INFORMATION AND COMPUTATION 76, 29–92 (1988)

Strictness Analysis and Denotational Abstract Interpretation

FLEMMING NIELSON

Department of Computer Science, The Technical University of Denmark, DTH Building 344, DK-2800 Lyngby, Denmark

A theory of abstract interpretation (P. Cousot and R. Cousot, *in* "Conf. Record, 4th ACM Symposium on Principles of Programming Languages," 1977) is developed for a typed λ-calculus. The typed λ-calculus may be viewed as the "static" part of a two-level denotational metalanguage for which abstract interpretation was developed by F. Nielson (Ph.D. thesis, University of Edinburgh, 1984; *in* "Proceedings, STACS 1986," Lecture Notes in Computer Science, Vol. 210, Springer-Verlag, New York/Berlin, 1986). The present development relaxes a condition imposed there and this sufices to make the framework applicable to strictness analysis for the λ-calculus. This shows the possibility of a general theory for the analysis of functional programs and it gives more insight into the relative precision of the various analyses. In particular it is shown that a collecting (static; P. Cousot and R. Cousot, *in* "Conf. Record, 6th ACM Symposium on Principles of Programming Languages," 1979) semantics exists, thus answering a problem left open by G. L. Burn, C. L. Hankin and S. Abramsky (*Sci. Comput. Programming* 7 (1986), 249–278). © 1988 Academic Press, Inc.

1. INTRODUCTION

Functional programming languages are becoming more and more popular, especially in the so-called lazy variants. In particular, a lazy language has the default parameter mechanism call-by-name (or call-by-need) rather than call-by-value. In order to implement these languages efficiently many researchers have studied sufficient conditions for safe replacement of call-by-name by call-by-value. This is of interest in implementing the languages on parallel architectures because arguments in call-by-value positions of some function may safely be evaluated in parallel. But even on sequential architectures it is beneficial to avoid the overhead of call-by-name when call-by-value would suffice.

In a purely functional language the only difference between call-by-name and call-by-value is that the latter may lead to non-termination when the former does not. So in a call

$$f(\cdots)$$

the parameter mechanism of f may be changed to call-by-value if non-termination of $\cdots$ implies non-termination of $f(\cdots)$. In the notation of denotational semantics this is written

$$f(\bot)=\bot,$$

i.e., that f is *strict*. It is generally undecidable whether or not f is strict but one may attempt to find a safe approximation that is decidable. This entails working with a safe representation of $\bot$, i.e., of when a computation definitely will not terminate, and for historical reasons this is written as the number 0. We therefore search for a recursive function f' such that $f'(0)=0$ implies that $f(\bot)=\bot$. A number of such *strictness analyses* are considered in Mycroft (1981), Mycroft and Nielson (1983), Burn *et al.* (1986), Abramsky (1985), Hughes (1986), and Maurer (1986). In the absence of a general theory the correctness of strictness analysis must be shown for each functional language considered. Equally important, for a fixed language there are many different analyses one wants to perform and these, too, must be shown to be correct one by one.

A similar situation prevailed a decade ago for analyses of imperative languages. To overcome the inconvenience of a multitude of correctness proofs a theory of *abstract interpretation* was developed (Cousot and Cousot, 1979) for flowchart languages. Subsequent research has studied how to extend the framework to models of programs other than the flowcharts. The extension to (first-order) recursion equation schemes has been performed in, e.g., Mycroft and Nielson (1983) and Jones and Mycroft (1986). The extension to a wide class of denotational definitions was performed in Nielson (1986a, 1984). The latter work defined a two-level metalanguage TMLs and developed abstract interpretation for all definitions in this metalanguage. Thereby the development is applicable to all programming languages definable in TMLs and this includes PASCAL-like languages. This is in contrast to most developments of static analyses of programs (e.g., Barbuti and Martelli, 1983; Nielson, 1982) where only a small toy language is considered. The benefit of working with some form of denotational definitions is that denotational semantics is one of the most general semantics frameworks available. However, some syntactic limitations in TMLs preclude a general treatment of functional languages.

In this paper we consider a smaller metalanguage, TMLb. Its syntax and semantics are presented in Section 3 whereas Section 2 surveys the necessary domain theory. In Section 4 we develop abstract interpretation for TMLb. This includes the study of *correctness* of analyses (w.r.t. the semantics), *safety* of analyses w.r.t. other analyses, and the *specification of best analyses* (starting from other analyses or the semantics). In Section 5 we apply this theory to examples studied in the literature; in particular we give a strictness analysis for the typed λ-calculus of Burn *et al.* (1986). With respect to Nielson (1986a, 1984) we have lifted a restriction imposed there and this suffices for a class of "independent attribute analyses" (Jones and Muchnick, 1981) for functional languages (as opposed to the "relational analyses" (Jones and Muchnick, 1981) also considered in Nielson (1986a, 1984) but for imperative languages only). Another main difference from Nielson (1986a, 1984) is that the development does not build on the existence of the so-called collecting semantics (static semantics; Cousot and Cousot, 1979). Therefore in Section 6 we show that the collecting semantics does exist and that correctness of an analysis is equivalent to safety of that analysis with respect to the collecting semantics. (This also answers a problem left open in Burn *et al.* (1986).) Finally, in Section 7 we present the main conclusions.

An extended abstract of this paper appeared previously as Nielson (1987) and some of the tedious proofs not given here may be found in Nielson (1986c).

2. Preliminaries

In this section we review the simpler parts of the theoretical foundations of denotational semantics. This will suffice for the subsequent development because the denotational meta-language introduced in the next section does not contain recursive domain equations.

A *partially ordered set* is a pair $(D, \sqsubseteq)$ where D is a set and $\sqsubseteq$ is a partial order, i.e., a binary relation over D that is reflexive (i.e., $d \sqsubseteq d$), transitive (i.e., $d \sqsubseteq e \wedge e \sqsubseteq f \Rightarrow d \sqsubseteq f$) and anti-symmetric (i.e., $d \sqsubseteq e \wedge e \sqsubseteq d \Rightarrow d = e$). An element $d \in D$ is an *upper bound* of a subset S of D iff $s \sqsubseteq d$ holds for all elements s of S. An element $d \in D$ is a *least upper bound* of S iff d is an upper bound of S and $d \sqsubseteq d'$ holds whenever d' is an upper bound of S. A subset S of D need not have a least upper bound but if it does the least upper bound is unique and is written $\bigsqcup S$. A partially ordered set is a *complete lattice* iff every subset has a least upper bound. A subset S of D is a *chain* iff $s \sqsubseteq s'$ or $s' \sqsubseteq s$ whenever s and s' are elements of S. We may define a complete partial order, or a *cpo*, to be a partially ordered set where all chains have least upper bounds. Every complete lattice is a cpo but not vice versa. An element d_0 of D is *least* iff $d_0 \sqsubseteq d$ holds for all d. A least element need not exist but if it does it is unique and is written $\perp_D$ or $\perp$. In a cpo (and hence also in a complete lattice) a least element always exists and is given by the formula $\perp = \bigsqcup \varnothing$, where $\varnothing$ is the empty set.

A function $f\colon (D, \sqsubseteq) \rightarrow (D', \sqsubseteq')$ from a partially ordered set $(D, \sqsubseteq)$ into a partially ordered set $(D', \sqsubseteq')$ is a function $f\colon D \rightarrow D'$ from D to D'. It is *monotonic* iff $d_1 \sqsubseteq d_2$ implies $f(d_1) \sqsubseteq' f(d_2)$. It is *continuous* iff for every *non-empty* chain $S \subseteq D$ with a least upper bound also $\{f(s) \mid s \in S\} \subseteq D'$ is a chain and it has $f(\bigsqcup S)$ as a least upper bound. Every continuous function is monotonic but not vice versa. However, when f is monotonic and $S \subseteq D$ is a chain then $\{f(s) \mid s \in S\}$ is always a chain; furthermore, if $\bigsqcup S$ and $\bigsqcup \{f(s) \mid s \in S\}$ both exist then $\bigsqcup \{f(s) \mid s \in S\} \sqsubseteq' f(\bigsqcup S)$. The function f is *strict* iff whenever D has a least element $\perp$ then D' has a least element $\perp'$ and $f(\perp) = \perp'$. It is *completely additive* iff for all subsets $S \subseteq D$ with a least upper bound, $\{f(s) \mid s \in S\} \subseteq D'$ also has a least upper bound and it is $f(\bigsqcup S)$. Every completely additive function is strict and continuous but not vice versa.

A *fixed point* of a function $f\colon (D, \sqsubseteq) \to (D, \sqsubseteq)$ is an element d of D such that $f(d) = d$. A *least fixed point* of f is an element d that is a fixed point and satisfies $d \sqsubseteq d'$ whenever d' is a fixed point of f. A least fixed point need not exist but if it does it is unique and is written $\mathrm{fix}(f)$. The existence of certain least fixed points is of vital importance for the treatment of iteration and recursion in denotational semantics.

FACT 1. *If $(D, \sqsubseteq)$ is a* cpo *and $f\colon (D, \sqsubseteq) \to (D, \sqsubseteq)$ is monotonic then it has a least fixed point.*

For a proof one may construct $\mathrm{fix}(f)$ by transfinite induction (Halmos, 1960) but we shall omit the details.

A predicate Q upon a cpo $(D, \sqsubseteq)$ is a predicate upon D. It is *admissible* iff for all chains $S \subseteq D$ we have that Q holds upon $\bigsqcup S$ whenever Q holds upon each element of S. A related concept is that of a sub-cpo. A cpo $(D', \sqsubseteq')$ is a *sub-cpo* of the cpo $(D, \sqsubseteq)$ iff D' is a subset of D and $\sqsubseteq'$ is the restriction of $\sqsubseteq$ to $D' \times D'$ and for every chain $S \subseteq D' \subseteq D$ that the least upper bound in D' equals that in D (i.e., $\bigsqcup' S = \bigsqcup S$). Whenever $(D, \sqsubseteq)$ is a cpo and Q is an admissible predicate one may define a sub-cpo $(D', \sqsubseteq')$ by putting $D' = \{d \in D \mid Q(d)\}$ and letting $\sqsubseteq'$ be the restriction of $\sqsubseteq$ to $D' \times D'$. Admissible predicates are important because they can be used to infer properties about least fixed points.

FACT 2 ("Scott-induction"). *If Q is an admissible predicate upon the cpo $(D, \sqsubseteq)$ and $f\colon (D, \sqsubseteq) \to (D, \sqsubseteq)$ is monotonic then*

$$Q(\mathrm{fix}(f))$$

follows if $Q(d) \Rightarrow Q(f(d))$ holds for all $d \in D$.

