PAPER

Transformation of Strictness-Related Analyses Based on Abstract Interpretation*

Mizuhito OGAWA† *and* Satoshi ONO††, *Members*

SUMMARY This paper newly proposes HOMomorphic Transformer (HOMT) in order to formalize relations among strictness-related analyses (SRAs) on first-order functional programs. A HOMT is defined to be a composition of special instances of abstract interpretation, and has enough ability to treat known SRAs including head/tail/total strictness detection on nonflat domains. A set of HOMTs, furthermore, is an algebraic space such that some composition of HOMTs can be reduced to a simpler HOMT. This structure gives a transformational mechanism between various SRAs, and further clarifies the equivalence and the hierarchy among them. First, we show a construction of a HOMT as a composition of Unit-HOMTs (U-HOMTs) which are specified by quadruplet representations. Second, algebraic relations among HOMTs are shown as reduction rules among specific pairs of quadruplet representations. Thus, hierarchy among HOMTs can be clarified by finding some adequate quadruplet representation which bridges a HOMT to the other. Third, various SRAs are formalized as HOMTs in either forward or back-ward manners. They are also shown to be safe under unified discussions. Finally, their equivalence and hierarchy are examined in terms of an algebraic structure of HOMTs.

1. Introduction

Stimulated by an urgent need for efficient implementation of lazy applicative languages, many Global Dataflow Analyses (GDAs) have been proposed(1),(13),(20). A most notable class in GDAs is a set of Strictness-Related Analyses (in short, SRAs), that collect information on the strictness of functions. SRAs(17) are classified into either Strictness Analysis (SA)(3),(5),(9),(12),(22), Relevance Analysis(RA) or Com putation Path Analysis (CPA)(18),(19). An SA detects a set of parameters that should be evaluated to obtain the resulting value of a function. Conversely, an RA detects a set of parameters that may be evaluated. CPA is a generalization of both SA and RA, detecting a set of all possible computation paths.

Abstract interpretation(1),(6),(11),(12) and projection analysis(4),(22) have been proposed as the basis for formalizing various GDAs. Abstract interpretation executes a program for all instances on a (possibly finite) abstract domain which reflects the objective property. In contrast, projection analysis interprets a program as a transformer on a (possibly finite) selections of projections, which reflect the objective property. Equivalence between their certain sub classes was investigated in Ref. (4). However, there are still remaining the following questions. Let us restrict discussions on SRAs.

• In Ref. (22), Wadler indicated that head-strictness detection on nonflat domain cannot be treated by a previously proposed abstract interpretation, although projection analysis can. In Ref. (4), Burn divided head-strictness into two parts: H-strictness and H_B-strictness. He showed that H_B-strictness can be treated by abstract interpretation. In fact, it is much easier to detect as a by-product property in total/tail strictness analysis. Then, the question is, can H-strictness also be treated by abstract interpretation ?

• Even restricting discussions on abstract interpretation, there are various algorithms to have been proposed. For instance, SAs on flat domain were proposed in both forward/backward manners(12),(20),(9). Furthermore, a result of some analysis intuitively introduces results of the others. For instance, this is a case of CPA and SAs. Then, the question is, how these equivalence and hierarchy can be formally shown ?

This paper newly proposes HOMomorphic Transformer(HOMT) in order to formalize relations among strictness-related analyses(SRAs) on first-order functional programs. A HOMT is defined to be a composition of special instances of abstract interpretation, and has enough ability to treat known SRAs including head/tail/total strictness detection on nonflat domains. A set of HOMTs, furthermore, is an algebraic space such that some composition of HOMTs can be reduced to a simpler HOMT. This structure gives a transformational mechanism between various SRAs, and further clarifies the equivalence and the hierarchy among them.

First, we show a construction of a HOMT as a composition of Unit-HOMTs (U-HOMTs) which are specified by quadruplet representations. Second, algebraic relations among HOMTs are shown as reduction rules among specific pairs of quadruplet representa-

Manuscript received October 4, 1990.
† The author is with NTT Basic Research Laboratories, Musashino-shi, 180 Japan.
†† The author is with NTT Software Laboratories, Tokyo, 108 Japan.

* This paper was recommended by the Editorial Committee.

tions. Thus, hierarchy among HOMTs can be clarified by finding some adequate quadruplet representation which bridges a HOMT to the other. Third, various SRAs are formalized as HOMTs in either forward or backward manners. They are also shown to be safe under uniffed discussions. Finally, their equivalence and hierarchy are examined in terms of an algebraic structure of HOMTs.

For simplicity, we will restrict the arguments to strongly-typed first-order functional programs with streams instead of general lists. Note that the techniques are not exclusive of typeless programs except that classifications among flat/nonflat domains and Boolean values which decide the choices of conditional branches.

2. Intuitive Comparison among SRAs

2.1 SRAs as HOMTs, Overview

SRAs are made up of three kinds of GDAs. That is, SAs, RAs and CPAs[17]. CPA computes the Property Dependency Parameter Set (PDPS), which is intuitively a set of all possible demand patterns of the function when demands are propagated to resulting value. RA detects relevant parameters which may need to be evaluated when demands propagate. SA detects requisite parameters which always need to be evaluated when demands propagate.

On the other hand, a HOMT, h, is a functional which maps continuous functions f on computational domains to $h(f)$ on (possibly finite) abstract domains (See Sect. 3.2). Thus, SRAs are formalized by HOMTs as follows. Assume h be a HOMT such that $h(f)$ preserves the objective property of an SRA on the original program f. Note that $h(f)$ may be not computable even if abstract domains are finite, since $h(f)$ reflects the exact run-time property which is never clarified before actual execution. Thus, the algorithms of SRAs are formalized according to the following two steps. First, compute the approximation $h^c(f)$ of $h(f)$, where $h^c(f)$ is the solution of some recursive equation on abstract domains. This result is called the computed HOMT (See Sect. 3.3). Next, execute $h^c(f)$ for all possible instances on abstract domains. As a result, the approximated property on f is detected, instead of the exact but noncomputable run-time property on f.

With the formalizations of SRAs as above, the equivalence of two SRAs which have different corresponding HOMTs h_1, h_2 is clarified by finding HOMTs h_{12}, h_{21} that transform to each other as $h_1(f) \xrightarrow{h_{12}} h_2(f)$ and $h_2(f) \xrightarrow{h_{21}} h_1(f)$ for each continuous function f. Similarly, the hierarchical relationship between two SRAs is clarified by finding a HOMT that transforms the stronger one into the weaker one.

Sections 2.2 and 2.3 examine both the equivalence and the hierarchy of the already proposed algorithms of SRAs.

2.2 Forward/Backward Equivalence among SRAs

An SRA may be either forward or backward. As SRA is said to be a forward analysis, if it clarifies the properties of results from the properties of parameters. Conversely, an SRA is said to be a backward analysis, if it clarifies the conditions satisfied by parameters from the properties of results.

Let us concentrate on SAs on flat domains (i.e. Integer, Boolean, etc). Forward SA (FSA) is an example of a forward analysis[12]. Similar algorithms are found in Refs. (1), (5).

FSA interprets a function, f, on a flat domain (such as Integer or, Boolean) to a $\{\mathbf{0}, \mathbf{1}\}$-valued function f_{FSA}, where $\mathbf{0}$ means totally undefined and $\mathbf{1}$ means possibly defined. Thus, f_{FSA} returns $\mathbf{1}$ if there possibly exists a computable real instance of f, and returns $\mathbf{0}$ if there never exists a computable real instance of f. For instance, ordinary $if\,(x, y, z)$ is interpreted to

$$\begin{aligned} if_{FSA} &: (\mathbf{1},\mathbf{1},\mathbf{1}) \to \mathbf{1}, \quad (\mathbf{1},\mathbf{1},\mathbf{0}) \to \mathbf{1}, \quad (\mathbf{1},\mathbf{0},\mathbf{1}) \to \mathbf{1}, \\ &: (\mathbf{0},\mathbf{1},\mathbf{1}) \to \mathbf{0}, \quad (\mathbf{1},\mathbf{0},\mathbf{0}) \to \mathbf{0}, \quad (\mathbf{0},\mathbf{1},\mathbf{0}) \to \mathbf{0}, \\ &: (\mathbf{0},\mathbf{0},\mathbf{1}) \to \mathbf{0}, \quad (\mathbf{0},\mathbf{0},\mathbf{0}) \to \mathbf{0}. \end{aligned}$$

Then, requisite parameters can be detected by firstly testing f_{FSA} for all $\{\mathbf{0}, \mathbf{1}\}$-input patterns, next collecting the set of minimum input patterns that returns $\mathbf{1}$ (called $\mathbf{1}$-frontier in Ref. (1)), and finally detecting requisite parameters that are always required to be $\mathbf{1}$ in all patterns in the $\mathbf{1}$-frontier. For instance, $if\,(x, y, z)$, the $\mathbf{1}$-frontier is $\{(\mathbf{1},\mathbf{1},\mathbf{0}), (\mathbf{1},\mathbf{0},\mathbf{1})\}$, and then, the requisite parameter is x.

