{\top_T}$ and $(s, v_2) \models x^{\top_T}$, v_1 and v_2 can be any value. Hence, $v_1 \sqcap_L v_2$ can be any value, and it implies $(s, v_1 \sqcap_L v_2) \models x^{\top_T}$ $(= x^{\top_T \ominus \top_T})$.

Case (IF) $\Delta, \Gamma \vdash \mathit{if}\ e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4 : \mathsf{xb}$.

By definition $\forall i \in \{1,2,4\} : \Delta, \Gamma \vdash e_i : \mathsf{xb}_i$, $\Delta \cup \Delta', \Gamma \vdash e_3 : \mathsf{xb}_3$ where $\Delta' = \{x \sqsubseteq_L y \mid \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\}, \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\}\}$, and $\mathsf{xb} = \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4$. Let $s \models \Gamma$, $s \models \Delta$ and $\mathit{if}\,\{s\}e_1 \sqsubseteq_L \{s\}e_2 ? \{s\}e_3 : \{s\}e_4 \hookrightarrow v$. We have to show: $(s, v) \models \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4$.

- When $\{s\}e_3 \hookrightarrow v$, by definition, $\{s\}e_1 \hookrightarrow v_1$, $\{s\}e_2 \hookrightarrow v_2$, and $v_1 \sqsubseteq_L v_2$. By induction hypothesis, $(s, v_1) \models \mathsf{xb}_1$, and $(s, v_2) \models \mathsf{xb}_2$. We first show $s \models \Delta \cup \Delta'$, i.e., $\forall x, y : x \sqsubseteq_L y \in \Delta \cup \Delta' \Rightarrow x, y \in \mathit{dom}(s) \wedge s(x) \sqsubseteq_L s(y)$.
  - If $x \sqsubseteq_L y \in \Delta$, then $x, y \in \mathit{dom}(s)$ and $s(x) \sqsubseteq_L s(y)$ because $s \models \Delta$.
  - If $x \sqsubseteq_L y \in \Delta'$, then $\mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\}$ and $\mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\}$. $\mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\}$ and $(s, v_1) \models \mathsf{xb}_1$ imply $x \in \mathit{dom}(s)$ and $s(x) \sqsubseteq_L v_1$. $\mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\}$ and $(s, v_2) \models \mathsf{xb}_2$ imply $y \in \mathit{dom}(s)$ and $v_2 \sqsubseteq_L s(y)$. $s(x) \sqsubseteq_L v_1$, $v_1 \sqsubseteq_L v_2$, and $v_2 \sqsubseteq_L s(y)$ imply $s(x) \sqsubseteq_L s(y)$

  Hence $s \models \Delta \cup \Delta'$. From $\Delta \cup \Delta', \Gamma \vdash e_3 : \mathsf{xb}_3$, $s \models \Gamma$, $s \models \Delta \cup \Delta'$, and $\{s\}e_3 \hookrightarrow v$, by induction hypothesis, $(s, v) \models \mathsf{xb}_3$. By Lemma 4.1, $(s, v) \models \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4$.
- When $\{s\}e_4 \hookrightarrow v$, by induction hypothesis, $(s, v) \models \mathsf{xb}_4$. By Lemma 4.1, $(s, v) \models \mathsf{xb}_3 \sqcup_{XB} \mathsf{xb}_4$.

Case (REF$-$) $\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{y^-\}$.

By definition $x \sqsubseteq_L y \in \Delta$, $\Delta, \Gamma \vdash e : \mathsf{xb}$, and $\mathsf{xb} \sqsubseteq_{XB} \{x^-\}$. Let $s \models \Gamma$, $s \models \Delta$, and $\{s\}e \hookrightarrow v$. By induction hypothesis, $(s, v) \models \mathsf{xb}$.

$x \sqsubseteq_L y \in \Delta$ and $s \models \Delta$	$\Rightarrow$	$s(x) \sqsubseteq_L s(y)$	by Definition 4.3
$\mathsf{xb} \sqsubseteq_{XB} \{x^-\}$ and $(s,v) \models \mathsf{xb}$	$\Rightarrow$	$v \sqsubseteq_L s(x)$	by Definition 4.1
$s(x) \sqsubseteq_L s(y)$ and $v \sqsubseteq_L s(x)$	$\Rightarrow$	$v \sqsubseteq_L s(y)$	by transitivity
$v \sqsubseteq_L s(y)$	$\Rightarrow$	$(s,v) \models y^-$	by Definition 4.1
$(s,v) \models y^-$	$\Rightarrow$	$(s,v) \models \{y^-\}$	by Lemma 4.1
$(s,v) \models \mathsf{xb}$ and $(s,v) \models \{y^-\}$	$\Rightarrow$	$(s,v) \models \mathsf{xb} \sqcap_{XB} \{y^-\}$	by Lemma 4.3.