We omit the proof, which is by transfinite induction.

It is useful to be able to construct cpo's and to have operations upon the elements. If S is a set we define the *flat cpo* $S_\perp$ to be $(D, \sqsubseteq)$, where

$$D = \{(0, 0)\} \cup (\{1\} \times S)$$

$$d_1 \sqsubseteq d_2 \quad \text{iff} \quad d_1 = (0, 0) \quad \text{or} \quad d_1 = d_2.$$

So (0, 0) plays the role of $\bot$ and D essentially is the disjoint union of S and $\{\bot\}$. When $(D_i, \sqsubseteq_i)$ are cpo's the *separated sum* of $(D_1, \sqsubseteq_1), \ldots, (D_k, \sqsubseteq_k)$ is the cpo $(D, \sqsubseteq)$ defined by

$$D = \{(0,0)\} \cup (\{1\} \times D_1) \cup \cdots \cup (\{k\} \times D_k)$$

$$\begin{aligned} d \sqsubseteq d' \quad & \text{iff} \quad d = (0,0) \\ & \text{or for some } i,\ d_i,\ d_i' \text{ that} \\ & d = (i, d_i) \wedge d' = (i, d_i') \wedge d_i \sqsubseteq_i d_i'. \end{aligned}$$

So D is the disjoint union of $D_1, \ldots, D_k$ and a new least element. We write

$$\text{in}_i(d) = (i, d)$$

$$\text{is}_i(d) = \begin{cases} \bot & \text{if } d = \bot \\ \text{true} & \text{if } d = (i, d') \text{ for some } d' \\ \text{false} & \text{if } d = (j, d') \text{ for some } d' \text{ and } j \neq i \end{cases}$$

$$\text{out}_i(d) = \begin{cases} d' & \text{if } d = (i, d') \text{ for some } d' \\ \bot & \text{otherwise.} \end{cases}$$

In practice we identify D_i with $(D_i, \sqsubseteq_i)$ and write $D_1 + \cdots + D_k$ for the sum. Also, we shall no longer clearly distinguish between the various partial orders so we just write $\sqsubseteq$ for the partial order in question.

The *cartesian product* $D_1 \times \cdots \times D_k$ of the $k \geqslant 1$ cpo's $(D_1, \sqsubseteq), \ldots, (D_k, \sqsubseteq)$ is the cpo $(D, \sqsubseteq)$, where

$$D = \{(d_1, \ldots, d_k) \mid d_1 \in D_1, \ldots, d_k \in D_k\}$$

$$(d_1, \ldots, d_k) \sqsubseteq (d_1', \ldots, d_k') \quad \text{iff} \quad d_1 \sqsubseteq d_1' \wedge \cdots \wedge d_k \sqsubseteq d_k'.$$

We write

$$(d_1, \ldots, d_k) \downarrow i = d_i$$

and one may check that

$$\bigsqcup S = \left(\bigsqcup \{s \downarrow 1 \mid s \in S\}, \ldots, \bigsqcup \{s \downarrow k \mid s \in S\}\right)$$

whenever $S \subseteq D_1 \times \cdots \times D_k$ is a chain. The (monotonic) *function space* $D' \to D''$ *of the* cpo's $(D', \sqsubseteq)$ and $(D'', \sqsubseteq)$ is the cpo $(D, \sqsubseteq)$, where

$$D = \{f : D' \to D'' \mid f \text{ is monotonic}\}$$

$$f_1 \sqsubseteq f_2 \quad \text{iff for all } d' \in D' \quad \text{that} \quad f_1(d') \sqsubseteq f_2(d').$$

One may check that

$$\left(\bigsqcup S\right) = \lambda d' \cdot \bigsqcup \{f(d') \mid f \in S\}$$

whenever $S \subseteq D$ is a chain. We shall explain later why we prefer not to require functions to be continuous. Finally, the equations

$$\bot \to d_1, d_2 = \bot$$
$$\text{true} \to d_1, d_2 = d_1$$
$$\text{false} \to d_1, d_2 = d_2$$

define a conditional construct $d \to d_1, d_2$.

3. The Metalanguage

The development of the present paper builds on language definitions in the style of denotational semantics (Stoy, 1977; Gordon, 1979; Milne and Strachey, 1976). A denotational language definition takes the form of semantic equations that in a syntax-directed way define a mapping from programs into denotations (or meanings). The semantics or semantic function of a language is then taken to be that mapping. The notation used for constructing denotations is called a metalanguage and is often studied in its own right. The motivation behind this is that the semantics may actually be split into two mappings: a mapping from programs into the metalanguage and a mapping (called an interpretation) from the metalanguage into denotations. This is illustrated by the top and bottom halves of Fig. 1. One advantage of this approach is that it becomes possible to construct a system that interprets denotational definitions (e.g., Mosses, 1979). Another advantage is that one may obtain different semantic functions from the same semantic equations: One semantic function (the standard semantics (Milne and Strachey, 1976)) might describe the usual input–output behaviour of programs whereas another semantic function might express some analysis of programs, e.g., strictness analysis. This is illustrated in Fig. 1.

Syntax

We shall consider the metalanguage TMLb whose types are given by

$$ct ::= A_i \mid \mathbf{B}_i \mid ct \times ct \mid ct + ct \mid ct \to ct \qquad (\text{for} \quad i \in I).$$

Clearly $\times$ means cartesian product, $+$ means separated sum, and $\to$

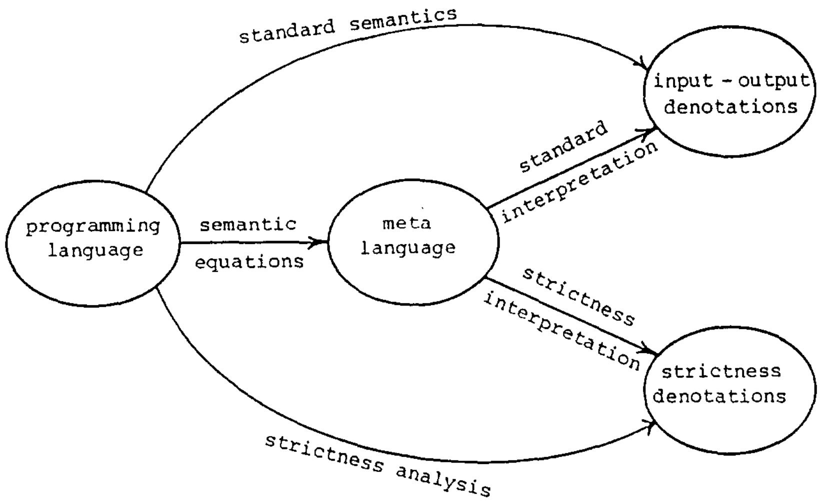


FIG. 1. The role of the metalanguage.

means monotonic function space. The difference between A_i and $\mathbf{B}_i$ will become clearer as we proceed. Here it suffices to say that entities of type A_i will mean the same in all interpretations whereas the meanings of entities of type $\mathbf{B}_i$ may be different in different interpretations. Another way to put this is to say that the A_i correspond to concrete datatypes whereas the $\mathbf{B}_i$ correspond to abstract datatypes. The index set I will mostly be assumed to be the positive integers but other sets could be used, e.g., the language of some context-free grammar.

The expressions of TMLb are fairly conventional. The context-free syntax is given by

$e ::= f_i$ (for $i \in I$)	constants
$\mid (e, e')$	constructing a tuple
$\mid e \downarrow i$	selecting a component
$\mid \mathrm{in}_i\, e$	injection into a sum
$\mid \mathrm{is}_i\, e$	testing the tag field
$\mid \mathrm{out}_i\, e$	projecting out of a sum
$\mid \lambda x\colon ct \cdot e$	function abstraction
$\mid e(e')$	function application
$\mid x$	variables
$\mid \mathrm{fix}_{ct}\, e$	least fixed points
$\mid e \rightarrow e', e''$	conditional (over the "concrete" truth values)

The metalanguage is typed, which means that certain type constraints must be satisfied. For this let a type environment tenv be a finitary mapping from variables into types, i.e., a partial function from variables to types that is only defined on a finite set dom(tenv) of variables. The well-typing relation tenv $\vdash$ e: ct then states that e is well-typed with type ct assuming that the free variables of e are in dom(tenv) and have types as specified by tenv. To be precise one can give the following inductive definition:

$\text{tenv} \vdash f_i : ct_i$ for fixed types associated with f_i

$$\frac{\text{tenv} \vdash e : ct \quad \text{tenv} \vdash e' : ct'}{\text{tenv} \vdash (e, e') : ct \times ct'}$$

$$\frac{\text{tenv} \vdash e : ct_1 \times ct_2}{\text{tenv} \vdash e \downarrow i : ct_i}$$

$$\frac{\text{tenv} \vdash e : ct_i}{\text{tenv} \vdash \text{in}_i\, e : ct_1 + ct_2}$$

$$\frac{\text{tenv} \vdash e : ct_1 + ct_2}{\text{tenv} \vdash \text{is}_i\, e : T},$$ where T is some A_i (e.g., A_1) that is viewed as being the truth values

$$\frac{\text{tenv} \vdash e : ct_1 + ct_2}{\text{tenv} \vdash \text{out}_i\, e : ct_i}$$

$$\frac{\text{tenv}[ct/x] \vdash e : ct'}{\text{tenv} \vdash \lambda x : ct \cdot e : ct \to ct'},$$ where tenv$[ct/x]$ is the type environment obtained from tenv by letting x be mapped to ct

$$\frac{\text{tenv} \vdash e : ct \to ct' \quad \text{tenv} \vdash e' : ct}{\text{tenv} \vdash e(e') : ct'}$$

$\text{tenv} \vdash x : ct$ whenever $\text{tenv}(x) = ct$

$$\frac{\text{tenv} \vdash e : ct \to ct}{\text{tenv} \vdash \text{fix}_{ct}\, e : ct},$$ where one should note that ct in fix_{ct} is the type of the result

$$\frac{\text{tenv} \vdash e : T \quad \text{tenv} \vdash e' : ct \quad \text{tenv} \vdash e'' : ct}{\text{tenv} \vdash e \to e', e'' : ct}$$

We shall write $\varnothing$ for the empty type environment. An expression e (as generated by the context-free syntax) is closed if $\varnothing \vdash e : ct$ holds for some ct.