On the other hand, Boolean-algebraic SA (BSA)[9],[17] is an example of SA as a backward analysis. BSA interprets a function, f, to a function f_{BSA} which is a symbolic manipulation on the set-characteristic expressions of input parameters. For example, $if\,(x, y, z)$ is interpreted to $if_{BSA}(x, y, z) = \lambda xyz.(x \cup y) \cap (x \cup z)$. Then, requisite parameters are collected by substituting actual variable names to corresponding setcharacteristic expressions. For instance, requisite parameters of $if\,(a, b, b)$ are detected as $(\lambda xyz.(x \cup y) \cap (x \cup z))(\{\boldsymbol{a}\}, \{\boldsymbol{b}\}, \{\boldsymbol{b}\}) = (\{\boldsymbol{a}\} \cup \{\boldsymbol{b}\}) \cap (\{\boldsymbol{a}\} \cup \{\boldsymbol{b}\}) = \{\boldsymbol{a}, \boldsymbol{b}\}$.

In both FSA and BSA, these interpretations for user defined functions are induced by ordinary fixpoint calculus based on given interpretations on primitive functions on abstract domains.

From an application view point, both FSA and BSA have equivalent analytical ability except that FSA can detect diverged functions when its $\mathbf{1}$-frontier is an empty set, whereas BSA cannot.

2. 3 Hierarchy among SRAs

The hierarchy of SRAs arises from 2 reasons:

• Objective property of program is the same for SRA_1 *and* SRA_2, but abstraction is more detailed on SRA_2 than SRA_1 (Approximation hierarchy).

• Objective property of program itself is more informative on SRA_2 than SRA_1 (Property hierarchy).

The approximation hierarchy arises from the fact that an SRA is a compile-time technique whereas the objective property is a run-time property. Thus, approximation accuracy is traded off with computational complexity (including termination). Therefore, even for the same objective property, there are many selections for approximation levels. These levels can be measured by domain abstractions.

The notable examples are SAs on non-flat domains (e. g. lazy list structures, streams) (1),(5),(10),(12),(8),(22). The notable feature of non-flat domains is non-strictness. That is, lazy functions such as *cons-stream* (x, y), *head*(x), *tail*(x) (as in Scheme) admit partially evaluated data for both inputs and outputs.

A trivial extension of FSA interprets functions to $\{\mathbf{0}, \mathbf{1}\}$-valued functions where the data structure is approximated as **1** if the evaluation requires a head-normal form (i. e. not delayed), and as **0** if delayed.

Conversely, Tail Strictness Analysis (Tail SA) on streams (8), (22) interprets functions as $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{NIL}\}$-valued functions. In the abstract domain $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{NIL}\}$, **0** means never evaluated, **1** means values that are evaluated until the head normal form is clarified, **2** means values that are evaluated until the spine of a list is clarified, and **NIL** means eventually evaluated to **Nil**.

Thus, for instance, a parameter x in length(x) is analyzed as simply not delayed by FSA, whereas it is analyzed as requisite at level **2** or NIL by Tail SA. Detailed discussions on SRAs on nonflat domains are found in Sect. 4. 2.

The property hierarchy is found in the relation among CPA, SA, and RA(14),(17). For example,

$$\begin{aligned} f(x, y, z, p, q) = {} & \text{if } p>0 \text{ then} \\ & (\text{if } p=1 \text{ then } (\text{if } z=0 \text{ then } x \text{ else } y) \\ & \text{else } f(z, z, 0, p-1, x)) \\ & \text{else } f(0, 0, z, 1, y) \end{aligned}$$

is analyzed as

$$\begin{cases} \text{PDPS} & \{\{x, z, p\}, \{y, z, p\}, \{z, p\}, \{p\}\} \\ \text{relevant parameters} & \{x, y, z, p\} \\ \text{requisite parameters} & \{p\} \end{cases}$$

Roughly speaking, the union of all elements in PDPS is a set of relevant parameters, and the intersection is a set of requisite parameters. Therefore, CPA is more powerful than both SA and RA.

3. Algebraic Structure on HOMTs

3. 1 Construction of U-HOMTs and Their Quadruplet Representations

A HOMT is defined to be a functional which maps continuous functions on computational domains to those on possibly finite abstract power domains. A HOMT is constructed as a composition of U-HOMTs. A U-HOMT h is a functional from a directed relation on a domain D to a directed relation on a possibly finite abstract domain *Abs*. Let us describe them more formally.

[Definition 1] Let D and Abs be cpo's(2). A map $abs: D \to Abs$ is said to be a domain abstraction if abs is an onto map and bottom-reflecting(4) (i. e. $abs(x) = \bot_{Abs}$ implies $x = \bot_D$).

A preorder $\sqsubseteq$ is defined to be a relation which satisfies a reflexive law (i. e. $X \sqsubseteq X$) and a transitive law (i. e. $X \sqsubseteq Y$ and $Y \sqsubseteq Z$ implies $X \sqsubseteq Z$), but may not satisfies an asymmetric law (i. e. $X \sqsubseteq Y$ and $X \sqsupseteq Y$ implies $X = Y$).

[Definition 2] Let D be a cpo. A power domain $PD[D]$(21) associated to a preorder $\sqsubseteq$ is defined to be $PD[D] \overset{\text{def}}{=} \{closure(X) \mid X(\neq \phi) \subseteq D\}$, where a pair of a closure function and a preorder $(closure(X), \sqsubseteq)$ is either $(RC(X), \sqsubseteq_0)$, $(LC(X), \sqsubseteq_1)$, $(Conv(X), \sqsubseteq_{EM})$, or $(id, \subseteq)$ (Definitions of these representative functions and preorders are shown in Table 1). $PD[D]$ with $(id, \subseteq)$ is specifically called a power set and noted as $PS[D]$.

[Definition 3] Let D_1 and D_2 be domains. Relation $\sim$ between them is defined to be a subset $A_\sim \subseteq D_1 \times D_2$. The relation is noted as $x \sim y$ if $(x, y) \in A_\sim$.

The relation $\sim$ is said to be directed if $\forall x \in D_1 \exists y \in D_2$ s. t. $x \sim y$, and noted $x \overset{\sim}{\rightarrow} y$. In case of $D_1 = D^n$ and $D_2 = D$ for some domain D and integer n, we will call that the relation is forward if $D_1 \overset{\sim}{\rightarrow} D_2$, and is backward if $D_1 \overset{\sim}{\leftarrow} D_2$.