Case (REF+) $\Delta, \Gamma \vdash e : \mathsf{xb} \sqcap_{XB} \{x^+\}$.
By definition $x \sqsubseteq_L y \in \Delta$, $\Delta, \Gamma \vdash e : \mathsf{xb}$, and $\mathsf{xb} \sqsubseteq_{XB} \{y^+\}$. Let $s \models \Gamma$, $s \models \Delta$, and $\{s\}e \hookrightarrow v$. By induction hypothesis, $(s,v) \models \mathsf{xb}$.

$x \sqsubseteq_L y \in \Delta$ and $s \models \Delta$	$\Rightarrow$	$s(x) \sqsubseteq_L s(y)$	by Definition 4.3
$\mathsf{xb} \sqsubseteq_{XB} \{y^+\}$ and $(s,v) \models \mathsf{xb}$	$\Rightarrow$	$s(y) \sqsubseteq_L v$	by Definition 4.1
$s(x) \sqsubseteq_L s(y)$ and $s(y) \sqsubseteq_L v$	$\Rightarrow$	$s(x) \sqsubseteq_L v$	by transitivity
$s(x) \sqsubseteq_L v$	$\Rightarrow$	$(s,v) \models x^+$	by Definition 4.1
$(s,v) \models x^+$	$\Rightarrow$	$(s,v) \models \{x^+\}$	by Lemma 4.1
$(s,v) \models \mathsf{xb}$ and $(s,v) \models \{x^+\}$	$\Rightarrow$	$(s,v) \models \mathsf{xb} \sqcap_{XB} \{x^+\}$	by Lemma 4.3.

■

§5 Algorithm

Our inference system is a variant of type inference system with inclusion constraints[1]. Our algorithm consists of three phases: we derive constraints for the extensivity behaviors, simplify the constraints, and then solve the equations derived from simplified constraints. The conventional fixpoint iteration can be applied to the simplified constraints since every operator ($\oplus, \ominus, f[t/x], \pi$, and $\sqcup_{XB}$) is monotonic. Because the least model for the constraints is equivalent to the least fixpoint of the corresponding equations[5], the algorithm will give the best approximation of extensivity that could be inferred in our inference system.

In our inference system, we assume that there is no dead code in a program, i.e., every function in a program must be called at least once. If there is a function which is never called, we can not dissolve the function's constraint because we don't have any constraint for its argument. A constraint for a function's argument is derived from its call-sites.

5.1 Extraction of Constraints

Extracted constraint ρ is of the following form:

$$\begin{array}{rrcl} \text{constraint} & \rho & ::= & L \supseteq R \mid \exists\alpha.\rho \mid \rho,\rho \\ \text{lhs term} & L & ::= & \alpha \mid L \to L \\ \text{rhs term} & R & ::= & L \mid \alpha \oplus \alpha \mid \alpha \ominus \alpha \mid \pi(\alpha,\alpha,\alpha) \mid \alpha \sqcup_{XB} \alpha \mid \mathsf{x}_L \end{array}$$

where α is a variable for the extensivity behavior, x_L is an extensivity behavior of lattice L, and refinement operator π is:

$$\pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3) = \mathit{gfp}\ \lambda\alpha.\mathsf{xb}_3 \sqcap (\Pi_{\mathsf{xb}_1,\mathsf{xb}_2}\alpha),$$

where $\Pi_{\mathsf{xb}_1,\mathsf{xb}_2}\alpha =$

$$\begin{cases} \alpha \sqcap_{XB} \{x^+\}, & \text{if } \exists y.\ \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\} \ \wedge\ \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\} \ \wedge\ \alpha \sqsubseteq_{XB} \{y^+\} \\ \alpha \sqcap_{XB} \{y^-\}, & \text{if } \exists x.\ \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\} \ \wedge\ \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\} \ \wedge\ \alpha \sqsubseteq_{XB} \{x^-\} \\ \alpha, & \text{otherwise.} \end{cases}$$

$\Pi_{\mathsf{xb}_1,\mathsf{xb}_2}$ refines an extensivity α with the order assumption generated from xb_1 and xb_2, i.e., Π is the combination of (REF$-$), (REF$+$), and generating order assumption Δ in (IF). Because we can apply (REF$-$) and (REF$+$) recursively on an expression and the refinement is indeed the greatest lower bound $\sqcap_{XB}$, the refinement operator $\pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3)$ should compute the greatest fixpoint $\mathit{gfp}\ \lambda\alpha.\mathsf{xb}_3 \sqcap (\Pi_{\mathsf{xb}_1,\mathsf{xb}_2}\alpha)$. Note that in the inference algorithm we only refine the extensivities of the true-branches of if-expressions, while in the checking system we can refine the extensivities of any expressions. This restriction is necessary to make our inference system deterministic.