Interpretations

We shall interpret types as cpo's and closed expressions as elements of cpo's. By a *type interpretation* we shall mean a structure that assigns a cpo to each $\mathbf{B}_i$. The semantics of types then is defined relative to a type interpretation. So given a type interpretation $\mathbf{I}$ we define the cpo $\mathbf{I}[\![ct]\!]$ as follows:

$$\begin{aligned}
\mathbf{I}[\![A_i]\!] &= \text{some fixed cpo } A_i \\
\mathbf{I}[\![\mathbf{B}_i]\!] &= \mathbf{I}(\mathbf{B}_i) \\
\mathbf{I}[\![ct \times ct']\!] &= \mathbf{I}[\![ct]\!] \times \mathbf{I}[\![ct']\!] && \text{(cartesian product)} \\
\mathbf{I}[\![ct + ct']\!] &= \mathbf{I}[\![ct]\!] + \mathbf{I}[\![ct']\!] && \text{(separated sum)} \\
\mathbf{I}[\![ct \to ct']\!] &= \mathbf{I}[\![ct]\!] \to \mathbf{I}[\![ct']\!] && \text{(monotonic function space).}
\end{aligned}$$

We shall not specify the A_i further but it is natural to assume that A_1 is the flat cpo $\{\text{true, false}\}_\perp$ of truth values.

By an *interpretation* $\mathbf{I}$ we shall mean a type interpretation (also denoted $\mathbf{I}$) together with a structure that assigns an element of $\mathbf{I}[\![ct_i]\!]$ to each constant f_i (of type ct_i). The semantics of an expression e is of course relative to an interpretation $\mathbf{I}$. It is also relative to a type environment tenv and a type ct such that tenv $\vdash e\colon ct$. So if dom(tenv) is the set $\{x_1, ..., x_k\}$ and tenv$(x_i) = ct_i$ we shall define a semantic function

$$\mathbf{I}[\![e]\!]_{(\text{tenv},ct)}\colon \mathbf{I}[\![ct_1]\!] \times \cdots \times \mathbf{I}[\![ct_k]\!] \to \mathbf{I}[\![ct]\!]$$

structurally upon e. However, it complicates the notation to be precise about tenv and ct so for the sake of readability we shall omit the (tenv, ct) index. We then have the equations

$$\begin{aligned}
\mathbf{I}[\![f_i]\!] &= \lambda(v_1, ..., v_k)\cdot \mathbf{I}(f_i) \\
\mathbf{I}[\![(e, e')]\!] &= \lambda(v_1, ..., v_k)\cdot (\mathbf{I}[\![e]\!](v_1, ..., v_k), \mathbf{I}[\![e']\!](v_1, ..., v_k)) \\
\mathbf{I}[\![e\downarrow i]\!] &= \lambda(v_1, ..., v_k)\cdot (\mathbf{I}[\![e]\!](v_1, ..., v_k))\downarrow i \\
\mathbf{I}[\![\text{in}_i e]\!] &= \lambda(v_1, ..., v_k)\cdot \text{in}_i(\mathbf{I}[\![e]\!](v_1, ..., v_k)) \\
\mathbf{I}[\![\text{is}_i\, e]\!] &= \lambda(v_1, ..., v_k)\cdot \text{is}_i(\mathbf{I}[\![e]\!](v_1, ..., v_k)) \\
\mathbf{I}[\![\text{out}_i\, e]\!] &= \lambda(v_1, ..., v_k)\cdot \text{out}_i(\mathbf{I}[\![e]\!](v_1, ..., v_k)) \\
\mathbf{I}[\![\lambda x_{k+1}\colon ct\cdot e]\!] &= \lambda(v_1, ..., v_k)\cdot \lambda v\cdot \mathbf{I}[\![e]\!](v_1, ..., v_k, v) \\
\mathbf{I}[\![e(e')]\!] &= \lambda(v_1, ..., v_k)\cdot (\mathbf{I}[\![e]\!](v_1, ..., v_k))(\mathbf{I}[\![e']\!](v_1, ..., v_k)) \\
\mathbf{I}[\![x_i]\!] &= \lambda(v_1, ..., v_k)\cdot v_i \\
\mathbf{I}[\![\text{fix}_{ct} e]\!] &= \lambda(v_1, ..., v_k)\cdot \text{fix}(\mathbf{I}[\![e]\!](v_1, ..., v_k)) \\
\mathbf{I}[\![e \to e', e'']\!] &= \lambda(v_1, ..., v_k)\cdot \mathbf{I}[\![e]\!](v_1, ..., v_k) \to \mathbf{I}[\![e']\!](v_1, ..., v_k), \\
&\quad\ \mathbf{I}[\![e'']\!](v_1, ..., v_k).
\end{aligned}$$

It should be clear that, e.g., the in_i occurring in $[\![\text{in}_i e]\!]$ is a syntactic construct of TMLb whereas the in_i on the right-hand side of the equation is the function defined in Section 2. It is not difficult to show by structural induction that the above equations define a monotonic function of the stated functionality.

EXAMPLE. The usual input–output semantics is obtained by specifying the standard interpretation **S**. We shall define $\mathbf{S}(\mathbf{B}_i) = A_i$ and only when we come to the applications shall we provide further information/assumptions about the A_i and hence the $\mathbf{S}(\mathbf{B}_i)$. Similarly we shall assume that the $\mathbf{S}(f_i)$ are fixed as elements of $\mathbf{S}[\![ct_i]\!]$ and we shall be more specific when we come to the applications. However, it is instructive to point out that a natural candidate for an f_i might be the "abstract" conditional

$$f_{ct}\colon \mathbf{B}_1 \times ct \times ct \to ct$$

for all choices of types ct. In the input–output semantics we will then have

$$\mathbf{S}(f_{ct}) = \lambda(v_1, v_2, v_3) \cdot v_1 \to v_2, v_3,$$

assuming that $\mathbf{B}_1$ is the "abstract" type of truth values. This is the version of conditional to be used when we want to *analyse* a program without knowing the exact values of variables. By contrast the conditional $\cdots \to \cdots, \cdots$ should be used for those aspects (e.g., static well-formedness) of a program that are the same in all interpretations.

It is convenient to name a special kind of interpretations. By a *lattice type interpretation* we shall mean a type interpretation that specifies a complete lattice, and not just a cpo, for each $\mathbf{B}_i$. We define a *lattice interpretation* similarly. Not all interpretations are lattice interpretations, and the standard interpretation **S** is an example because $\mathbf{S}(\mathbf{B}_1) = A_1 = \{\text{true}, \text{false}\}_\perp$ is not a complete lattice. The interest in lattice interpretations arises when one considers program analyses (abstract interpretations or data flow analyses). This is because a complete lattice is a cpo that has least upper bounds of two-element sets (Markowsky, 1976): the interest in cpo's is inherent in denotational semantics and the interest in least upper bounds of two-element sets arises when one wants to combine the effects along the "then" and "else" branches of an "abstract" conditional (the f_{ct} of the previous example).

Relations between Interpretations

In considering more than one interpretation of the metalanguage it is important to be able to relate their effects. So let **I** and **J** be (type) interpretations and let $Q = (Q_i)_i$ be a family of admissible relations (or predicates)

$$Q_i\colon \mathbf{I}(\mathbf{B}_i) \times \mathbf{J}(\mathbf{B}_i) \to \{\text{true}, \text{false}\}.$$

The intention with Q_i is that $Q_i(u, v)$ holds when $u \in \mathbf{I}(\mathbf{B}_i)$ and $v \in \mathbf{J}(\mathbf{B}_i)$ are related as desired, e.g., when the property v correctly describes the actual value u. Given $(Q_i)_i$ we can use the idea of a relational functor (Reynolds, 1974) or logical relation (Plotkin, 1973) to extend the relationship to all types, i.e., to obtain a relation

$$\text{sim}_{ct}[Q]: \mathbf{I}[\![ct]\!] \times \mathbf{J}[\![ct]\!] \to \{\text{true}, \text{false}\}$$

for all types ct.

This relation is defined structurally upon ct by the clauses

$$\begin{aligned}
\text{sim}_{A_i}[Q](u, v) &\equiv u = v \\
\text{sim}_{\mathbf{B}_i}[Q](u, v) &\equiv Q_i(u, v) \\
\text{sim}_{ct \times ct'}[Q](u, v) &\equiv \text{sim}_{ct}[Q](u \downarrow 1, v \downarrow 1) \wedge \text{sim}_{ct'}[Q](u \downarrow 2, v \downarrow 2) \\
\text{sim}_{ct + ct'}[Q](u, v) &\equiv (u = \bot \wedge v = \bot) \\
&\quad \vee \exists u', v'\colon u = \text{in}_1(u') \wedge v = \text{in}_1(v') \wedge \text{sim}_{ct}[Q](u', v') \\
&\quad \vee \exists u', v'\colon u = \text{in}_2(u') \wedge v = \text{in}_2(v') \wedge \text{sim}_{ct'}[Q](u', v') \\
\text{sim}_{ct \to ct'}[Q](u, v) &\equiv \forall u', v'\colon \text{sim}_{ct}[Q](u', v') \Rightarrow \text{sim}_{ct'}[Q](u(u'), v(v')).
\end{aligned}$$

The first two clauses clearly show that A_i and $\mathbf{B}_i$ are regarded differently. The data domains A_i contain "static entities" that must be the same in all interpretations. The data domains $\mathbf{B}_i$, on the other hand, contain "dynamic entities" that need not be the same in all interpretations. The remaining three clauses extend the relation "componentwise" to all types.

It is not difficult to show (by structural induction) that if each Q_i is admissible then all $\text{sim}_{ct}[Q]$ are admissible. A further result is that we get structural induction over expressions for free:

PROPOSITION 3. *If* $\text{sim}_{ct_i}[Q](\mathbf{I}(f_i), \mathbf{J}(f_i))$ *holds for all constants* f_i *of type* ct_i, *then* $\text{sim}_{ct}[Q](\mathbf{I}[\![e]\!](\bot), \mathbf{J}[\![e]\!](\bot))$ *holds for all closed expressions* e *of type* ct.

Proof. Let $\mathbf{I}$ and $\mathbf{J}$ be interpretations such that $\text{sim}_{ct_i}[Q](\mathbf{I}(f_i), \mathbf{J}(f_i))$ holds for all constants f_i of type ct_i. The proof then amounts to showing by structural induction on expressions e that

$$\begin{aligned}
&\text{if tenv} \vdash e\colon ct \\
&\text{where dom(tenv)} = \{x_1, ..., x_k\} \text{ and tenv}(x_i) = ct'_i \\
&\text{then } \text{sim}_{ct'_1 \times ... \times ct'_k \to ct}[Q](\mathbf{I}[\![e]\!], \mathbf{J}[\![e]\!]).
\end{aligned}$$

We omit the straightforward structural induction. (In the case of $\text{fix}_{ct}\, e$ we use Fact 2.) ▌

A similar use of relational functors and logical relations may be found in Nielson (1984), Abramsky (1985), and Mycroft and Jones (1985).