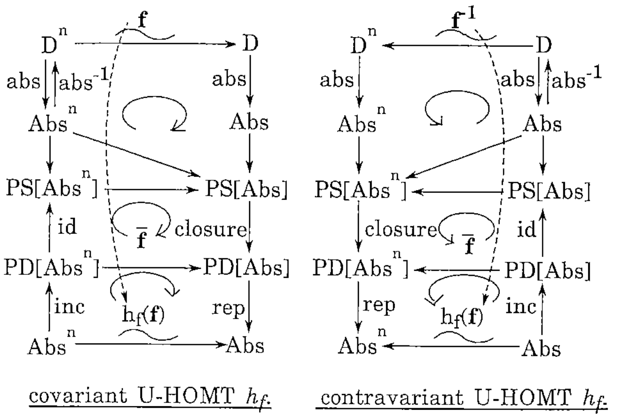
A U-HOMT, which is a HOMT induced from a domain abstraction, is classified into two types. That is, covariant U-HOMTs and contravariant U-HOMTs, corresponding to forward analyses and backward analyses. A covariant U-HOMT transforms a directed relation to those that of a same direction (i. e. a forward relation to a forward one, and a backward relation to a backward one), and a contravariant U-HOMT transforms it to those that of a contrary direction (i. e. a forward relation to a backward one, and a backward relation to a forward one). Their constructions are shown in commutative diagrams in Fig. 1. A U-HOMT is constructed in three steps. For instance, a covariant U-HOMT is as follows: First, a directed

Table 1 Definitions of preorder related notations.

preorder	**definition**	**closure function** / **representative function**
$X \sqsubseteq_{EM} Y$	$X \sqsubseteq_0 Y \wedge X \sqsubseteq_1 Y$	$Conv(X) \overset{\text{def}}{=} LC(X) \cap RC(X)$
		$Conv(X)$
$X \sqsubseteq_0 Y$	$RC(X) \supseteq RC(Y)$	$RC(X) \overset{\text{def}}{=} \{x \in D \mid \exists y \in X \text{ s.t. } y \sqsubseteq x\}$
		$Min(X) \overset{\text{def}}{=} \{x \in X \mid \neg\exists y \in X \text{ s.t. } y \sqsubset x\}$
$X \sqsubseteq_1 Y$	$LC(X) \subseteq LC(Y)$	$LC(X) \overset{\text{def}}{=} \{x \in D \mid \exists y \in X \text{ s.t. } x \sqsubseteq y\}$
		$Max(X) \overset{\text{def}}{=} \{x \in X \mid \neg\exists y \in X \text{ s.t. } x \sqsubset y\}$

Table 2 Typical selections of parameters of quadruplet representations.

domain abstraction		**direction**		**preorder**		**representative function**
base domain abstraction	×	covariant $(+)$	×	$\subseteq$	×	id
$abs_b : x \rightarrow \begin{cases} 1 & (\text{if } x \not\equiv \bot) \\ 0 & (\text{if } x \equiv \bot) \end{cases}$				$\sqsubseteq_{EM}$		$Conv$
(where $\mathbf{0} \sqsubseteq \mathbf{1}$.)		contravariant $(-)$		$\sqsubseteq_0$		Min
identical abstraction				$\sqsubseteq_1$		Max
$abs_{id} : x \rightarrow x \quad (\forall x \in D)$						



$$\begin{tikzcd}
D^n \arrow[r, "f"] \arrow[d, shift right, "abs"'] & D \arrow[d, "abs"] \\
Abs^n \arrow[u, shift right, "abs^{-1}"'] \arrow[d] \arrow[rd] & Abs \arrow[d] \\
PS[Abs^n] \arrow[r] & PS[Abs] \arrow[d, "closure"] \\
PD[Abs^n] \arrow[u, "id"] \arrow[r, "\bar{f}"] & PD[Abs] \arrow[d, "rep"] \\
Abs^n \arrow[u, "inc"] \arrow[r, "h_f(f)"] & Abs
\end{tikzcd}
\qquad
\begin{tikzcd}
D^n \arrow[d, "abs"'] & D \arrow[l, "f^{-1}"'] \arrow[d, shift left, "abs"] \\
Abs^n \arrow[d] & Abs \arrow[u, shift left, "abs^{-1}"] \arrow[d] \arrow[ld] \\
PS[Abs^n] \arrow[d, "closure"'] & PS[Abs] \arrow[l] \\
PD[Abs^n] \arrow[d, "rep"'] & PD[Abs] \arrow[u, "id"'] \arrow[l, "\bar{f}"'] \\
Abs^n & Abs \arrow[u, "inc"'] \arrow[l, "h_f(f)"']
\end{tikzcd}$$

Fig. 1 Construction of U-HOMT h_f.

relation $f : D^n \overset{\sim}{\rightarrow} D$ is transformed to a function on a powerset of abstract domain as $abs \circ f \circ abs^{-1}$: $PS[Abs^n] \rightarrow PS[Abs]$. Second, it is naturally embedded into a function $\bar{f} = closure \circ abs \circ f \circ abs^{-1} \circ id : PD[Abs^n] \rightarrow PD[Abs]$ on a power domain of an abstract domain. Finally, $h(f)$ is realized as a directed relation as $h(f) : x \overset{\sim}{\rightarrow} rep \circ abs \circ f \circ abs^{-1} \circ inc(x)$ where rep is a representative function and a map $inc : Abs^n \rightarrow PD[Abs^n]$ is defined to be $inc(x) = closure(\{x\})$. On the other hand, a contravariant U-HOMT h is defined to be $h(f) : x \overset{\sim}{\rightarrow} rep \circ abs \circ f^{-1} \circ inc(x)$. (Note that $rep \circ closure = rep$.)

A result of a contravariant U-HOMT $h(f)$ is induced from an inverse relation f^{-1}, whereas a result of a covariant U-HOMT is induced from a relation f itself. Thus, $h(f)$ may not be well-defined when f is not an onto relation. That is, The function inverse may require ϕ as a result value, but a power domain does not include ϕ. To avoid such a case, we extend a power domain $PD[D]$ with an empty set ϕ. The preorders are naturally extended except the case of a power domain with ($\sqsubseteq_{EM}$, Conv). Thus, as an exception, we abopt a power domain with ($\sqsubseteq_{EM}$, Conv) only for covariant HOMTs.

The parameters which specify these U-HOMT constructions are a domain abstraction, a direction (that is, whether covariant or contravariant), a power domain construction, and a representative function. By definition, a power domain construction is composed of the selection of preorder $\sqsubseteq$ on power set combined with a closure function *closure*. Table 2 presents selections for these parameters.

[Definition 4] A U-HOMT, h, is specified by a domain abstraction $abs : D \rightarrow Abs$, a direction *dir* (whether covariant $(+)$ or contravariant $(-)$), a preorder $\sqsubseteq$, and a representative function *rep*. A quadruplet (*abs, dir,* $\sqsubseteq$, *rep*) is called the quadruplet representation of h.

3.2 Algebraic Relations among HOMTs

A U-HOMT is a functional from a directed relation to a directed relation on an abstract domain. A HOMT is a composition of U-HOMTs, and specifically restricts its application on continuous functions $f : D^n \rightarrow D$ which is a special instance of a forward relation with a prefix notation. We will freely convert a direct relation to a function on a power domain using a representative function *rep* and a closure function *closure*. More precisely, a directed relation $rep \circ \bar{f} \circ closure$ on a domain corresponds to a function $\bar{f}$ on a power domain.

[Definition5] A HOMT is defined to be a composition of U-HOMTs regarding a function $f : D^n \rightarrow D$ as a forward relation. A HOMT is also represented as a composition of quadruplet representations.

Let a HOMT h be a composition of U-HOMTs $h_n \circ \cdots \circ h_1$. A HOMT is said to be *covariant* if a product of all directions in each quadruplet expression of U-HOMTs $h_1, \cdots, h_n$ is positive, and said to be contravariant if a product of all directions is negative. From this definition, the name of a covariant HOMT is validated as it transforms a directed relation to those that of same direction (i. e. a forward relation to a forward relation, and a backward relation to a backward relation). Similarly, the name of a contravariant HOMT is validated as it transforms a directed relation to those that of different direction (i. e. a forward relation to a backward relation, and a backward relation to a forward relation).

Some compositions of U-HOMTs may eventually coincide with a simpler U-HOMT. This shows that some HOMTs may be equivalent although they have different representations as compositions of U-HOMTs. This equivalence is introduced from the next Reduction theorem, and defines an algebraic structure on HOMTs. Before introducing it we prepare two notations on preorders.

[Definition 6] The order $\ll$ among preorders $\sqsubseteq$ is defined to be $\sqsubseteq' \ll \sqsubseteq$ iff $X \sqsubseteq Y \Rightarrow X \sqsubseteq' Y$.

[Definition 7] The preorder $\sqsubseteq_{-*}$ is defined to be $X \sqsubseteq_{-*} Y$ iff $X \sqsupseteq_* Y$ where $\sqsubseteq_*$ is either $\sqsubseteq_0$, $\sqsubseteq_1$, $\sqsubseteq_{EM}$, or $\sqsubseteq_{set}$ (i. e. $\subseteq$).

The lattice due to this ordering $\ll$ is presented in Fig. 2. Intuitively speaking, $\sqsubseteq' \ll \sqsubseteq$ means $\sqsubseteq$ has more detailed information than $\sqsubseteq'$. The first part of the next theorem follows easily from the definition above. The second part follows by a long but routine process of set-theoretical relations.