The validity of the constraint ρ, written as "$\vdash \rho$" is defined as follows. $\{\mathsf{xb}/\alpha\}\rho$ denotes, as usual, the result from substituting xb for variable α in ρ.

$$\frac{\mathsf{xb}_1 \sqsupseteq_{XB} \mathsf{xb}_2}{\vdash \mathsf{xb}_1 \supseteq \mathsf{xb}_2} \quad \frac{\vdash \{\mathsf{xb}/\alpha\}\rho}{\vdash \exists\alpha.\rho} \quad \frac{\vdash \rho_1 \quad \vdash \rho_2}{\vdash \rho_1, \rho_2}$$

We extract the constraint from an expression e using a recursive procedure $C(\Gamma, e, \mathsf{xb})$ (Figure 3). It has a linear time complexity (with respect to the size of e). The size of the generated constraint is also linear in the size of e.

The constraint generation $C(\Gamma, e, \mathsf{xb})$ and its solution is a correct implementation of our deductive system:

Theorem 5.1
If $\vdash C(\Gamma, e, \mathsf{xb})$ then $\emptyset, \Gamma \vdash e : \mathsf{xb}$.

Proof We proceed by structural induction on e.

$$
\begin{array}{lcl}
C(\Gamma, \bot_L, \mathsf{xb}) & = & \mathsf{xb} \supseteq \{x \mapsto - \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\} \\
C(\Gamma, c_L, \mathsf{xb}) & = & \mathsf{xb} \supseteq \{x \mapsto \top_T \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\} \\
C(\Gamma, \top_L, \mathsf{xb}) & = & \mathsf{xb} \supseteq \{x \mapsto + \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\} \\
C(\Gamma, x, \mathsf{xb}) & = & \mathsf{xb} \supseteq \Gamma(x) \\
C(\Gamma, \lambda x.e, \mathsf{xb}) & = & \exists \alpha_1 \alpha_2 \alpha_2'. \\
& & C(\Gamma + x : \alpha_1, e, \alpha_2'), \\
& & \mathsf{xb} \supseteq \alpha_1 \to \alpha_2,\ \alpha_2 \supseteq \alpha_2' \\
C(\Gamma, Y \lambda x.e, \mathsf{xb}) & = & C(\Gamma, \lambda x.e, \mathsf{xb} \to \mathsf{xb}) \\
C(\Gamma, e_1\ e_2, \mathsf{xb}) & = & \exists \alpha_1 \alpha_1'. \\
& & C(\Gamma, e_1, \alpha_1 \to \mathsf{xb}), \quad C(\Gamma, e_2, \alpha_1'), \\
& & \alpha_1 \supseteq \alpha_1' \\
C(\Gamma, e_1 \sqcup_L e_2, \mathsf{xb}) & = & \exists \alpha_1 \alpha_2. \\
& & C(\Gamma, e_1, \alpha_1), \quad C(\Gamma, e_2, \alpha_2), \\
& & \mathsf{xb} \supseteq \alpha_1 \oplus \alpha_2 \\
C(\Gamma, e_1 \sqcap_L e_2, \mathsf{xb}) & = & \exists \alpha_1 \alpha_2. \\
& & C(\Gamma, e_1, \alpha_1), \quad C(\Gamma, e_2, \alpha_2), \\
& & \mathsf{xb} \supseteq \alpha_1 \ominus \alpha_2 \\
\multicolumn{3}{l}{C(\Gamma, \mathit{if}\ e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4, \mathsf{xb})} \\
& = & \exists \alpha_1 \alpha_2 \alpha_3 \alpha_4. \\
& & C(\Gamma, e_1, \alpha_1), C(\Gamma, e_2, \alpha_2), C(\Gamma, e_3, \alpha_3), C(\Gamma, e_4, \alpha_4), \\
& & \mathsf{xb} \supseteq \pi(\alpha_1, \alpha_2, \alpha_3) \sqcup_{XB} \alpha_4
\end{array}
$$

Fig. 3 Constraint Extraction Procedure

Case x.

Let$\vdash C(\Gamma, x, \mathsf{xb})$. It implies $\mathsf{xb} \sqsupseteq_{XB} \Gamma(x)$. By (VAR), $\emptyset, \Gamma \vdash x : \Gamma(x)$. By Lemma 4.1, $\emptyset, \Gamma \vdash x : \mathsf{xb}$.

Case $\bot_L$.

Let$\vdash C(\Gamma, \bot_L, \mathsf{xb})$. It implies $\mathsf{xb} \sqsupseteq_{XB} \{x \mapsto - \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$. Because $\emptyset, \Gamma \vdash \bot_L : \{x \mapsto - \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$ by (CON-1), $\emptyset, \Gamma \vdash \bot_L : \mathsf{xb}$.

Case $\top_L$.

Let$\vdash C(\Gamma, \top_L, \mathsf{xb})$. It implies $\mathsf{xb} \sqsupseteq_{XB} \{x \mapsto + \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$. Because $\emptyset, \Gamma \vdash \top_L : \{x \mapsto + \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$ by (CON-2), $\emptyset, \Gamma \vdash \top_L : \mathsf{xb}$.

Case for other constants c.