Later we shall study special instances of $\mathrm{sim}_{ct}[Q]$ in some detail and this motivates the use of special notation for these. The relation $\approx_{ct}$ is obtained by taking $Q_i = \lambda(u', v') \cdot \text{true}$, i.e.,

$$u \approx_{ct} v \equiv \mathrm{sim}_{ct}[(\lambda(u', v') \cdot \text{true})_i](u, v)$$

and this expresses whether two elements are indistinguishable in the type system. Note that $u \approx_{ct} v$ will not always be true (e.g., take $ct = A_i$). Assuming that each $\mathbf{I}(\mathbf{B}_i)$ equals $\mathbf{J}(\mathbf{B}_i)$ we may define the relation $\leqslant_{ct}$ by

$$u \leqslant_{ct} v \equiv \mathrm{sim}_{ct}[(\lambda(u', v') \cdot u' \sqsubseteq v')_i](u, v).$$

This relation gives a structural way of extending the approximation relation to all types. Note that $u \leqslant_{ct} v$ will not always be $u \sqsubseteq v$ (e.g., take $ct = A_i$). Finally, suppose there is a family *mv* of *strict* and *continuous* functions

$$mv_i \colon \mathbf{I}(\mathbf{B}_i) \to \mathbf{J}(\mathbf{B}_i)$$

and define the relation $\mathbf{mv}_{ct}$ by

$$u \;\mathbf{mv}_{ct}\; v \equiv \mathrm{sim}_{ct}[(\lambda(u', v') \cdot mv_i(u') \sqsubseteq v')_i](u, v).$$

It is not difficult to show that each $\mathbf{mv}_{ct}$ is an admissible relation. One may view $\leqslant_{ct}$ as $\mathbf{id}_{ct}$, where id is the family of identity functions and $\approx_{ct}$ as $\perp_{ct}$, where $\perp$ is the family of bottom functions.

It is convenient to extend the relations **mv**, $\leqslant$, and $\approx$ to interpretations. In the case of **mv** we may define

$$\begin{aligned} \mathbf{I} \;\mathbf{mv}\; \mathbf{J} &\Leftrightarrow \mathbf{I}(f_i) \;\mathbf{mv}_{ct_i}\; \mathbf{J}(f_i) && \text{for all } f_i \text{ of type } ct_i \\ &\Leftrightarrow \mathbf{I}[\![e]\!](\perp) \;\mathbf{mv}_{ct}\; \mathbf{J}[\![e]\!]\,(\perp) && \text{for all closed } e \text{ of type } ct, \end{aligned}$$

where the last implication is by Proposition 3. We define $\mathbf{I} \leqslant \mathbf{J}$ and $\mathbf{I} \approx \mathbf{J}$ in a similar way.

4. Abstract Interpretation

The interpretations of the metalanguage to be considered in this paper intuitively fall within two groups. One consists of the *standard* interpretation, **S**, that describes the input–output behaviour of expressions. The other consists of *approximating* interpretations, **I**, that describe various analyses of expressions. As motivated previously these will be lattice interpretations, i.e., the $\mathbf{I}(\mathbf{B}_i)$ will be complete lattices. In Section 6 we study an approximating interpretation that intuitively is as precise as possible; it is

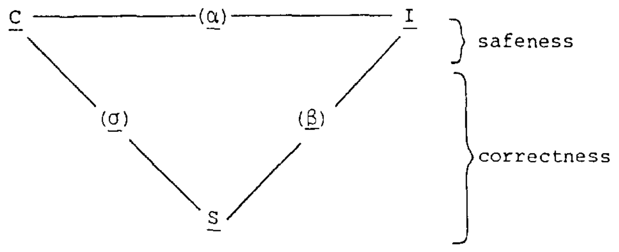
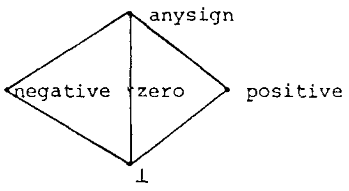


FIG. 2. Safeness and correctness.

called the *collecting* interpretation, **C**. Intuitively this interpretation works on certain sets of values (from the standard interpretation) and it corresponds to the static semantics of Cousot and Cousot (1979).[1]

The relations between these interpretations also intuitively fall into two groups. The relations between **S** and **C** and between **S** and **I** express the *correctness* of some approximating interpretation, **C** or **I**, with respect to the ordinary input–output behaviour. The relation between **C** and **I** express the fact that one analysis, **I**, is a *safe* approximation of another analysis, **C**; i.e., **I** may not be as precise as **C** but will not give information that contradicts information given by **C**. These relationships are illustrated in Fig. 2 and will be clarified shortly.

For a more concrete example suppose that there is just one base type, $\mathbf{B}_1$, and just one function, $f_1 : \mathbf{B}_1 \to \mathbf{B}_1$. We shall assume that the standard interpretation interprets the base type as the flat cpo $Z_\perp$ of integers and the function as some monotonic function $f: Z_\perp \to Z_\perp$. Concerning the collecting interpretation we assume that the base type is interpreted as the powerset $\mathscr{P}(Z)$ of integers ordered by subset inclusion $\subseteq$, and the function as some monotonic function g: $\mathscr{P}(Z) \to \mathscr{P}(Z)$. Finally, we must consider the approximating interpretation **I**, where we assume that the base type is interpreted as some complete lattice L and the function as some monotonic function h: $L \to L$. A possible choice of L is



[1] The term "collecting semantics" was introduced in Nielson (1982) as a variant of the standard semantics that had sets of values collected at program points. However, the use of "static semantics" in Cousot and Cousot (1979) conflicts with a well-established use of that term and eventually the term "collecting semantics" came to mean "static semantics in the sense of Cousot and Cousot (1979)" (see Mycroft, 1981; Mycroft and Nielson, 1983).

i.e., the complete lattice with elements $\perp$, anysign, negative, zero, and positive and ordered by

$$\perp \sqsubseteq l \sqsubseteq \text{anysign} \qquad \text{when } l \text{ is negative, zero, or positive.}$$

With these assumptions Fig. 2 specializes to Fig. 3 and we shall now explain the α, β, γ, and σ.

The functions β and σ are *strict* and *continuous* functions. They are termed *representation functions* since the intention is that $\beta(z) \in L$ and $\sigma(z) \in \mathscr{P}(Z)$ are the properties that best describe $z \in Z$. Thus it is natural to put

$$\begin{aligned} &\beta(\perp) = \perp \\ &\beta(z) = \text{negative} && \text{if } z < 0 \\ &\beta(0) = \text{zero} \\ &\beta(z) = \text{positive} && \text{if } z > 0 \end{aligned}$$

and

$$\sigma(z) = \begin{cases} \{z\} & \text{if } z \neq \perp \\ \varnothing & \text{if } z = \perp. \end{cases}$$

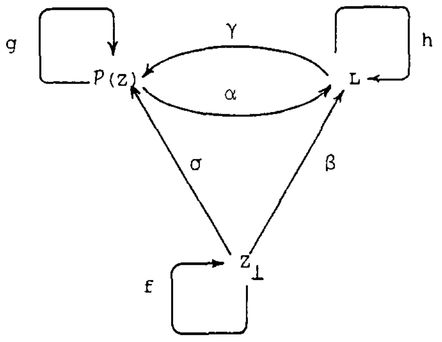
Clearly this defines strict and continuous functions β and σ.

The correctness of h with respect to f may then be expressed by the condition that

$$\beta(z) \sqsubseteq l \Rightarrow \beta(f(z)) \sqsubseteq h(l),$$

which says that

whenever l correctly describes z, $h(l)$ also correctly describes $f(z)$.



$$\begin{tikzcd} P(Z) \arrow[loop left, "g"] \arrow[rr, bend right, "\alpha"'] & & L \arrow[ll, bend right, "\gamma"'] \arrow[loop right, "h"] \\ & Z_\perp \arrow[ul, "\sigma"] \arrow[ur, "\beta"'] \arrow[loop left, "f"] & \end{tikzcd}$$

FIG. 3. An instance of Fig. 2.

So if, e.g., $z=0$, $f(0)=0$, and $l=$ anysign we have $\beta(0) \sqsubseteq$ anysign and thus h must satisfy that zero $\sqsubseteq h$(anysign). Note that the condition $\beta(z) \sqsubseteq l$ is nothing but $z \; \boldsymbol{\beta}_{\mathbf{B}} \; l$ so that the correctness of h with respect to f becomes

$$f \; \boldsymbol{\beta}_{\mathbf{B} \to \mathbf{B}} \; h.$$

Similarly,the correctness of g with respect to f is captured by $f \; \boldsymbol{\sigma}_{\mathbf{B} \to \mathbf{B}} \; g$.

The relationship between $\mathscr{P}(Z)$ and L is expressed using the framework of Abstract Interpretation (Cousot and Cousot, 1979). The fundamental ingredient is that of a pair (α, γ) of *abstraction* and *concretization* functions. The intention with the concretization function γ is to formalize the intuitive meaning of the properties in L, so one has $\gamma(\text{negative}) = \{..., -2, -1\}$, $\gamma(\bot) = \varnothing$, $\gamma(\text{anysign}) = Z$, etc. Given a set S of integers the intention with the abstraction function α is that $\alpha(S)$ is the *best safe* description of S in L, so one would expect, e.g., $\alpha(\{-3, -2\}) = \text{negative}$ and $\alpha(\{-3, 2\}) = \text{anysign}$.