[Theorem 1] Let h and h' be U-HOMTs, and $(abs, dir, \sqsubseteq, rep)$ and $(abs', dir', \sqsubseteq', rep')$ be their quadruplet representations, where $abs : D_1 \rightarrow D_2$ and $abs' : D_2 \rightarrow D_3$ are domain abstractions. If one of following conditions is satisfied, a composition $h' \circ h$ is reduced to a single U-HOMT as $(abs', dir', \sqsubseteq', rep') \circ (abs, dir, \sqsubseteq, rep) = (abs' \circ abs, dir' \circ dir, \sqsubseteq', rep')$

(1) For $dir' = +$,

either (a) $\sqsubseteq' \ll \sqsubseteq$ or (b) $\sqsubseteq' \ll \sqsubseteq_-$

(2) For $dir' = -$,

either (a) $(\sqsubseteq, \sqsubseteq') = (\sqsubseteq_{\pm 0}, \sqsubseteq_{\pm 1}) \wedge rep = \text{Min}$,

or (b) $(\sqsubseteq, \sqsubseteq') = (\sqsubseteq_{\pm 1}, \sqsubseteq_{\pm 0}) \wedge rep = \text{Max}$,

or (c) $(\sqsubseteq, \sqsubseteq') = (\sqsubseteq, \sqsubseteq)$

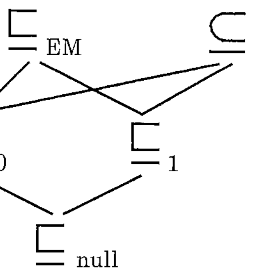


Fig. 2 Relations among preorders.

3.3 Computation of a HOMT and Its Safeness

A HOMT is defined as above, but the algorithm to compute it remains still unspecified. In fact, $h(f)$ that reflects a run-time property of a function (program) f is not computable, even if the abstract domain *Abs* is finite. Therefore, instead of $h(f)$, we introduce $h^c(f)$ which approximates $h(f)$, and is computable if the abstract domain is finite. Thus, a computation of $h^c(f)$ exactly corresponds to an algorithm of an SRA on a program f, whereas a HOMT h can be regarded as a property extractor of a program.

[Definition 8] Let f be a $fix(\tau) \equiv \sqcup_n \tau^n(\Omega)$, for a recursion equation τ and an undefined function Ω which maps every elements to a bottom element $\perp$. Let h be a HOMT. A computed HOMT h^c is defined to be $h^c(f) \overset{\text{def}}{=} \sqcup_n (h_\tau(\tau))^n (h(\Omega))$, where $h_\tau(\tau)$ is defined to be a syntactically identical (resp. inverse) equation, but replaces each primitive functions priv with h(priv), if h is a covariant (resp. contravariant) HOMT.

Intuitively, $h^c(f)$ is a result of a fixed-point computation on a power domain of an abstract domain, starting with $h(\Omega)$. From a viewpoint of SRAs as HOMTs, a computed HOMT h^c must properly approximates a HOMT h. For instance, SRAs on flat domains must satisfies conditions shown in Table 3. These conditions are generalized to safeness condition as defined below.

[Definition 9] A HOMT h is said to be safe *iff* $h(f) \sqsubseteq h^c(f)$ for all continuous functions f where an order $\sqsubseteq$ is associated to a power domain construction of a target abstract domain of h.

Table 3 Safeness of SRAs on flat domains.

SRA	real property		detected property
SA	*real strict parameters*	$\supseteq$	*detected strict parameters*
RA	*real relevant parameters*	$\subseteq$	*detected relevant parameters*
CPA	*real PDPS*	$\subseteq$	*detected PDPS*

In general, safeness $h \sqsubseteq h^c$, is shown by two steps. That is, first show $h(f \circ g) \sqsubseteq h(f) \circ h(g)$ if h is covariant, and $h(f \circ g) \sqsubseteq h(g) \circ h(f)$ if h is contravariant. Second, show $h(f \sqcup g) \sqsubseteq h(f) \sqcup h(g)$. The first condition guarantees $h(f^{(i)}) \sqsubseteq (h_T(\tau))^i(h(\Omega))$. Thus, $\sqcup_i h(f^{(i)}) \sqsubseteq \sqcup_i(h_T(\tau))^i(h(\omega)) = h^c(f)$. The second condition guarantees $h(f) = h(\sqcup_i f^{(i)}) \sqsubseteq \sqcup_i h(f^{(i)})$. Therefore, safeness $h(f) \sqsubseteq h^c(f)$ is shown.

The first condition $h(f \circ g) \sqsubseteq h(f) \circ h(g)$ is satisfied if h is a U-HOMT and a preorder $\sqsubseteq$ is cooperative with $\subseteqq$ (i. e. $X \sqsubseteq Y \Longrightarrow$ *closure*$(X) \subseteqq$ *closure*(Y).). This is the case $\sqsubseteq \in \{\sqsubseteq_{-0}, \sqsubseteq_1, \subseteqq\}$. The special case of the second condition is $h(f \sqcup g) = h(f) \sqcup h(g)$. This means h is continuous. This is the case in Theorem 2.

[Theorem 2] Let h be a U-HOMT s. t. h has a quadruplet representation $(abs, dir, \sqsubseteq, rep)$. Then h and their compositions safe HOMTs if the following conditions are satisfied.

• The domain abstraction *abs* is continuous, and its inverse abs^{-1} satisfies $x \sqsubseteq y \Longrightarrow abs^{-1}(x) \sqsubseteq_{EM} abs^{-1}(y)$ for any elements x, yin an abstract domain.

• The pair $(dir, \sqsubseteq, rep)$ is one of the followings: $(+, \sqsubseteq_{-0}, \mathrm{Min})$, $(+, \sqsubseteq_1, \mathrm{Max})$, $(-, \sqsubseteq_{-0}, \mathrm{Min})$, $(-, \sqsubseteq_1, \mathrm{Max})$

[Theorem 3[11]] The notations are same as Theorem 2. If a domain abstraction $abs: D \rightarrow Abs$ satisfies the same condition as Theorem 2, a U-HOMT h with a quadruplet representation $(+, \sqsubseteq_{EM}, Conv)$ is a safe HOMT.

When a domain abstraction is not continuous, the next theorem is useful. This theorem is obtained by pursuing set-inclusive relations.

[Theorem 4] Let h be a HOMT and h' be a U-HOMT with a quadruplet representation $(abs', dir', \sqsubseteq', rep')$ and a domain abstraction $abs': D_1 \rightarrow D_2$. Then, $h' \circ h$ is a safe HOMT if the following conditions are satisfied.

• If h is a covariant HOMT, $h(f \circ g) \subseteqq h(f) \circ h(g)$ and $h(f \sqcup g) \subseteqq h(f) \sqcup h(g)$. If h is a contravariant HOMT, $h(f \circ g) \subseteqq h(g) \circ h(f)$ and $h(f \sqcup g) \subseteqq h(f) \sqcup h(g)$.

• $\exists x \sqcup y$ for $x, y \in D_1$ implies that there exists $abs'(x) \sqcup abs'(y) \in D_2$ and $abs'(x \sqcup y) \sqsubseteq abs'(x) \sqcup abs'(y)$. Conversely, $\exists u \sqcup v$ for $u, v \in D_2$ implies $abs'^{-1}(u \sqcup v) \sqsubseteq_{EM} abs'^{-1}(u) \sqcup abs'^{-1}(v)$.

• A preorder $\sqsubseteq$ is cooperative with $\subseteqq$ (i. e. $\sqsubseteq \in \{\sqsubseteq_{-0}, \sqsubseteq_1, \subseteqq\}$).

4. SRAs as HOMTs

4. 1 SRAs on Flat Domains

The algorithms of various SRAs are independently proposed by many authors[1],[5],[8],[9],[11],[12],[15],[18],[20],[22]. These SRAs are formalized in terms of HOMTs by checking following two points: (1) an interpretation of primitive functions, (2) definedness ordering on abstract domains. Thus, an interpretation of general functions is constructed by the ordinary least fixed point calculus, and this corresponds to an algorithm of an SRA and a computed HOMT. We first shows SRAs on flat domains as HOMTs.

[Example 1] A quadruplet representation of FSA is $q_{FSA} = (abs_b, +, \sqsubseteq_1, \mathrm{Max})$. This is shown from the correspondence among interpretations on primitive functions and definedness ordering on abstract domains.