Let$\vdash C(\Gamma, c_L, \mathsf{xb})$. It implies $\mathsf{xb} \sqsupseteq_{XB} \{x \mapsto \top_T \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$. Because $\emptyset, \Gamma \vdash c_L : \{x \mapsto \top_T \mid x \in \mathit{Var}_L \cap \mathit{dom}(\Gamma)\}$ by (CON-3), $\emptyset, \Gamma \vdash c : \mathsf{xb}$.

Case $\lambda x.e$.

Let $\vdash C(\Gamma, \lambda x.e, \mathsf{xb})$. It implies that there are $\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_2'$ such that $\vdash C(\Gamma + x : \mathsf{xb}_1, e, \mathsf{xb}_2')$, $\mathsf{xb} \sqsupseteq_{XB} \mathsf{xb}_1 \to \mathsf{xb}_2$, and $\mathsf{xb}_2' \sqsubseteq_{XB} \mathsf{xb}_2$. By induction hypothesis, $\emptyset, \Gamma + x : \mathsf{xb}_1 \vdash e : \mathsf{xb}_2'$. By (LAM), $\emptyset, \Gamma \vdash \lambda x.e : \mathsf{xb}_1 \to \mathsf{xb}_2$. Because $\mathsf{xb} \sqsupseteq_{XB} \mathsf{xb}_1 \to \mathsf{xb}_2$, $\emptyset, \Gamma \vdash \lambda x.e : \mathsf{xb}$.

Case $Y\lambda x.e$.

Let $\vdash C(\Gamma, Y\lambda x.e, \mathsf{xb})$. It implies that $C(\Gamma, \lambda x.e, \mathsf{xb} \to \mathsf{xb})$. By induction hypothesis, $\emptyset, \Gamma \vdash \lambda x.e : \mathsf{xb} \to \mathsf{xb}$. By (REC), $\emptyset, \Gamma \vdash Y\lambda x.e : \mathsf{xb}$.

Case $e_1\, e_2$.

Let $\vdash C(\Gamma, e_1 e_2, \mathsf{xb})$. It implies $C(\Gamma, e_1, \mathsf{xb}_1 \to \mathsf{xb})$, $C(\Gamma, e_2, \mathsf{xb}_1')$, and $\mathsf{xb}_1' \sqsubseteq_{XB} \mathsf{xb}_1$. By induction hypothesis, $\emptyset, \Gamma \vdash e_1 : \mathsf{xb}_1 \to \mathsf{xb}$ and $\emptyset, \Gamma \vdash e_2 : \mathsf{xb}_1'$. By (APP), $\emptyset, \Gamma \vdash e_1\, e_2 : \mathsf{xb}$.

Case $e_1 \sqcup_L e_2$.

Let $\vdash C(\Gamma, e_1 \sqcup_L e_2, \mathsf{xb})$. It implies $C(\Gamma, e_1, \mathsf{xb}_1)$, $C(\Gamma, e_2, \mathsf{xb}_2)$, and $\mathsf{xb} \sqsupseteq_{XB} \mathsf{xb}_1 \oplus \mathsf{xb}_2$. By induction hypothesis, $\emptyset, \Gamma \vdash e_1 : \mathsf{xb}_1$ and $\emptyset, \Gamma \vdash e_2 : \mathsf{xb}_2$. By (LUB), $\emptyset, \Gamma \vdash e_1 \sqcup_L e_2 : \mathsf{xb}_1 \oplus \mathsf{xb}_2$. Because $\mathsf{xb} \sqsupseteq_{XB} \mathsf{xb}_1 \oplus \mathsf{xb}_2$, $\emptyset, \Gamma \vdash e_1 \sqcup_L e_2 : \mathsf{xb}$.

Case $e_1 \sqcap_L e_2$.

Let $\vdash C(\Gamma, e_1 \sqcap_L e_2, \mathsf{xb})$. It implies $C(\Gamma, e_1, \mathsf{xb}_1)$, $C(\Gamma, e_2, \mathsf{xb}_2)$, and $\mathsf{xb} \sqsupseteq_{XB} \mathsf{xb}_1 \ominus \mathsf{xb}_2$. By induction hypothesis, $\emptyset, \Gamma \vdash e_1 : \mathsf{xb}_1$ and $\emptyset, \Gamma \vdash e_2 : \mathsf{xb}_2$. By (GLB), $\emptyset, \Gamma \vdash e_1 \sqcap_L e_2 : \mathsf{xb}_1 \ominus \mathsf{xb}_2$. Because $\mathsf{xb} \sqsupseteq_{XB} \mathsf{xb}_1 \ominus \mathsf{xb}_2$, $\emptyset, \Gamma \vdash e_1 \sqcap_L e_2 : \mathsf{xb}$.

Case *if* $e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4$.