To be a bit more precise, by "S is *safely* described by l" we shall mean that

$$S \subseteq \gamma(l).$$

So the set $\{-3, -2\}$ is safely described by any one of the properties negative and anysign. But γ(negative) is a proper subset of γ(anysign) and negative is therefore the *better* property. Actually it is more convenient if one could use the fact that negative $\sqsubseteq$ anysign to deduce that one should prefer negative. For this to succeed the partial orders of $\mathscr{P}(Z)$ and L must be suitably related. Similarly, there must be a relation between α and γ in order that one can claim that α produces the *best safe* description (w.r.t. γ). This is all captured by the *adjoinedness* condition (Cousot and Cousot, 1979).

$$\forall S \in \mathscr{P}(Z) \colon \forall l \in L \colon \qquad S \subseteq \gamma(l) \Leftrightarrow \alpha(S) \sqsubseteq l,$$

which may be reformulated as

$$\begin{array}{ll} \multicolumn{2}{l}{\alpha \text{ and } \gamma \text{ are monotonic}} \\ \forall S\colon & S \subseteq (\gamma \cdot \alpha)(S) \\ \forall l\colon & (\alpha \cdot \gamma)(l) \sqsubseteq l. \end{array}$$

We refer to Cousot and Cousot (1979) for a detailed motivation for demanding adjoinedness. Given adjoinedness one may formulate the safeness condition $S \subseteq \gamma(l)$ as

$$\alpha(S) \sqsubseteq l$$

and this is nothing but $S\,\boldsymbol{\alpha}_{\mathbf{B}}\,l$. The safeness of h with repect to g then is

$$\alpha(S) \sqsubseteq l \Rightarrow \alpha(g(S)) \sqsubseteq h(l),$$

which is nothing but $g\,\boldsymbol{\alpha}_{\mathbf{B}\to\mathbf{B}}\,h$. Due to the adjoinedness of α and γ and the monotonicity of g and h it may be reformulated as

$$\alpha \cdot g \cdot \gamma \sqsubseteq h,$$

as is illustrated in Fig. 4.

In general one may use abstraction and concretization functions to relate arbitrary complete lattices of properties. So if L and M are complete lattices and α: $L \to M$ and γ: $M \to L$ we say that (α, γ) is *adjoined* when

$$\forall l \in L: \forall m \in M: \qquad \alpha(l) \sqsubseteq m \Leftrightarrow l \sqsubseteq \gamma(m).$$

This may be reformulated as above and the intention is that $\alpha(l) \sqsubseteq m$ holds whenever l is safely described by m. The function α is called a *lower adjoint* iff there exists a function γ such that (α, γ) is an adjoined pair. There need not exist such a γ but if it does it is uniquely determined by the formula (Cousot and Cousot, 1979)

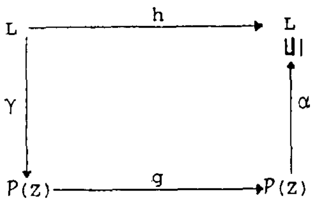
$$\gamma(m) = \bigsqcup \{l \mid \alpha(l) \sqsubseteq m\}.$$

(Adjoinedness is also studied in category theory (MacLane, 1971).) Whenever α is completely additive (see Section 2) it is a lower adjoint and vice versa. Hence each $\boldsymbol{\alpha}_{ct}$ will be an admissible relation. Finally, γ is called an *upper adjoint* iff there is an α such that (α, γ) is an adjoined pair and then α is uniquely determined by

$$\alpha(l) = \bigsqcap \{m \mid l \sqsubseteq \gamma(m)\},$$

where $\bigsqcap Y = \bigsqcup \{y \mid \forall y' \in Y: y \sqsubseteq y'\}$ is the greatest lower bound of Y.

It follows that there is a one–one correspondence between a lower adjoint α and an upper adjoint γ. Turning to Fig. 3 this may be extended with a one–one correspondence to a representation function β assuming



$$\begin{tikzcd} L \arrow[r, "h"] \arrow[d, "\gamma"'] & L \\ & \sqcup\!\shortmid \\ P(Z) \arrow[r, "g"] & P(Z) \arrow[u, "\alpha"'] \end{tikzcd}$$

FIG. 4. h safely approximates g.

that σ is fixed. For given α we may define a strict and continuous function β by $\beta = \alpha \cdot \sigma$ and given β we may define a completely additive function α by $\alpha(S) = \bigsqcup \{\beta(s) \mid s \in S\}$. It is not hard to show that these constructions are inverses of one another.

Returning to the general setting of Fig. 2 we have already argued that $\boldsymbol{\beta}_{\mathbf{B} \to \mathbf{B}}$ is the proper predicate for relating elements of $\mathbf{S}[\![\mathbf{B} \to \mathbf{B}]\!]$ and $\mathbf{I}[\![\mathbf{B} \to \mathbf{B}]\!]$. Extending this to all types we may express the *correctness* of $\mathbf{I}$ with respect to $\mathbf{S}$ as

$$\mathbf{S}\,\boldsymbol{\beta}\,\mathbf{I}$$

Similarly,

$$\mathbf{S}\,\boldsymbol{\sigma}\,\mathbf{C}$$

expresses the correctness of the interpretation $\mathbf{C}$ and

$$\mathbf{C}\,\boldsymbol{\alpha}\,\mathbf{I}$$

states that $\mathbf{I}$ is a *safe* approximation to $\mathbf{C}$. (Here each of β, σ, and α stands for a family of functions, e.g., β_i: $\mathrm{S}(\mathbf{B}_i) \to \mathbf{I}(\mathbf{B}_i)$.)

The Problem of Inducing

Assume now that $\mathbf{I}$ is a lattice interpretation describing some analysis and consider the task of defining another analysis $\mathbf{J}$ that safely approximates $\mathbf{I}$, i.e., $\mathbf{I}\ \boldsymbol{\alpha}\ \mathbf{J}$. The choice of the complete lattices $\mathbf{J}(\mathbf{B}_i)$ is intended to record a deliberate decision about certain aspects that are to be treated less precisely in $\mathbf{J}$ than in $\mathbf{I}$, e.g., to use $\mathbf{J}(\mathbf{B}_1) = L$ whereas $\mathbf{I}(\mathbf{B}_1) = \mathscr{P}(Z)$. Given this choice there is still much freedom when defining $\mathbf{J}(f_i)$ as a safe approximation to $\mathbf{I}(f_i)$. As an example let f_1 have type $\mathbf{B}_1 \to \mathbf{B}_1$ so that the safeness condition

$$\mathbf{I}(f_1)\,\boldsymbol{\alpha}_{\mathbf{B}_1 \to \mathbf{B}_1}\,\mathbf{J}(f_1)$$

becomes

$$\alpha_1 \cdot \mathbf{I}(f_1) \cdot \gamma_1 \sqsubseteq \mathbf{J}(f_1)$$

(compare Fig. 4, suitably relabelled). It follows that $\mathbf{J}(f_1) = \alpha_1 \cdot \mathbf{I}(f_1) \cdot \gamma_1$ will be the best choice. In particular it will be better than $\mathbf{J}(f_1) = \lambda v \cdot \top$, where $\top = \bigsqcup \mathbf{J}(\mathbf{B}_1)$ is the greatest element of $\mathbf{J}(\mathbf{B}_1)$, as this choice would yield a rather uninformative analysis. So in general we want to define $\mathbf{J}(f_i)$ in a best way, given $\mathbf{J}(\mathbf{B}_i)$ and $\mathbf{I}(f_i)$, so that the resulting analysis will not be too uninformative. This is the problem of *inducing* $\mathbf{J}(f_i)$ from $\mathbf{I}(f_i)$ and the criterion that $\mathbf{J}(f_i)$ is *best* may be formulated as

$$\forall h \in \mathbf{J}[\![ct_i]\!]: \mathbf{I}(f_i)\,\boldsymbol{\alpha}_{ct_i}\,h \Rightarrow \mathbf{J}(f_i) \leqslant_{ct_i} h,$$

where, as usual,

$$\mathbf{I}(f_i)\ \boldsymbol{\alpha}_{ct_i}\ \mathbf{J}(f_i)$$

expresses the safeness condition. In particular it is important to observe that $\leqslant_{ct}$ rather than $\sqsubseteq$ is the proper relation to use for comparing analyses (as is clear when, e.g., $ct = A_i$).

So let $\mathbf{I}$ be a lattice interpretation, $\mathbf{J}$ a lattice type interpretation, and (α, γ) a family of adjoined functions

$$(\alpha_i : \mathbf{I}(\mathbf{B}_i) \to \mathbf{J}(\mathbf{B}_i), \gamma_i : \mathbf{J}(\mathbf{B}_i) \to \mathbf{I}(\mathbf{B}_i)).$$

The functions α_i and γ_i move elements from $\mathbf{I}(\mathbf{B}_i)$ to $\mathbf{J}(\mathbf{B}_i)$ and vice versa and we now consider how to extend this to all types and then later to interpretations. Let us abbreviate

$$\mathbf{IJ}[\![ct]\!] = (\mathbf{I}[\![ct]\!] \to \mathbf{J}[\![ct]\!]) \times (\mathbf{J}[\![ct]\!] \to \mathbf{I}[\![ct]\!])$$

and note that (α_i, γ_i) is an element of $\mathbf{IJ}[\![\mathbf{B}_i]\!]$. Consider next a type ct that mentions only $\mathbf{B}_i$'s in the list $\mathbf{B}_1, ..., \mathbf{B}_N$. We then define a function

$$[\![ct]\!] : (\mathbf{IJ}[\![\mathbf{B}_1]\!] \times \cdots \times \mathbf{IJ}[\![\mathbf{B}_N]\!]) \to \mathbf{IJ}[\![ct]\!]$$

as follows, where we write ϕ_i for (α_i, γ_i) and where id is the identity,

$$[\![A_i]\!](\phi_1, ..., \phi_N) = (\text{id}, \text{id})$$

$$[\![\mathbf{B}_i]\!](\phi_1, ..., \phi_N) = \phi_i$$

$$\begin{aligned}
&[\![ct' + ct'']\!](\phi_1, ..., \phi_N)\\
&\quad = (\lambda u \cdot \text{is}_1(u) \to \text{in}_1([\![ct']\!](\phi_1, ..., \phi_N) \downarrow 1\ (\text{out}_1(u))),\\
&\qquad\qquad\qquad \text{in}_2([\![ct'']\!](\phi_1, ..., \phi_N) \downarrow 1\ (\text{out}_2(u))),\\
&\qquad \lambda v \cdot \text{is}_1(v) \to \text{in}_1([\![ct']\!](\phi_1, ..., \phi_N) \downarrow 2\ (\text{out}_1(v))),\\
&\qquad\qquad\qquad \text{in}_2([\![ct'']\!](\phi_1, ..., \phi_N) \downarrow 2\ (\text{out}_2(v))))
\end{aligned}$$

$$\begin{aligned}
[\![ct' \times ct'']\!](\phi_1, ..., \phi_N) = (&\lambda(u', u'') \cdot ([\![ct']\!](\phi_1, ..., \phi_N) \downarrow 1(u'),\\
&\qquad [\![ct'']\!](\phi_1, ..., \phi_N) \downarrow 1(u'')),\\
&\lambda(v', v'') \cdot ([\![ct']\!](\phi_1, ..., \phi_N) \downarrow 2(v'),\\
&\qquad [\![ct'']\!](\phi_1, ..., \phi_N) \downarrow 2(v'')))
\end{aligned}$$

$$\begin{aligned}
[\![ct' \to ct'']\!](\phi_1, ..., \phi_N) = (&\lambda f \cdot [\![ct'']\!](\phi_1, ..., \phi_N) \downarrow 1 \cdot f \cdot [\![ct']\!](\phi_1, ..., \phi_N) \downarrow 2,\\
&\lambda g \cdot [\![ct'']\!](\phi_1, ..., \phi_N) \downarrow 2 \cdot g \cdot [\![ct']\!](\phi_1, ..., \phi_N) \downarrow 1).
\end{aligned}$$

We now explain these equations.