Let h_{FSA} be a HOMT whose quadruplet representation is q_{FSA}. Equivalence among interpretations on primitive functions is checked by testing all possible values on abstract domains. For example, by *FSA*, if (x, y, z) are interpreted to

$$\begin{aligned} if_{FSA}:\ & (\mathbf{1},\mathbf{1},\mathbf{1}) \rightarrow \mathbf{1}, \quad (\mathbf{1},\mathbf{1},\mathbf{0}) \rightarrow \mathbf{1}, \quad (\mathbf{1},\mathbf{0},\mathbf{1}) \rightarrow \mathbf{1}, \\ & (\mathbf{0},\mathbf{1},\mathbf{1}) \rightarrow \mathbf{0}, \quad (\mathbf{1},\mathbf{0},\mathbf{0}) \rightarrow \mathbf{0}, \quad (\mathbf{0},\mathbf{1},\mathbf{0}) \rightarrow \mathbf{0}, \\ & (\mathbf{0},\mathbf{0},\mathbf{1}) \rightarrow \mathbf{0}, \quad (\mathbf{0},\mathbf{0},\mathbf{0}) \rightarrow \mathbf{0}, \end{aligned}$$

and by h_{FSA},

$$\begin{aligned} h_{FSA}(if):\ & \mathrm{Max}(\{(\mathbf{1},\mathbf{1},\mathbf{1})\}) \rightarrow \mathrm{Max}(\{\mathbf{1}\}), \\ & \mathrm{Max}(\{(\mathbf{1},\mathbf{1},\mathbf{0})\}) \rightarrow \mathrm{Max}(\{\mathbf{1}\}), \\ & \mathrm{Max}(\{(\mathbf{1},\mathbf{0},\mathbf{1})\}) \rightarrow \mathrm{Max}(\{\mathbf{1}\}), \\ & \mathrm{Max}(\{(\mathbf{0},\mathbf{1},\mathbf{1})\}) \rightarrow \mathrm{Max}(\{\mathbf{0}\}), \\ & \mathrm{Max}(\{(\mathbf{1},\mathbf{0},\mathbf{0})\}) \rightarrow \mathrm{Max}(\{\mathbf{0}\}), \\ & \mathrm{Max}(\{(\mathbf{0},\mathbf{1},\mathbf{0})\}) \rightarrow \mathrm{Max}(\{\mathbf{0}\}), \\ & \mathrm{Max}(\{(\mathbf{0},\mathbf{0},\mathbf{1})\}) \rightarrow \mathrm{Max}(\{\mathbf{0}\}), \\ & \mathrm{Max}(\{(\mathbf{0},\mathbf{0},\mathbf{0})\}) \rightarrow \mathrm{Max}(\{\mathbf{0}\}). \end{aligned}$$

Thus, equivalence among if_{FSA} and $h_{FSA}(if)$ is easily shown from an embedding: $x \in \{\mathbf{0}, \mathbf{1}\} \rightarrow \{x\} \in PD(\{\mathbf{0}, \mathbf{1}\})$. Equivalence of definedness order is obvious from $x \sqsubseteq y \Rightarrow \{x\} \sqsubseteq_1 \{y\}$.

[Example 2] Strictness Information Analysis (SIA), the extension of BSA, is a HOMT h_{SIA} whose quadruplet representation is $(abs_b, -, \sqsubseteq_{-0}, \mathrm{Min})$. The main difference between BSA and SIA is that SIA can detect diverged functions whereas BSA cannot. This is because a totally undefined function $\Omega(x_1, \cdots, x_n)$ is interpreted to $\lambda x_1 \cdots x_n$. UNDEF by SIA with a special value UNDEF, whereas $\Omega(x_1, \cdots, x_n)$ is simply interpreted to $\lambda x_1 \cdots x_n.\ x_1 \cup \cdots \cup x_n$ (strict function) by BSA. Except $\Omega(x_1, \cdots, x_n)$, other primitive functions are interpreted to the equivalent abstract functions. For example, $if\ (x, y, z)$ and $+(x, y)$ are interpreted to

f	$h_{BSA}(f)$
$if\ (x, y, z)$	$\lambda xyz.(x \cup y) \cap (x \cup z)$
$+(x, y)$	$\lambda xy.\ x \cup y$

f	$h_{SIA}(f)$
$if\ (x, y, z)$	$\{\mathbf{1}\} \rightarrow \mathrm{Min}(\{(\mathbf{1},\mathbf{1},\mathbf{1}), (\mathbf{1},\mathbf{1},\mathbf{0}), (\mathbf{1},\mathbf{0},\mathbf{1})\})$
$+(x, y)$	$\{\mathbf{1}\} \rightarrow \mathrm{Min}(\{(\mathbf{1},\mathbf{1})\})$

Correspondence among definedness orderings is also easily checked from the fact $X \subseteq Y \Rightarrow X \sqsubseteq_{-0} Y$.

[Example 3] Computation Path Analysis (CPA) detects demand propagation patterns. CPA can not be treated as a single U-HOMT, but represented as a composition of two HOMTs. We will show it with the example of *if* (x, y, z) and *serial-or* (x, y, z), where *serial-or* (x, y, z) returns values true when $(x, y, z)=$ (true, $\perp$, $\perp$), (false, true, $\perp$), (false, false, true), and returns false only when (false, false, false). By definition, $if_{CPA}=\lambda xyz.\{\{x, y\}, \{x, z\}\}$ and $serial\text{-}or_{CPA}=\lambda xyz.\{\{x\}, \{x, y\}\}, \{x, y, z\}\}$(17).

The first step to get a HOMT h_{CPA} is the construction of a function inverse. This U-HOMT h_1 has a quadruplet representation $(id, -, \sqsubseteq_{-0}, \text{Min})$ such as

$$\begin{aligned}
h_1(if): \quad & \{5\} \to \{(\text{true}, 5, \perp), (\text{false}, \perp, 5)\}, \\
& \{4\} \to \{(\text{true}, 4, \perp), (\text{false}, \perp, 4)\}, \\
& \text{etc.} \\
h_1(serial\text{-}or): \quad & \{\text{true}\} \to \{(\text{true}, \perp, \perp), (\text{false}, \text{true}, \\
& \perp), (\text{false}, \text{false}, \text{true})\}, \\
& \{\text{false}\} \to \{(\text{false}, \text{false}, \text{false})\}. \\
h_1(\Omega): \quad & \{5\} \to \phi, \\
& \{\text{true}\} \to \phi, \text{ etc.}
\end{aligned}$$

The second step abstracts differences of values, but keeps the difference between an evaluated value and $\perp$. This U-HOMT h_2 has a quadruplet representation $(abs_b, +, \subseteq, Id)$. Then,

$$\begin{aligned}
h_2 \circ h_1(if) \quad & : \{\mathbf{1}\} \to \{(\mathbf{1}, \mathbf{1}, \mathbf{0}), (\mathbf{1}, \mathbf{0}, \mathbf{1})\}. \\
h_2 \circ h_1(serial\text{-}or) \quad & : \{\mathbf{1}\} \to \{(\mathbf{1}, \mathbf{0}, \mathbf{0}), (\mathbf{1}, \mathbf{1}, \mathbf{0}), \\
& \quad (\mathbf{1}, \mathbf{1}, \mathbf{1})\}. \\
h_2 \circ h_1(\Omega) \quad & : \{\mathbf{1}\} \to \phi
\end{aligned}$$

which exactly correspond to if_{CPA} and $serial\text{-}or_{CPA}$. Thus, CPA as a HOMT has a quadruplet representation $(abs_b, +, \subseteq, id) \circ (id, -, \sqsubseteq_{-0}, \text{Min})$(14).

4. 2 Tail/Total Strictness Detection

There are several SRAs on nonflat domains in literatures. They are tail/total/head strictness detection(15),(22) and an error detection based on minor signature analysis(11). We concentrate discussions on tail/total/head strictness detection. Formal definitions of these strictness are found in Refs. (15), (22).

Tail strictness is a property that the result is obtained only when input list data are evaluated until their spines are clarified. For instance,

$$\begin{aligned}
length(x) = & \text{if } null(x) \text{ then } 0 \\
& \text{else add1}(\text{length}(cdr(x)))
\end{aligned}$$

is tail-strict.

Total strictness is a property that the result is obtained only when input list data are completely evaluated. For instance,

$$\begin{aligned}
sum(x) = & \text{if } null(x) \text{ then } 0 \\
& \text{else add}(car(x), sum(cdr(x)))
\end{aligned}$$

is total-strict.