Let $\vdash C(\Gamma, \textit{if}\ e_1 \sqsubseteq_L e_2\ ?\ e_3\ :\ e_4, \mathsf{xb})$. It implies $C(\Gamma, e_1, \mathsf{xb}_1)$, $C(\Gamma, e_2, \mathsf{xb}_2)$, $C(\Gamma, e_3, \mathsf{xb}_3)$, $C(\Gamma, e_4, \mathsf{xb}_4)$, and $\mathsf{xb} \sqsupseteq_{XB} \pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3) \sqcup_{XB} \mathsf{xb}_4$. By induction hypothesis, $\emptyset, \Gamma \vdash e_1 : \mathsf{xb}_1$, $\emptyset, \Gamma \vdash e_2 : \mathsf{xb}_2$, $\emptyset, \Gamma \vdash e_3 : \mathsf{xb}_3$, and $\emptyset, \Gamma \vdash e_4 : \mathsf{xb}_4$. By Lemma 4.4, $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3$ where $\Delta = \{x \sqsubseteq_L y \mid \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\}, \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\}\}$. In order to show $\Delta, \Gamma \vdash e_3 : \pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3)$, it is sufficient to show that $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^i_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB})$ for all $i \geq 0$, because the height of XB is finite. We prove it by induction on i.

- $i = 0$.
  Because $\mathsf{xb}_3 \sqcap_{XB} (\Pi^0_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB}) = \mathsf{xb}_3$ and $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3$, $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^0_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB})$.
- $i > 0$.
  By induction hypothesis, $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB})$.
  - If $\exists x, y : \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\} \ \wedge\ \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\} \ \wedge\ \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB}) \sqsubseteq_{XB} \{y^+\}$, $x \sqsubseteq_L y \in \Delta$. By (REF+), $x \sqsubseteq_L y \in \Delta$, $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB})$, and $\mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1, \mathsf{xb}_2} \top_{XB}) \sqsubseteq_{XB} \{y^+\}$ imply

$\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB}) \sqcap_{XB} \{x^+\}$.

Hence $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB})$.

– If $\exists x, y : \mathsf{xb}_1 \sqsubseteq_{XB} \{x^+\} \wedge \mathsf{xb}_2 \sqsubseteq_{XB} \{y^-\} \wedge \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB}) \sqsubseteq_{XB} \{x^-\}$, $x \sqsubseteq_L y \in \Delta$. By (REF$-$), $x \sqsubseteq_L y \in \Delta$, $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB})$, and $\mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB}) \sqsubseteq_{XB} \{x^-\}$ imply

$\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB}) \sqcap_{XB} \{y^-\}$.

Hence $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap (\Pi^{i}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB})$.

– Otherwise, $\mathsf{xb}_3 \sqcap_{XB} (\Pi^{i}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB}) = \mathsf{xb}_3 \sqcap_{XB} (\mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB})) = \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i-1}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB})$. Hence $\Delta, \Gamma \vdash e_3 : \mathsf{xb}_3 \sqcap_{XB} (\Pi^{i}_{\mathsf{xb}_1,\mathsf{xb}_2} \top_{XB})$.

Therefore $\Delta, \Gamma \vdash e_3 : \pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3)$. From $\emptyset, \Gamma \vdash e_1 : \mathsf{xb}_1$, $\emptyset, \Gamma \vdash e_2 : \mathsf{xb}_2$, $\Delta, \Gamma \vdash e_3 : \pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3)$, and $\emptyset, \Gamma \vdash e_4 : \mathsf{xb}_4$, by (IF), $\emptyset, \Gamma \vdash \mathit{if}\, e_1 \sqsubseteq_L e_2 ? e_3 : e_4 : \pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3) \sqcup_{XB} \mathsf{xb}_4$. Because $\mathsf{xb} \sqsupseteq_{XB} \pi(\mathsf{xb}_1, \mathsf{xb}_2, \mathsf{xb}_3) \sqcup_{XB} \mathsf{xb}_4$, $\emptyset, \Gamma \vdash \mathit{if}\, e_1 \sqsubseteq_L e_2 ? e_3 : e_4 : \mathsf{xb}$. ■

5.2 Solving the constraints

Figure 4 shows simplification rules $\mathcal{S}$ for simplifying initial constraints. Rules consist of one transitivity rule and two elimination rules.

We write $A \vdash_{\mathcal{S}} c$ if a constraint c is derivable from A using rules $\mathcal{S}$, and write $\mathcal{S}^*(A)$ for the closure of A under rules $\mathcal{S}$, i.e., the set $\mathit{lfp}(\lambda X.A \cup \{c \mid X \vdash_{\mathcal{S}} c\})$.

An atomic constraint is $\alpha \supseteq at$ whose right-hand-side at (atomic term) explicitly denotes a simple expression of non-functional extensivity behavior x_L:

$$at \ ::= \ \mathsf{x}_L \ \mid \ \alpha \oplus \alpha \ \mid \ \alpha \ominus \alpha \ \mid \ \pi(\alpha, \alpha, \alpha)$$

Among the constraints $\mathcal{S}^*(A)$, completely dissolved constraints $(\mathit{atom}(\mathcal{S}^*(A)))$ constitute the least model of A, where $\mathit{atom}(C) = \{\alpha \supseteq at \in C\}$. We can prove this property easily, because our constraint system is a special case of a general setting[1].