The equation for $\mathbf{B}_i$ is straightforward, as is the one for A_i because $\mathbf{I}(\mathbf{A}_i)=\mathbf{J}(\mathbf{A}_i)$. The equations for $ct'+ct''$ and $ct'\times ct''$ express a componentwise definition that we shall illustrate for $ct'\times ct''$ assuming that $N=0$. Then

$$[\![ct']\!](\ )\downarrow 1: \mathbf{I}[\![ct']\!]\to\mathbf{J}[\![ct']\!]$$

$$[\![ct'']\!](\ )\downarrow 1: \mathbf{I}[\![ct'']\!]\to\mathbf{J}[\![ct'']\!]$$

so that

$$\lambda(u',u'')\cdot([\![ct']\!](\ )\downarrow 1(u'), [\![ct'']\!](\ )\downarrow 1(u''))$$

has functionality

$$\mathbf{I}[\![ct'\times ct'']\!]\to\mathbf{J}[\![ct'\times ct'']\!]$$

and it is correct to define $[\![ct'\times ct'']\!](\ )\downarrow 1$ to be this function. A similar explanation motivates the definition of $[\![ct'\times ct'']\!](\ )\downarrow 2$ and hence of $[\![ct'\times ct'']\!]$. The equation for $ct'\to ct''$ may be surprising in that $[\![ct'\to ct'']\!](\ )\downarrow 1$ uses $[\![ct']\!](\ )\downarrow 2$ and $[\![ct'']\!](\ )\downarrow 1$ rather than $[\![ct']\!](\ )\downarrow 1$ and $[\![ct'']\!](\ )\downarrow 1$. However, it will be seen from Fig. 5 that this is the correct thing to do. In particular if $ct'=\mathbf{B}_1=ct''$ then $[\![ct'\to ct'']\!](\ )\downarrow 1$ maps f to $\alpha_1\cdot f\cdot\gamma_1$ (because $[\![ct']\!](\ )\downarrow 2=\gamma_1$ and $[\![ct'']\!](\ )\downarrow 1=\alpha_1$). A similar explanation motivates the definition of $[\![ct'\to ct'']\!]\downarrow 2$ and hence of $[\![ct'\to ct'']\!]$. That the above equations do define a function $[\![ct]\!]$ of the stated functionality is a consequence of

PROPOSITION 4. *If ct mentions only $\mathbf{B}_i$'s among $\mathbf{B}_1, \ldots, \mathbf{B}_N$ the equations define a monotonic function*

$$[\![ct]\!]: \mathbf{IJ}[\![\mathbf{B}_1]\!]\times\cdots\times\mathbf{IJ}[\![\mathbf{B}_N]\!]\to\mathbf{IJ}[\![ct]\!]$$

that satisfies the functor laws

$$[\![ct]\!]((\mathrm{id},\mathrm{id}),\ldots,(\mathrm{id},\mathrm{id}))=(\mathrm{id},\mathrm{id})$$

$$[\![ct]\!](\phi'_1\cdot\phi_1,\ldots,\phi'_N\cdot\phi_N)=[\![ct]\!](\phi'_1,\ldots,\phi'_N)\cdot[\![ct]\!](\phi_1,\ldots,\phi_N),$$

where $\phi'\cdot\phi=(\phi'\downarrow 1\cdot\phi\downarrow 1, \phi\downarrow 2\cdot\phi'\downarrow 2)$.

$$\begin{tikzcd}
\underline{J}[\![ct']\!] \arrow[r, dashed] \arrow[d, "{[\![ct']\!](\ )\downarrow 2}"] & \underline{J}[\![ct'']\!] \\
\underline{I}[\![ct']\!] \arrow[r, "f"'] & \underline{I}[\![ct'']\!] \arrow[u, "{[\![ct'']\!](\ )\downarrow 1}"']
\end{tikzcd}$$

FIG. 5. The definition of $[\![ct'\to ct]\!](\ )\downarrow 1$.

We shall begin by motivating the definition of $\phi' \cdot \phi$. Here ϕ' is (α', γ') and ϕ is (α, γ) so the definition reads

$$(\alpha', \gamma') \cdot (\alpha, \gamma) = (\alpha' \cdot \alpha, \gamma \cdot \gamma').$$

This is illustrated by Fig. 6, which shows that if (α, γ) and (α', γ') are pairs of abstraction and concretization functions then so is $(\alpha' \cdot \alpha, \gamma \cdot \gamma')$ and this pair may be regarded as the composition of (α', γ') and (α, γ). The proof is by a straightforward structural induction and is omitted. There are many more results one can show about $[\![ct]\!]$ but we shall postpone this and instead consider the problems in passing from $\mathbf{I}[\![ct]\!]$ to $\mathbf{J}[\![ct]\!]$ when we do not have pairs of adjoined functions that relate $\mathbf{I}(\mathbf{B}_i)$ and $\mathbf{J}(\mathbf{B}_i)$. An example of this setting is one in which we want to induce an analysis directly from the standard interpretation rather than from some other analysis (such as the collecting semantics). So let $\mathbf{L}$ be an interpretation and $\mathbf{R}$ a lattice type interpretation and let mv be a family of *strict* and *continuous* functions

$$mv_i \colon \mathbf{L}(\mathbf{B}_i) \to \mathbf{R}(\mathbf{B}_i).$$

Note that it is *not* assumed that $\mathbf{L}(\mathbf{B}_i)$ is a complete lattice nor is it assumed that mv_i is a lower adjoint.

We are searching for a way to transform elements of $\mathbf{L}[\![ct]\!]$ into $\mathbf{R}[\![ct]\!]$; i.e., we are looking for a suitable function

$$\text{move}_{ct}[mv] \colon \mathbf{L}[\![ct]\!] \to \mathbf{R}[\![ct]\!].$$

Motivated by the definition of $[\![ct]\!](\phi_1, \ldots, \phi_N) \downarrow 1$ it is natural for us to propose

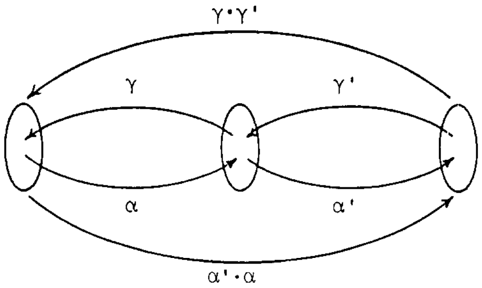


FIG. 6. The composition of pairs of functions.

$$\text{move}_{A_i}[mv] = \lambda l \cdot l$$

$$\text{move}_{\mathbf{B}_i}[mv] = mv_i$$

$$\text{move}_{ct'+ct''}[mv] = \lambda l \cdot \text{is}_1(l) \to \text{in}_1(\text{move}_{ct'}[mv]\,(\text{out}_1(l))), \\ \text{in}_2(\text{move}_{ct''}[mv](\text{out}_2(l)))$$

$$\text{move}_{ct' \times ct''}[mv] = \lambda(l', l'') \cdot (\text{move}_{ct'}[mv](l'), \text{move}_{ct''}[mv](l'')).$$

It is more troublesome to handle the case $ct = ct' \to ct''$, as is illustrated by the need to use $[\![ct']\!](\phi_1, ..., \phi_N) \downarrow 2$ when defining $[\![ct' \to ct'']\!](\phi_1, ..., \phi_N) \downarrow 1$, but we propose

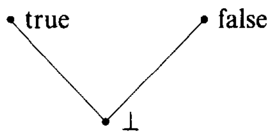
$$\text{move}_{ct' \to ct''}[mv] = \lambda l \cdot \lambda ra \cdot \bigsqcup \{\text{move}_{ct''}[mv](l(la)) \mid la\ \mathbf{mv}_{ct'}\ ra\}.$$

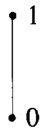
Here l is a function from $\mathbf{L}[\![ct']\!]$ to $\mathbf{L}[\![ct'']\!]$ and la is an argument in $\mathbf{L}[\![ct']\!]$ corresponding to an argument ra in $\mathbf{R}[\![ct']\!]$. One motivation for the above formula is the formula

$$\gamma(ra) = \bigsqcup \{la \mid \alpha(la) \sqsubseteq ra\}$$

for an upper adjoint in terms of a lower adjoint, and another is that la $\mathbf{mv}_{ct'}$ ra is the generalization of $\alpha(la) \sqsubseteq ra$ to all types (assuming $\alpha_i = mv_i$).[2] The idea thus is that since we have no function to transform an element $ra \in \mathbf{R}[\![ct']\!]$ into an element of $\mathbf{L}[\![ct']\!]$ we must instead consider all $la \in \mathbf{L}[\![ct']\!]$ that are safely described by ra and then combine the results for all these la's.

However, the above definition is not well-defined as it stands. The motivation for requiring $\mathbf{R}$ to be a lattice type interpretation is that the least upper bound in the definition of $\text{move}_{ct' \to ct''}$ then exists if ct'' is some $\mathbf{B}_i$. But this is not enough to guarantee existence in general as the following example shows. Let T and O be the cpo's given by

T: 

O: 

[2] Theorem 12 will give conditions for when la $\mathbf{mv}_{ct'}$ ra is equivalent to $\text{move}_{ct'}[mv](la) \leqslant_{ct'} ra$ but note that in general (e.g., when $ct' = A_i$) this will not be equivalent to $\text{move}_{ct'}[mv](la) \sqsubseteq ra$.