This section shows that tail/total strictness detection can be realized as extensions of FSA. (These are formalized as either forward or backward SAs. For a backward manner, the extensions of SIA with same domain abstractions below are appropriate.) In any case, the interests concentrated on finding adequate domain abstractions. For simplicity, we will restrict the arguments to strongly-typed first-order functional programs with integer streams instead of general lists. Note that the techniques are not exclusive of typeless programming except that classifications among flat/nonflat domains and Boolean values which decide the choices of conditional branches.

At first, we divide abs_b to abs_{int} and abs_{Bool} which are corresponding to integers and Booleans, respectively. abs_{int} is defined identically to abs_b except type-restrictions. abs_{Bool} is defined to be an identity map on Booleans. That is, an abstraction on Boolean is not necessary because Boolean domain is originally finite, and an abstraction on Boolean should be avoided because a Boolean value select a conditional branches which is crucial in SRAs on nonflat domains.

The following domain abstraction $abs_{list(int)}$ ignores differences among values, but keeps shapes of list data structures.

[Definition 10] The base domain abstraction abs_{int} : int $\to \{\mathbf{0}, \mathbf{1}\}$ on a flat domain *int* is extended inductively according to structure of domain D. For instance, the extension to *list*(*int*) is

$$abs_{list(int)}: x \to \begin{cases} \text{constructor}(abs_{int}(y), abs_{list(int)}(z)) \\ \qquad \text{if } x = \text{constructor}(y, z) \\ \mathbf{NIL} \quad \text{if } x = \text{Nil} \end{cases}$$

[Example 4] $abs_{list(int)}(list(int))$ is still an infinite domain, thus additional abstractions are required. They are,

$$\begin{aligned}
& abs_{tail/total}: list(int) \to \{\alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \\
& \qquad \alpha^{NIL}\}, \\
& abs_{total}: list(int) \to \{\beta^0, \beta^1, \beta^2, \beta^3, \beta^4, \beta^{NIL}\}, \\
& abs_{tail}: list(int) \to \{\gamma^0, \gamma^1, \gamma^2, \gamma^{NIL}\}, \\
& abs_{FSA}: list(int) \to \{\delta^0, \delta^1\}.
\end{aligned}$$

as shown in Fig. 3. In it an abstraction $abs: D \to Abs$ is regarded as a partition. (That is, $D = \bigcup_{x \in Abs} abs^{-1}(x)$, and $x \neq y \Longrightarrow abs^{-1}(x) \cap abs^{-1}(y) = \phi$.) For

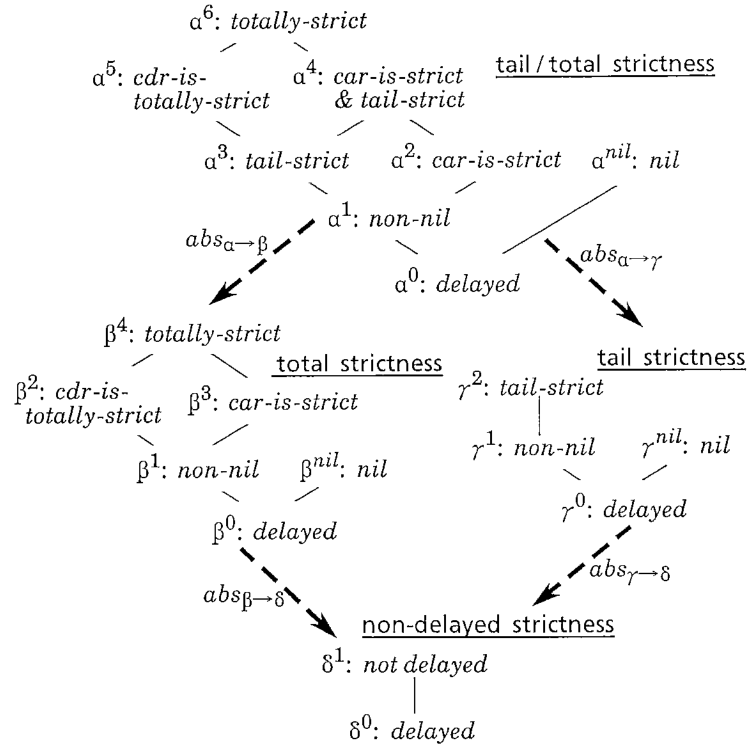


Fig. 3 A hierarchy of abstract domains for Tail/Total strictness detection.

instance, in tail/total strictness detection, α^*'s have meanings as follows.

- α^0 and α^{NIL} consists of a single element $\bot$ and NIL, respectively.
- α^1 collects the elements which are non-nil lists, but neither their car-parts nor their spines are evaluated.
- α^2 collects the elements in which their car-parts are evaluated, but spines are not clarified.
- α^3 collects the elements in which their spines are clarified, but car-parts and else remain unevaluated.
- α^4 collects the elements in which their car-parts and spines are evaluated.
- α^5 collects the elements in which every parts except car-parts are evaluated.
- α^6 consists of the elements which are completely evaluated.

They make a lattice and its ordering are represented by lines in Fig. 3. For β^*, γ^* and δ^* have similar meanings.

Then, Tail/Total-SA, which detects both tail/total strictness, has a quadruplet representation of $(abs_{tail/total}, +, \sqsubseteq_1, \mathrm{Max})$. Similarly, Total-SA, Tail-SA, and FSA have each quadruplet representations $(abs_{total}, +, \sqsubseteq_1, \mathrm{Max})$, $(abs_{tail}, +, \sqsubseteq_1, \mathrm{Max})$, and $(abs_{FSA}, +, \sqsubseteq_1, \mathrm{Max})$, respectively.

4. 3 Head Strictness Detection

Head strictness is a property that the result is obtained only when leaves (i. e. car-part) of input list data are evaluated synchronously with their evaluations on spines(This is equivalent to H-strictness in Ref. (4)). For instance,

$$\begin{aligned} search0(x) = \text{if } & null(x) \text{ then } 0 \\ & \text{else if } zerop(car(x)) \text{ then } 1 \\ & \qquad \text{else } search0(cdr(x)) \end{aligned}$$

is head-strict. Head strictness detection has been proposed as a projection analysis(22). However, the method based on abstract interpretation is unknown. We show that SAs are not enough for head strictness detection (which is indicated in Ref. (22)), but CPA with a non-monotonic domain abstraction abs_{head} can safely detect head-strictness. We call it head-CPA.

[Example 5] Let abs_{head} : $list(int) \rightarrow \{\varepsilon^{NIL}, \varepsilon^3, \varepsilon^2, \varepsilon^1, \varepsilon^0\}$ be as shown in Fig. 4. Regarding as ε^*'s as a partition, their meanings are,

- ε^0 and ε^{NIL} consist of a single element $\bot$ and Nil respectively.
- ε^1 collects the elements in which their cdr-parts are head-strict, but their car-parts remain unevaluated (including$(\bot\ .\ \bot)$).
- ε^2 collects the elements which are head-strict

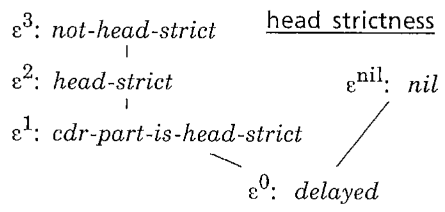


Fig. 4 Abstract domain for head-strictness detection.

(including (value. $\perp$).)

• ε^3 collects the rest of elements in non-nil lists.