The simplified atomic constraint constitutes one equation for each unknown; "$\alpha \supseteq at_1, \cdots, \alpha \supseteq at_n$" is equivalent to "$\alpha = at_1 \sqcup_{XB} \cdots \sqcup_{XB} at_n$". The least solution of the set of such equations corresponds to the least model of the simplified atomic constraints[5].

The least solution is computed by iteration: starting from bottom $\{x \mapsto$

$$\frac{L \supseteq \alpha \quad \alpha \supseteq R}{L \supseteq R} \text{ (TRANS)}$$

$$\frac{L_1 \to L_2 \supseteq L_3 \to L_4}{L_3 \supseteq L_1 \quad L_2 \supseteq L_4} \text{ ($\to$-ELIM)} \qquad \frac{L \supseteq \alpha_1 \sqcup_{XB} \alpha_2}{L \supseteq \alpha_1 \quad L \supseteq \alpha_2} \text{ ($\sqcup_{XB}$-ELIM)}$$

Fig. 4 Simplification Rules $\mathcal{S}$

$\perp_T \mid x \in \mathit{Var}_L\}$ for every α_L we iteratively apply the right-hand-sides of the equations to the intermediate results. This procedure terminates with the least fixpoint, because the involved operators $(\oplus, \ominus, \pi, \sqcup_{XB})$ are all monotonic[*1].

Example 5.1
For $e_{5.1} = (\lambda x.(x \sqcap_L c_L) \sqcup_L y)\ z$, let Γ consist of $y : \{y^0\}$ and $z : \{z^0\}$. The constraint generation procedure $C(\Gamma, (\lambda x.(x \sqcap_L c_L) \sqcup_L y)\ z, \alpha)$ returns the following constraint:

$$\begin{array}{lll} \alpha_1 \supseteq \alpha_2 & \alpha_2 \supseteq z^0 & \alpha_1 \to \alpha \supseteq \alpha_3 \to \alpha_4 \\ \alpha_4 \supseteq \alpha_5 & \alpha_5 \supseteq \alpha_6 \oplus \alpha_7 & \alpha_6 \supseteq \alpha_8 \ominus \alpha_9 \\ \alpha_7 \supseteq y^0 & \alpha_8 \supseteq \alpha_3 & \alpha_9 \supseteq \top_{X_L} \end{array}$$

Rules $\mathcal{S}$ simplifies initial constraints into the simplified atomic constraints:

$$\begin{array}{lll} \alpha \supseteq \alpha_6 \oplus \alpha_7 & \alpha_1 \supseteq z^0 & \alpha_2 \supseteq z^0 \\ \alpha_3 \supseteq z^0 & \alpha_4 \supseteq \alpha_6 \oplus \alpha_7 & \alpha_5 \supseteq \alpha_6 \oplus \alpha_7 \\ \alpha_6 \supseteq \alpha_8 \ominus \alpha_9 & \alpha_7 \supseteq y^0 & \alpha_8 \supseteq z^0 \\ \alpha_9 \supseteq \top_{X_L} & & \end{array}$$

The simplified atomic constraints constitute the equations:

$$\begin{array}{lll} \alpha = \alpha_6 \oplus \alpha_7 & \alpha_1 = z^0 & \alpha_2 = z^0 \\ \alpha_3 = z^0 & \alpha_4 = \alpha_6 \oplus \alpha_7 & \alpha_5 = \alpha_6 \oplus \alpha_7 \\ \alpha_6 = \alpha_8 \ominus \alpha_9 & \alpha_7 = y^0 & \alpha_8 = z^0 \\ \alpha_9 = \top_{X_L} & & \end{array}$$

The least solution for α is computed in 4 iterations, and corresponds to the result of $(x^0 \ominus \top_{X_L}) \oplus y^0$ which is $\{x^{\top_T}, y^+\}$.

[*1] $\oplus$ and $\ominus$ are monotonic by definition; we can check it case by case. π is monotonic because *gfp* and Π are monotonic.

Theorem 5.2
Let n be the size (the number of sub-expressions) of an input analysis specification, r be its largest type size[*2], and w be the number of variables in the analysis specification. The time complexity of our inference system is $O(n^{2r}) + O(n^2 \times w^3 \times r^3)$.