Suppose that $\mathbf{L}(\mathbf{B}_1)=T$ and that all other $\mathbf{L}(\mathbf{B}_i)$ and all $\mathbf{R}(\mathbf{B}_i)$ are O. Define

$$mv_1(\text{true})=1, \qquad mv_1(\text{false})=1, \qquad mv_1(\perp)=0$$

and otherwise $mv_i(x)=x$. Finally consider

$$\mathbf{L}(\phi)\in\mathbf{L}[\![\mathbf{B}_1\to\mathbf{B}_2+\mathbf{B}_2]\!]$$

defined by

$$\mathbf{L}(\phi)(\text{true})=\text{in}_1(1), \qquad \mathbf{L}(\phi)(\text{false})=\text{in}_2(1), \qquad \mathbf{L}(\phi)(\perp)=\perp.$$

When calculating $\text{move}_{\mathbf{B}_1\to\mathbf{B}_1+\mathbf{B}_1}[mv](\mathbf{L}(\phi))(1)$ we have

$$\text{true}\ \mathbf{mv}_{\mathbf{B}_1}\ 1, \qquad \text{false}\ \mathbf{mv}_{\mathbf{B}_1}\ 1, \qquad \perp\ \mathbf{mv}_{\mathbf{B}_1}\ 1$$

and so must take the least upper bound of the subset

$$\{\text{in}_1(1), \text{in}_2(1), \perp\}$$

of $\mathbf{R}[\![\mathbf{B}_1+\mathbf{B}_1]\!]=O+O$. But this set has no least upper bound.

Faithfulness

To overcome the above problem we shall exclude functions like $\mathbf{L}(\phi)$ from consideration in the hope that this will be sufficient for move to be well-defined. Intuitively, the problem with $\mathbf{L}(\phi)$ is that different elements of type $\mathbf{B}_1$ give rise to results (of type $\mathbf{B}_2+\mathbf{B}_2$) that can be distinguished in the type system, namely that one result is in the first summand and the other result is in the second summand. So the idea will be to impose as a condition that differences among elements of the $\mathbf{B}_i$ may lead to different results in the $\mathbf{B}_i$ but not differences that can be distinguished by the type system. This vague formulation is captured by the relation

$$u\approx_{ct}v\equiv\text{sim}_{ct}[(\lambda(u',v')\cdot\text{true})_i](u,v)$$

defined earlier. As an example we shall show that

$$\mathbf{L}(\phi)\not\approx_{\mathbf{B}_1\to\mathbf{B}_2+\mathbf{B}_2}\mathbf{L}(\phi).$$

It is straightforward to calculate that

$$\text{true}\approx_{\mathbf{B}_1}\text{false}$$

$$\neg(\text{in}_1(1)\approx_{\mathbf{B}_2+\mathbf{B}_2}\text{in}_2(1))$$

and this proves the claim.

Given a type interpretation $\mathbf{I}$ we now define a type interpretation $\mathbf{I}_{\mathrm{f}}$ called the *faithful part* of $\mathbf{I}$. The aim is to restrict $\mathbf{I}[\![ct]\!]$ to containing only well-behaved elements. The definition is

$$\begin{aligned}
&\mathbf{I}_{\mathrm{f}}[\![A_i]\!] = \mathbf{I}[\![A_i]\!]\\
&\mathbf{I}_{\mathrm{f}}[\![\mathbf{B}_i]\!] = \mathbf{I}(\mathbf{B}_i)\\
&\mathbf{I}_{\mathrm{f}}[\![ct' \times ct'']\!] = \mathbf{I}_{\mathrm{f}}[\![ct']\!] \times \mathbf{I}_{\mathrm{f}}[\![ct'']\!] \qquad \text{(cartesian product)}\\
&\mathbf{I}_{\mathrm{f}}[\![ct' + ct'']\!] = \mathbf{I}_{\mathrm{f}}[\![ct']\!] + \mathbf{I}_{\mathrm{f}}[\![ct'']\!] \qquad \text{(separated sum)}\\
&\mathbf{I}_{\mathrm{f}}[\![ct' \to ct'']\!] = \{f\colon \mathbf{I}_{\mathrm{f}}[\![ct']\!] \to \mathbf{I}_{\mathrm{f}}[\![ct'']\!] \mid f \text{ is monotonic}\\
&\qquad\qquad\qquad\qquad\qquad \text{and } f \leqslant_{ct' \to ct''} f \text{ and } f \approx_{ct' \to ct''} f\}.
\end{aligned}$$

The requirement that functions in $\mathbf{I}_{\mathrm{f}}[\![ct' \to ct'']\!]$ must be monotonic is just as before and we have already motivated the requirement that $f \approx_{ct' \to ct''} f$ by the desire to exclude $\mathbf{L}(\phi)$ from consideration. The requirement that $f \leqslant_{ct' \to ct''} f$ amount to

$$u \leqslant_{ct'} v \Rightarrow f(u) \leqslant_{ct''} f(v)$$

so that when v is a safe approximation of u then $f(v)$ is also a safe approximation of $f(u)$. This may be expressed as the requirement that f be monotonic with respect to "safe approximation." Finally, note that since $\leqslant_{ct' \to ct''}$ and $\approx_{ct' \to ct''}$ are admissible relations $\mathbf{I}_{\mathrm{f}}[\![ct' \to ct'']\!]$ is still a cpo and hence all $\mathbf{I}_{\mathrm{f}}[\![ct]\!]$ are cpo's. Also, the least upper bounds of chains in $\mathbf{I}_{\mathrm{f}}[\![ct]\!]$ are calculated as in $\mathbf{I}[\![ct]\!]$. When $\mathbf{I}$ is an interpretation such that all constants $\mathbf{I}(f_i)$ are elements of $\mathbf{I}_{\mathrm{f}}[\![ct_i]\!]$ we say that $\mathbf{I}$ is a *faithful interpretation.*

We now restrict our attention to the faithful part of a (type) interpretation $\mathbf{I}$ and study the families $\leqslant$ and $\approx$ of relations.

DEFINITION. A subset Y of $\mathbf{I}_{\mathrm{f}}[\![ct]\!]$ is a *faithful set* iff

$$\forall y_1, y_2 \in Y\colon \qquad y_1 \approx_{ct} y_2.$$

An element y of $\mathbf{I}_{\mathrm{f}}[\![ct]\!]$ is *faithful* iff $\{y\}$ is a faithful set and it is *faithful to* the element y' of $\mathbf{I}_{\mathrm{f}}[\![ct]\!]$ iff $\{y, y'\}$ is a faithful set.

Faithful sets are of interest because they exclude the set

$$\{\mathrm{in}_1(1), \mathrm{in}_2(1), \perp\},$$

which caused problems previously. Some properties of faithful sets and elements are given by:

FACT 5. *If $Y \subseteq \mathbf{I}_{\mathrm{f}}[\![ct]\!]$ is a faithful set and $g\colon \mathbf{I}_{\mathrm{f}}[\![ct]\!] \to \mathbf{J}_{\mathrm{f}}[\![ct]\!]$ is faithful, i.e., $u \approx_{ct} v \Rightarrow g(u) \approx_{ct} g(v)$, then*

$$g[Y] = \{g(y) \mid y \in Y\}$$

is a faithful set.

FACT 6. *All elements of $\mathbf{I}_{\mathrm{f}}[\![ct]\!]$ are faithful and $\leqslant$-monotonic; i.e., if $v \in \mathbf{I}_{\mathrm{f}}[\![ct]\!]$ then $v \approx_{ct} v$ and $v \leqslant_{ct} v$ hold.*

The proof of Fact 5 is straightforward and the proof of Fact 6 is by an easy structural induction.

Recall that the composition $R_1 \cdot R_2$ of two relations R_1 and R_2 is given by

$$u\, R_1 \cdot R_2\, w \equiv \exists v\colon u\, R_1\, v \wedge v\, R_2 w$$

and that an equivalence relation is a reflexive, symmetric, and transitive relation. We then have

PROPOSITION 7. *Let $\mathbf{L}$ be a type interpretation and let us restrict our attention to the faithful part $\mathbf{L}_{\mathrm{f}}$. Then*

$\leqslant_{ct}$ is a partial order on $\mathbf{L}_{\mathrm{f}}[\![ct]\!]$ that implies $\sqsubseteq$,

$\approx_{ct}$ is an equivalence relation on $\mathbf{L}_{\mathrm{f}}[\![ct]\!]$, and

$\leqslant_{ct} \cdot \approx_{ct}$ is $\approx_{ct}$ and $\approx_{ct} \cdot \leqslant_{ct}$ is $\approx_{ct}$.

Note that it trivially follows from this proposition that $\leqslant_{ct} \cdot \leqslant_{ct}$ is $\leqslant_{ct}$ and $\approx_{ct} \cdot \approx_{ct}$ is $\approx_{ct}$.

Proof. The proof is by structural induction on ct and most cases are straightforward. (In the case of $ct = ct' + ct''$ it is important that we use the separated sum rather than the coalesced sum.) We illustrate only the harder case when $ct = ct' \to ct''$. That $\leqslant_{ct}$ is reflexive on $\mathbf{L}_{\mathrm{f}}[\![ct]\!]$ is because $\mathbf{L}_{\mathrm{f}}[\![ct]\!]$ is defined to include only functions f that satisfy $f \leqslant_{ct} f$. To see that $\leqslant_{ct}$ is transitive let $f \leqslant_{ct} g \leqslant_{ct} h$. If $v \leqslant_{ct'} w$ we have $v \leqslant_{ct'} v \leqslant_{ct'} w$ by the induction hypothesis so that $f(v) \leqslant_{ct''} g(v) \leqslant_{ct''} h(w)$ and by the induction hypothesis $f(v) \leqslant_{ct''} h(w)$, as was to be shown. For anti-symmetry let $f \leqslant_{ct} g \leqslant_{ct} f$. For $v \in \mathbf{L}_{\mathrm{f}}[\![ct']\!]$ we have $v \leqslant_{ct'} v$ by the induction hypothesis so that $f(v) \leqslant_{ct''} g(v) \leqslant_{ct''} f(v)$. It follows that $f(v) = g(v)$ for all v so that $f = g$. Finally, $\leqslant_{ct}$ implies $\sqsubseteq$ because if $f \leqslant_{ct} g$ and $v \in \mathbf{L}_{\mathrm{f}}[\![ct']\!]$ we have $v \leqslant_{ct'} v$ and $f(v) \leqslant_{ct''} g(v)$; so by hypothesis $f(v) \sqsubseteq g(v)$ so that $f \sqsubseteq g$.