Then, Head-CPA as a HOMT is defined with the quadruplet representation $(abs_{head}, +, \subseteqq, Id) \circ (id, -, \sqsubseteq_{-0}, \text{Min})$. In the algorithm of head-CPA, a path merging requires a $\sqcup$-operation. In a HOMT-framework, this is naturally introduced as $h(copy)$: $(y, z) \rightarrow y \sqcup z$ for a copying function copy : $x \rightarrow (x, x)$. For instance, in the definition of $search0(x)$, a variable x is copied three times in the right-hand side. We call them x^1, x^2, and x^3, respectively. Let us imagine the computation path which through the third branch in the definition in $search0(x)$. (i. e. $null(x)$ and $zerop(car(x))$ are both false.) If the property of x^1 is a non-nil list (cdr-part is head-strict), the property of x^2 is head-strict, and the property of x^3 is cdr-part is head-strict. These properties are merged to head-strict at the left-hand side occurrence of x. Note that if two properties makes conflicts to each other, this path will be erased. (i. e. maps to ϕ.) More precisely, $search0(x)$ is analyzed as follows. (Notations are same as in Sect. 3. 3) Since $search0(x)$ has a type $list(int) \rightarrow int$, the algorithm starts with $h(\Omega)$ which maps $\mathbf{1} \rightarrow \phi$. Let us note $h^{(i)}(f) = (h_\tau(\tau))^i(h(\Omega))$ (i. e. $\sqcup_i h^{(i)}(f) = h^c(f)$). Then, a convergent sequence of $h^c(search0)$ is

$$
\begin{aligned}
h^{(0)}(search0) :\quad & \mathbf{1} \rightarrow \phi\\
h^{(1)}(search0) :\quad & \mathbf{1} \rightarrow \{\varepsilon^{NIL}, \varepsilon^1 \sqcup \varepsilon^2\}\\
& = \{\varepsilon^{NIL}, \varepsilon^2\}\\
h^{(2)}(search0) :\quad & \mathbf{1} \rightarrow \{\varepsilon^{NIL}, \varepsilon^1 \sqcup \varepsilon^2, \varepsilon^1 \sqcup \varepsilon^2 \sqcup \varepsilon^1\}\\
& = \{\varepsilon^{NIL}, \varepsilon^2\} \text{ (Converged)}
\end{aligned}
$$

Thus, $h^c(search0)$ is computed as $\mathbf{1} \rightarrow \{\varepsilon^{NIL}, \varepsilon^2\}$. Therefore, $search0(x)$ is detected to be head-strict.

Note that Head-CPA is a more powerful SRA than those that proposed in Ref. (22), because Wadler's method has two major restrictions which head-CPA has not. That is,

• It works effectively only on primitive recursive functions.

• The objective language has two conditional expressions, *case* and *if*. It works effectively only on *case*-sentences which has quite severe restrictions, and works little on *if*-sentences which treat general conditional branches.

We imagine that SIA and FSA might work as $(abs_{head}, -, \sqsubseteq_{-0}, \text{Min})$ and $(abs_{head}, +, \sqsubseteq_1, \text{Max})$. However, abs_{head} is a nonmonotonic domain abstraction (i. e. not continuous). In fact, abs_{head} maps an ascending sequence $((\perp\perp . \perp)), ((\perp 2 . \perp))), ((12 . \perp))\cdots$ to $\varepsilon^3, \varepsilon^1, \varepsilon^2 \cdots$. Therefore, the conditions for safeness (in Theorem 2) are not satisfied. For instance,

$$
\begin{aligned}
& \textit{interval-search}0(x)\\
& = \text{if } null(x) \text{ then } 0\\
& \qquad \text{else if } zerop(car(x)) \text{ then } 1\\
& \qquad\qquad \text{else } \textit{interval-search}0(cdr(cdr(x)))
\end{aligned}
$$

is detected head-strict by both SIA and FSA, although this function is actually not head-strict.

5. Equivalence and Hierarchy among SRAs

5. 1 Forward/Backward Equivalence

The analytical powers of SRAs are compared by existence of transformational methods from one to the other.

[Definition 11] Let h_1 and h_2 be HOMTs. Then, $h_1 \leqq h_2$ *iff* $\exists h_{21}$: HOMT s. t. $h_1 = h_{21} \circ h_2$ Equivalence among HOMTs $h_1 \cong h_2$ is defined to be $h_1 \leqq h_2 \wedge h_2 \geqq h_1$.

[Example 6] The equivalence of FSA and SIA is proved by the existence of forward/backward conversion operators. These operators are easily found from reduction theorem (See Sect. 3. 2). That is,

$$h_{SIA} = \underline{(id, -, \sqsubseteq_{-0}, \text{Min})} \circ h_{FSA}$$

$$h_{FSA} = \underline{(id, -, \sqsubseteq_1, \text{Max})} \circ h_{SIA}$$

where $h_{FSA} = (abs_b, +, \sqsubseteq_1, \text{Max})$, and

$$h_{SIA} = (abs_b, -, \sqsubseteq_{-0}, \text{Min}).$$

5. 2 Appproximation Hierarchy

There are two cases that cause hierarchy of analytical power among SRAs: approximation hierarchy and property hierarchy, as mentioned in Sect. 2. 3. These are clarified by the existence of projection operators in terms of HOMTs.

The example is the relation among SAs on non-flat domains (e. g. streams), such as Tail/Total SA, Total SA, Tail SA, and FSA. Recall that their quadruplet representations as HOMTs are

$$h_{tail/total} = (abs_{tail/total}, +, \sqsubseteq_1, \text{Max}),$$

$$h_{tail} = (abs_{tail}, +, \sqsubseteq_1, \text{Max}),$$

$$h_{total} = (abs_{total}, +, \sqsubseteq_1, \text{Max}),$$

$$h_{FSA} = (abs_{FSA}, +, \sqsubseteq_1, \text{Max}).$$

Table 4 Additional abstractions for hierarchy among tail/total SAs.

$$
abs_{\alpha\to\beta}:\begin{cases}\alpha^{NIL} & \to \beta^{NIL}\\ \alpha^{6} & \to \beta^{4}\\ \alpha^{5} & \to \beta^{2}\\ \alpha^{4},\alpha^{2} & \to \beta^{3}\\ \alpha^{3},\alpha^{1} & \to \beta^{1}\\ \alpha^{0} & \to \beta^{0}\end{cases}
\qquad
abs_{\alpha\to\gamma}:\begin{cases}\alpha^{NIL} & \to \gamma^{NIL}\\ \alpha^{6},\alpha^{5},\alpha^{4},\alpha^{3} & \to \gamma^{2}\\ \alpha^{1},\alpha^{1} & \to \gamma^{1}\\ \alpha^{0} & \to \gamma^{0}\end{cases}
$$

$$
abs_{\beta\to\delta}:\begin{cases}\beta^{NIL},\beta^{4},\beta^{3},\beta^{2},\beta^{1} & \to \delta^{1}\\ \beta^{0} & \to \delta^{0}\end{cases}
\qquad
abs_{\gamma\to\delta}:\begin{cases}\gamma^{NIL},\gamma^{2},\gamma^{1} & \to \delta^{1}\\ \gamma^{0} & \to \delta^{0}\end{cases}
$$

Let abstraction maps be as Table 4. Note that they are all continuous functions. Then, from Theorem 1, following reduction rules are introduced. Thus, their approximation hierarchy is clarified as the existence of underlined quadruplet representations.

$$
\begin{aligned}
h_{total} &= (abs_{total},\ +,\ \sqsubseteq_1,\ \mathrm{Max})\\
&= \underline{(abs_{\alpha\to\beta},\ +,\ \sqsubseteq_1,\ \mathrm{Max})}\ \circ\\
&\quad (abs_{tail/total},\ +,\ \sqsubseteq_1,\ \mathrm{Max})\\
h_{tail} &= (abs_{tail},\ +,\ \sqsubseteq_1,\ \mathrm{Max})\\
&= \underline{(abs_{\alpha\to\gamma},\ +,\ \sqsubseteq_1,\ \mathrm{Max})}\ \circ\\
&\quad (abs_{tail/total},\ +,\ \sqsubseteq_1,\ \mathrm{Max})\\
h_{FSA} &= (abs_{FSA},\ +,\ \sqsubseteq_1,\ \mathrm{Max})\\
&= \underline{(abs_{\beta\to\delta},\ +,\ \sqsubseteq_1,\ \mathrm{Max})}\ \circ\\
&\quad (abs_{total},\ +,\ \sqsubseteq_1,\ \mathrm{Max})\\
&= \underline{(abs_{\gamma\to\delta},\ +,\ \sqsubseteq_1,\ \mathrm{Max})}\ \circ\\
&\quad (abs_{tail},\ +,\ \sqsubseteq_1,\ \mathrm{Max})
\end{aligned}
$$

5. 3 Property Hierarchy

The example of property hierarchy is the relation among CPA and SIA on flat domains.