Proof The constraint extraction procedure C takes $O(n)$ time to generate $O(n)$ number of constraints. The simplification procedure ($\mathcal{S}^*$) takes a time proportional to the number of simplification rules that can be applied. Because the number of possible constraints ($L \supseteq R$) is $n^r \times n^r$ (n^r terms for L and R respectively), and because the simplification rules introduce new constraints that always consist of sub-terms of existing constraints, the simplification procedure reaches the closure within $O(n^{2r})$ time. After the simplification, the number of equations from the atomic constraints is n. The fixpoint iteration over the equations computes elements of the chains in XB and the length of a chain is bounded by $3 \times r \times w$. Recall that 3 is the height of the extensivity token domain $T = \{\bot_T, 0, -, +, \top_T\}$. Thus we iterate up to $O(r \times w)$ time. Each iteration per equation takes $O(r^2 \times w^2 \times n)$ time because each equation computes a new extensivity behavior xb whose size is $O(r \times w)$, the right-hand-side of each equation can have up to n αs, and refinement π takes $O(r \times w)$ because it computes a fixpoint. Hence the worst-case time complexity of the fixpoint iteration is $O(n \times (r \times w) \times (r^2 \times w^2 \times n)) = O(n^2 \times w^3 \times r^3)$. Hence the overall complexity is $O(n^{2r}) + O(n^2 \times w^3 \times r^3)$. ■

In our practice the largest type size r is mostly 2; it means in our input specifications of abstract interpreters, most functions are first-order. Hence the number w of variables is linear to the expression size n, and thus we expect the complexity in practice is bounded by $O(n^5)$.

§6 Example: Extensivity Checking of A Widening/Narrowing Operator

We give an example of static extensivity analysis for the widening and narrowing operators[4] of abstract interpretations. The use of widening and narrowing operators is common in abstract interpretation when the height of the domain lattice is infinite. A widening operator allows the fixpoint iteration

[*2] Type size is the number of basic type-components (lattices in our case). For example, type $(L \rightarrow L) \rightarrow (L \rightarrow L)$ has size 4.

reaches a stable point in a finite time, and a narrowing operator refines the stable point for an accuracy improvement.

Because a widening and narrowing operators are, by definition, extensive (and reductive, respectively) but not monotonic, they are good candidates for testing our extensivity analysis.

A widening $\nabla : L \times L \rightarrow L$ is a function such that:

1. $\forall x, y \in L : x \sqsubseteq_L x \nabla y$
2. $\forall x, y \in L : y \sqsubseteq_L x \nabla y$
3. for all increasing sequences $\{x_n\}_n$, the "widened" sequence $y_0 = x_0$, ..., $y_{n+1} = y_n \nabla x_{n+1}$, ... eventually stabilizes.

Condition 1 and 2 say that the widened value $x \nabla y$ should always be greater than or equal to x and y, i.e., extensive for both x and y.

Let's consider the following widening operator:

$$x \nabla y = \mathit{if}\ y \sqsubseteq_L x\ ?\ x\ :\ \top_L.$$

This widening raises unstable elements up to the greatest element $\top_L$. A well-known widening on the integer interval domain[2, 4)] is a straightforward instance of this widening. Note that this widening operator is not monotonic because for all $x, y \in L$ s.t. $x \sqsubset_L y \sqsubset_L \top_L$, $y \nabla y = y \sqsubset_L \top_L = x \nabla y$.

Our extensivity analysis computes the extensivity of $x \nabla y$ as $\{x^+, y^+\}$, which confirms $x \nabla y$ is extensive for both x and y. The constraint generation procedure $C(\{x : \{x^0\}, y : \{y^0\}\}, \mathit{if}\ y \sqsubseteq_L x\ ?\ x\ :\ \top_L, \alpha)$ returns the following constraints:

$$\begin{array}{llll} \alpha_1 \supseteq \{y^0\} & \alpha_2 \supseteq \{x^0\} & \alpha_3 \supseteq \{x^0\} & \alpha_4 \supseteq \{x^+, y^+\} \\ \multicolumn{4}{l}{\alpha \supseteq \pi(\alpha_1, \alpha_2, \alpha_3) \sqcup_{XB} \alpha_4.} \end{array}$$

The above constraints constitute the equations:

$$\begin{array}{llll} \alpha_1 = \{y^0\} & \alpha_2 = \{x^0\} & \alpha_3 = \{x^0\} & \alpha_4 = \{x^+, y^+\} \\ \multicolumn{4}{l}{\alpha = \pi(\alpha_1, \alpha_2, \alpha_3) \sqcup_{XB} \alpha_4.} \end{array}$$

The least solution for α is computed as follows:

$$\begin{array}{lcl} \alpha & = & \pi(\{y^0\}, \{x^0\}, \{x^0\}) \sqcup_{XB} \{x^+, y^+\} \\ & = & \{x^0, y^+\} \sqcup_{XB} \{x^+, y^+\} \\ & = & \{x^+, y^+\}. \end{array}$$

Note that without refinement π, we compute a less accurate extensivity:

$$\begin{aligned} \alpha &= \alpha_3 \sqcup_{XB} \alpha_4 = \{x^0\} \sqcup_{XB} \{x^+, y^+\} = \{x^0, y^{\top_T}\} \sqcup_{XB} \{x^+, y^+\} \\ &= \{x^{\top_T}, y^+\}. \end{aligned}$$

We can similarly compute the extensivity of narrowing operators. A narrowing $\triangle : L \times L \to L$ is a function such that:

1. $\forall x, y \in L : (y \sqsubseteq_L x) \Rightarrow (y \sqsubseteq_L x \triangle y \sqsubseteq_L x)$
2. for all decreasing sequences $\{x_n\}_n$, the "narrowed" sequence $y_0 = x_0$, ..., $y_{n+1} = y_n \triangle x_{n+1}$, ... eventually stabilizes.

Condition 1 says that $x \triangle y$ is reductive for x and extensive for y when $y \sqsubseteq_L x$.

Let's consider the following narrowing operator:

$$x \triangle y = \mathit{if}\ \top_L \sqsubseteq_L x\ ?\ y\ :\ x.$$

This narrowing refines the widened result $\top_L$ to the currently known lower bound y. In example 3.5, we show that the extensivity of the above expression is $\{x^-, y^+\}$ under order assumption $\{y \sqsubseteq_L x\}$, i.e., $(y \sqsubseteq_L x) \Rightarrow (y \sqsubseteq_L x \triangle y \sqsubseteq_L x)$. We can infer the same extensivity $\mathsf{xb} = \{x^-, y^+\}$. First, we give a constraint $\mathsf{xb} \supseteq \pi(\{y^0\}, \{x^0\}, \alpha)$ from the initial order assumption $\{y \sqsubseteq_L x\}$. Then we extract the constraints for α by $C(\{x : \{x^0\}, y : \{y^0\}\}, x \triangle y, \alpha)$. Finally we solve those constraints as usual: simplification, equation generation, and fixpoint iteration.

§7 Conclusion

Our work provides a method of extensivity verification for λ-definable functions over arbitrary finite-height lattices. Static extensivity analysis is an interesting problem on its own and apparently not much work has been done. Our interest in this topic was motivated by Zoo[15], which is a program-analyzer generator. Now that it can automatically check whether the input specification is extensive or not, termination of the specified analysis is guaranteed if the outcome of the test is positive. Thus we can prevent Zoo from generating divergent analyzers, or from generating extra "joining" operations[8, 9] necessary to enforce the extensivity of fixpoint iterations. Our work may also suggest a similar solution to existing program analyzer generators like PAG[10].

The extensivity analysis is complementary to the previous work[11] on the static monotonicity analysis(checking $\forall x \leq y : f(x) \leq f(y)$). Because the extensivity and monotonicity are incomparable properties and either of them guarantees the termination of the generated analyzers, the two analyses in disjunctive combination enlarges the set of sound input specifications for Zoo. Our verification procedure is an inference system, which can be classified as monovariant flow-insensitive analysis. Our extensivity analysis is a kind of relational analysis, which estimates each expression's extensivity in relations with (as a function of) its free variables.

References

1) Alexander Aiken and Edward L. Wimmers. Type inclusion constraints and type inference. In *Proceedings of Functional Programming Languages and Computer Architecture*, pages 31–41, 1993.
2) Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.
3) Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, 1979.
4) Patrick Cousot and Radhia Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. Technical Report LIX/RR/92/09, Ecole Polytechnique, 1992.
5) Patrick Cousot and Radhia Cousot. Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form. In *Proceedings of the 7th International Conference on Computer-Aided Verification*, volume 939 of *Lecture Notes in Computer Science*, pages 293–308. Springer-Verlag, 1995.
6) Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. Number Report TR99-107 in Electronic Colloquium on Computational Compleity, June 1999.
7) Oded Goldreich, Shafi Goldwasser, Eric Lehman, and Dand Ron. Testing monotonicity. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998.
8) B. Le Charlier and P. Van Hentenryck. A universal top-down fixpoint algorithm. Technical Report TR-CS-92-25, Brown University, Dept. of Computer Science, May 1992. (also as a technical report of Institute of Computer Science, University of Namur).
9) B. Le Charlier and P. Van Hentenryck. Experimental Evaluation of a Generic Abstract Interpretation Algorithm for Prolog. *ACM Transactions on Programming Languages and Systems*, 16(1):35–101, January 1994.
10) Florian Martin. PAG – an efficient program analyzer generator. *International Journal on Software Tools for Technology Transfer*, 2(1):46–67, 1998.
11) Andrzej Murawski and Kwangkeun Yi. Static monotonicity analysis for lambda-definable functions over lattices. In *The Proceedings of the 3rd International*

Workshop on Verification, Model Checking and Abstract Interpretation, volume 2294 of *Lecture Notes in Computer Science*, pages 139–153, January 2002.

12) D.E. Rutherford. *Introduction to Lattice Theory.* Hafner Publishing Company, New York, 1965.

13) Winfrid G. Schneeweiss. A necessary and sufficient criterion for the monotonicity of boolean functions with deterministic and stochastic. *IEEE Transactions on Computers*, 45(11):1300–1302, November 1996.

14) Andrei Voronenko. On the complexity of the monotonicity verification. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, pages 4–7, July 2000.

15) Kwangkeun Yi. *Program Analysis System Zoo.* Research On Program Analysis: National Creative Research Center, KAIST, July 2001. `http://ropas.kaist.ac.kr/zoo/doc/rabbit-e.ps`.