[Example 7] A quadruplet representation of CPA is $(abs_b,\ +,\ \sqsubseteq_{set},\ id)\circ(id,\ -,\ \sqsubseteq_{-0},\ \mathrm{Min})$. From reduction theorem, SIA is induced from CPA as

$$
\begin{aligned}
&\underline{(id,\ +,\ \sqsubseteq_0,\ \mathrm{Min})}\ \circ h_{CPA}\\
&= (id,\ +,\ \sqsubseteq_0,\ \mathrm{Min})\circ((abs_b,\ +,\ \subseteqq,\ id)\ \circ\\
&\quad (id,\ -,\ \sqsubseteq_{-0},\ \mathrm{Min}))\\
&= ((id,\ +,\ \sqsubseteq_0,\ \mathrm{Min})\circ(abs_b,\ +,\ \subseteqq,\ id))\ \circ\\
&\quad (id,\ -,\ \sqsubseteq_{-0},\ \mathrm{Min})\\
&= (abs_b,\ +,\ \sqsubseteq_0,\ \mathrm{Min})\circ(id,\ -,\ \sqsubseteq_0,\ \mathrm{Min})\\
&= (abs_b,\ -,\ \sqsubseteq_0,\ \mathrm{Min})\\
&= h_{SIA}
\end{aligned}
$$

We imagine RA on flat domain can be obtained by a similar method. RA seems to have a corresponding HOMTs with a quadruplet representation $(ads_b,\ +,\ \sqsubseteq_1,\ \mathrm{Max})\circ(id,\ -,\ \sqsubseteq_{-0},\ \mathrm{Min})$, and to have a hierarchical relation with CPA as

$$
\begin{aligned}
&\underline{(id,\ +,\ \sqsubseteq_1,\ \mathrm{Max})}\ \circ h_{CPA}\\
&= (id,\ +,\ \sqsubseteq_1,\ \mathrm{Max})\circ((abs_b,\ +,\ \subseteqq,\ id)\ \circ\\
&\quad (id,\ -,\ \sqsubseteq_0,\ \mathrm{Min}))\\
&= ((id,\ +,\ \sqsubseteq_1,\ \mathrm{Max})\circ(abs_b,\ +,\ \subseteqq,\ id))\ \circ\\
&\quad (id,\ -,\ \sqsubseteq_0,\ \mathrm{Min})\\
&= (abs_b,\ +,\ \sqsubseteq_1,\ \mathrm{Max})\circ(id,\ -,\ \sqsubseteq_{-0},\ \mathrm{Min})\\
&= h_{RA}
\end{aligned}
$$

However, this formalization leads a trivial SRA. In fact, the initial function $h_{RA}(\Omega)$ is the greatest function with respect to $\sqsubseteq_1$, so that the result of this fixed-point computation $h^{c}_{RA}(f)$ for any recursive function f sticks to it.

6. Conclusion

A new formalization method for SRAs on first-order functional programs was proposed. For this purpose, the concept called HOMomorphic Transformer (HOMT) was introduced. Intuitively speaking, a HOMT is a special instance of abstract interpretation. A set of HOMTs is an algebraic space, where equivalence relations (or reduction rules) are defined. This paper has clarified that HOMTs can be used not only for a formalization of SRAs, but also as a transformational mechanism between these analyses. Thus, equivalent and hierarchical relationships among these analyses can be discussed on a unified basis.

The direction of further works is an automatic program generation of SRAs. This is partially done in Ref. (16). In fact, an experimental system which, is implemented on VAX Common Lisp (about 3. 5K lines), can produce total/tail-SA and head-CPA. However, their specifications require hand-coded data tables. As automatic program generation of SRAs still remain in naive status.

Acknowledgements

The authors would like to thank their colleagues for their useful discussions and encouragements.

References

(1) eds. Abramsky S. and Hankin C.: "Abstract interpretation of declarative languages", Ellis Horwood Limited (1987).

(2) Barendregt H. P.: "The Lambda Calculus, Its Syntax and Semantics", North Holland (1981).

(3) Bloss B. and Hudak P.: "Variations on strictness analysis", ACM Conf. on LISP and Functional Programming, pp. 132-142 (1986).

(4) Burn G. L.: "A Relationship Between Abstract Interpretation and Projection Analysis (Extended Abstract)", 17 th ACM POPL, pp. 151-156 (1990).

(5) Clack C. and Peyton Jones S. L.: "Strictness analysis—a practical approach", Functional Programming Languages and Computer Architecture, LNCS, **201**, pp. 35-49, Springer-Verlag (1985).

(6) Cousot P. and Cousot R.: "Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints", 4 th ACM POPL, pp. 238-252 (1977).

(7) Dybjer P.: "Inverse Image Analysis", 4 th ICALP, LNCS **267**, pp. 21-30, Springer-Verlag (1987).

(8) Hankin C. L., Burn G. L. and Peyton Jones S. L.: "A Safe Approach to Parallel Combinator Reduction", Theoretical Computer Science, **56**, pp. 17-36 (1988).

(9) Hudak P. and Young R.: "Higher-order strictness analysis in untyped lambda calculus", 13 th ACM POPL, pp. 97-109 (1986).

(10) Hughes J.: "Strictness Detection in Non-Flat Domains", LNCS, **217**, pp. 112-135, Springer-Verlag (1985).

(11) Mishra P. and Keller A. M.: "Static Inference of Properties of Applicative Programs", 11 th ACM POPL, pp. 235-244 (1984).

(12) Mycroft A.: "The theory and practice of transforming call-by-need into call-by-value", LNCS, **83**, pp. 269-281, Springer-Verlag (1980).

(13) Nielson F.: "A Bibliography on abstract interpretation", ACM SIGPLAN Notices, **21**, 5, pp. 31-38 (1986).

(14) Ogawa M. and Ono S.: "Transformation of Strictness-Related Analyses based on Abstract Interpretation", Proc. Int. Conf. Fifth Generation Computer Systems 1988, pp. 430-438 (1988).

(15) Ogawa M. and Ono S.: "Detecting non-monotonic properties in functional programs", COMP89-107, IEICE, pp. 1-7 (Feb. 1990).

(16) Ono S., Ogawa M. and Tsuruoka Y.: "Deriving Inductive Properties of Recursive Programs based on Least-fixpoint Computation", Proc. JSPP'90, IPSJ, pp. 249-256 (May 1990).

(17) Ono S.: "Computation Path Analysis: Towards an Autonomous Global Dataflow Analysis", The Second France-Japan Artificial Intelligence and Computer Science Symposium, Sophia, France (1987).

(18) Ono S. and Takahashi N.: "Algorithm for computing dependency property sets in recursive function systems", Trans. IECE Japan, **J69-D**, 5, pp. 714-723 (1986).

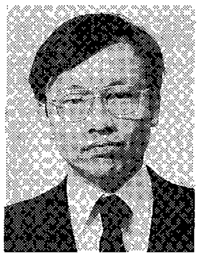
(19) Ono S., Takahashi N. and Amamiya M.: "Optimized demand-driven evaluation of functional programs on a dataflow machine", IEEE ICPP'86, pp. 421-428 (1986).

(20) Peyton Jones S. L.: "The implementation of functional programming languages", Prentice-Hall (1987).

(21) Smyth M. B.: "Power Domains", JCSS **16**, pp. 23-36 (1978).

(22) Wadler P. and Hughes R. J. M.: "Projections for Strictness Analysis", Functional Programming Languages and Computer Architecture, LNCS, **274**, pp. 385 - 407, Springer-Verlag (1987).

Mizuhito Ogawa was born in Yokohama, Japan, in 1960. He received B. Sci. and M. Sci. degrees in Mathematics both from The University of Tokyo, Tokyo, Japan, in 1983 and 1985, respectively. From 1985, he was a researcher at NTT Musashino Electric Communication Laboratories. Currently, he is a researcher at NTT Basic Research Laboratories. His research interests include functional programming, term rewriting systems, computation, verification, and theorem proving. He is a member of ACM, IEEE and IPSJ.



Satoshi Ono was born in Sapporo, Japan, in 1954. He received a B. Eng. degree in electronic engineering in 1977, and received M. Eng. and Ph. D. degrees in electrical engineering from The University of Tokyo, Tokyo, Japan, in 1979, and 1982, respectively. From 1982, he was a researcher at NTT Musashino Electric Communication Laboratories. He has been working on parallel processing, parallel programming, and CASE. Currently, he is a senior research engineer at NTT Software Laboratories. His research interests include formal specification, analyses on realtime, parallel and distributed programs, and their applications to CASE. He is a member of ACM, IEEE and IPSJ.