That $\approx_{ct}$ is reflexive is because $\mathbf{L}_{\mathrm{f}}[\![ct]\!]$ is defined to include only faithful functions. For transitivity let $f \approx_{ct} g \approx_{ct} h$. If $v \approx_{ct} w$ we have $v \approx_{ct} v \approx_{ct} w$ by hypothesis so that $f(v) \approx_{ct''} g(v) \approx_{ct''} h(w)$ and hence $f(v) \approx_{ct''} h(w)$. That $\approx_{ct}$ is symmetric is shown in a similar way.

Next consider the equation $\leqslant_{ct} \cdot \approx_{ct} = \approx_{ct}$. That $f \approx_{ct} g$ implies $f(\leqslant_{ct} \cdot \approx_{ct}) g$ is because $f \leqslant_{ct} f \approx_{ct} g$ since we have shown $\leqslant_{ct}$ to be reflexive. That $f \leqslant_{ct} g \approx_{ct} h$ implies $f \approx_{ct} h$ is because $v \approx_{ct'} w$ implies $v \leqslant_{ct'} v \approx_{ct'} w$ so that $f(v) \leqslant_{ct''} g(v) \approx_{ct''} h(w)$ and the result then follows by the induction hypothesis. The other equation is similar. ▮

For a partial order it is of interest to know whether least upper bounds exist.

Proposition 8. *Let* $\mathbf{L}$ *be a type interpretation and let us restrict our attention to the faithful part* $\mathbf{L}_f$. *If* $Y \subseteq \mathbf{L}_f[\![ct]\!]$ *is a non-empty and faithful set such that* $\bigsqcup Y$ *exists in* $\mathbf{L}_f[\![ct]\!]$ *then*

$\bigsqcup Y$ *is the least upper bound of* Y *w.r.t.* $\leqslant_{ct}$, *and*

$Y \cup \{\bigsqcup Y\}$ *is a faithful set.*

The requirement that Y be non-empty is because $\perp = \bigsqcup \varnothing$ is not the least upper bound of $\varnothing \subseteq \mathbf{L}_f[\![\mathbf{A}_i]\!]$ with respect to $\leqslant_{A_i}$, which is $=$.

Proof. The proof is by structural induction on ct and we illustrate the case where $ct = ct' \to ct''$. We know that $\bigsqcup Y = \lambda v \cdot \bigsqcup Y[v]$, where $Y[v] = \{y(v) \mid y \in Y\}$. To see that $\bigsqcup Y$ is an upper bound of Y w.r.t. $\leqslant_{ct}$ consider some $y \in Y$ and show $y \leqslant_{ct} \bigsqcup Y$. For this let $v \leqslant_{ct'} w$ and show $y(v) \leqslant_{ct''} \bigsqcup Y[w]$. Since $y \leqslant_{ct} y$ we have $y(v) \leqslant_{ct''} y(w)$. Since $y(w) \in Y[w]$ we have $y(w) \leqslant_{ct''} \bigsqcup Y[w]$ by the induction hypothesis. The result then follows by the transitivity of $\leqslant_{ct''}$. To see that $\bigsqcup Y$ is the least upper bound of Y w.r.t. $\leqslant_{ct}$ let $g \in \mathbf{L}_f[\![ct]\!]$ be an upper bound of Y. When $v \leqslant_{ct'} w$ we have $y(v) \leqslant_{ct''} g(w)$ for all $y \in Y$ and by the induction hypothesis $\bigsqcup Y[v] \leqslant_{ct''} g(w)$. This shows $\bigsqcup Y \leqslant_{ct} g$.

To show that $Y \cup \{\bigsqcup Y\}$ is faithful it suffices by transitivity of $\approx_{ct}$ to consider $y \in Y$ and show $y \approx_{ct} \bigsqcup Y$. When $v \approx_{ct'} w$ we have $y(v) \approx_{ct''} y(w)$ so by transitivity of $\approx_{ct''}$ it suffices to consider $w \in \mathbf{L}_f[\![ct']\!]$ and show $y(w) \approx_{ct''} (\bigsqcup Y)(w)$. But $(\bigsqcup Y)(w) = \bigsqcup Y[w]$ and the result follows because the induction hypothesis asserts that $Y[w] \cup \{\bigsqcup Y[w]\}$ is faithful. ▮

For lattice type interpretations we do not need to assume that $\bigsqcup Y$ exists:

Proposition 9. *Let* $\mathbf{R}$ *be a lattice type interpretation. If* $Y \subseteq \mathbf{R}_f[\![ct]\!]$ *is a non-empty and faithful set then* $\bigsqcup Y$ *exists in* $\mathbf{R}_f[\![ct]\!]$.

Proof. The proof is by structural induction on ct and we illustrate the case where $ct = ct' \to ct''$. We first show that $\bigsqcup Y$ exists in $\mathbf{R}_f[\![ct']\!] \to \mathbf{R}_f[\![ct'']\!]$. For $v \in \mathbf{R}_f[\![ct']\!]$ we have $v \approx_{ct'} v$ and therefore the set $Y[v] = \{y(v) \mid y \in Y\}$ is nonempty and faithful. Therefore $\lambda v \cdot \bigsqcup Y[v]$ exists by the induction hypothesis and it equals $\bigsqcup Y$ in $\mathbf{R}_f[\![ct']\!] \to \mathbf{R}_f[\![ct'']\!]$. (Clearly $\bigsqcup Y$ is a monotonic function.) To show that $\bigsqcup Y$ exists in $\mathbf{R}_f[\![ct]\!]$ we must show that $\bigsqcup Y$ is faithful and $\leqslant$-monotonic. Let y be some element of Y. If $v \approx_{ct'} w$ we have $y(v) \approx_{ct''} y(w)$. By Proposition 8, $\bigsqcup Y[v] \approx_{ct''} y(v)$ and $\bigsqcup Y[w] \approx_{ct''} y(w)$. By transitivity of $\approx_{ct''}$ we get $(\bigsqcup Y)(v) \approx_{ct''} (\bigsqcup Y)\ (w)$. Next if $v \leqslant_{ct'} w$ we have $y(v) \leqslant_{ct''} y(w)$. Since $y(w) \leqslant_{ct''} (\bigsqcup Y)(w)$ by Proposition 8 we get $y(v) \leqslant_{ct''} (\bigsqcup Y)\,(w)$. By yet another application of Proposition 8 we get $(\bigsqcup Y)\ (v) \leqslant_{ct''} (\bigsqcup Y)\,(w)$. ∎

Before we prove the existence of move we need some results about the behaviour of $[\![ct]\!]$. Let us write

$$\mathbf{IJ}_f[\![ct]\!] = (\mathbf{I}_f[\![ct]\!] \to \mathbf{J}_f[\![ct]\!]) \times (\mathbf{J}_f[\![ct]\!] \to \mathbf{I}_f[\![ct]\!])$$

and begin by stating that $[\![ct]\!]$ specializes to a mapping between faithful parts:

PROPOSITION 10. *If ct mentions only* $\mathbf{B}_i$*'s among* $\mathbf{B}_1, ..., \mathbf{B}_N$ *the equations for* $[\![ct]\!]$ *may be viewed as defining a monotonic function*

$$[\![ct]\!]: \mathbf{IJ}_f[\![\mathbf{B}_1]\!] \times \cdots \times \mathbf{IJ}_f[\![\mathbf{B}_N]\!] \to \mathbf{IJ}_f[\![ct]\!]$$

that satisfies the functor laws of Proposition 4 *and such that*

$$[\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 1: \mathbf{I}_f[\![ct]\!] \to \mathbf{J}_f[\![ct]\!]$$
$$[\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 2: \mathbf{J}_f[\![ct]\!] \to \mathbf{I}_f[\![ct]\!]$$

are faithful and $\leqslant$*-monotonic when all* α_i *and* γ_i *are monotonic.*

Proof. The proof is by structural induction on ct and we illustrate the case where $ct = ct' \to ct''$. As a shorthand we shall write (α, γ) instead of $((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N))$. We concentrate on showing the correctness of the functionality of $[\![ct]\!](\alpha, \gamma) \downarrow 1$ and that it is monotonic, faithful, and $\leqslant$-monotonic. The results about $[\![ct]\!](\alpha, \gamma) \downarrow 2$ are similar and together with the proof of Proposition 4 the remaining results follow.

Concerning the functionality we must show that whenever $f \in \mathbf{I}_f[\![ct' \to ct'']\!]$ then

$$[\![ct]\!]\,(\alpha, \gamma) \downarrow 1(f) = [\![ct'']\!]\,(\alpha, \gamma) \downarrow 1 \cdot f \cdot [\![ct']\!]\,(\alpha, \gamma) \downarrow 2$$

is monotonic, faithful, and $\leqslant$-monotonic. But f is by assumption and $[\![ct']\!]\ (\alpha,\ \gamma) \downarrow 2$ and $[\![ct'']\!]\,(\alpha, \gamma) \downarrow 1$ are by the induction hypothesis and the result then follows because monotonicity, faithfulness, and $\leqslant$-monotonicity are preserved under composition. Next suppose that $f \leqslant_{ct} g$ and show that

$$[\![ct]\!](\alpha, \gamma) \downarrow 1(f) \leqslant_{ct} [\![ct]\!]\,(\alpha, \gamma) \downarrow 1(g).$$

But if $u \leqslant_{ct'} v$ we get

$$[\![ct']\!]\,(\alpha, \gamma) \downarrow 2(u) \leqslant_{ct'} [\![ct']\!]\,(\alpha, \gamma) \downarrow 2(v)$$

by the induction hypothesis and hence

$$f([\![ct']\!]\,(\alpha, \gamma) \downarrow 2(u)) \leqslant_{ct''} g([\![ct']\!]\,(\alpha, \gamma) \downarrow 2(v))$$

so that by using the induction hypothesis once more we have

$$\begin{aligned} &[\![ct'']\!](\alpha, \gamma) \downarrow 1(f([\![ct']\!](\alpha, \gamma) \downarrow 2(u))) \\ &\quad \leqslant_{ct''} [\![ct'']\!](\alpha, \gamma) \downarrow 1(g([\![ct']\!](\alpha, \gamma) \downarrow 2(v))), \end{aligned}$$

which is the desired result. Faithfulness is shown in a similar way and monotonicity is straightforward. ▌

The primary use of $[\![ct]\!]\,((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N))$ is in instances when all (α_i, γ_i) are pairs of abstraction and concretization functions. Since it is customary to require these to be adjoined pairs it is of interest whether $[\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N))$ is also an adjoined pair.

PROPOSITION 11. *If ct mentions only* $\mathbf{B}_i$*'s among* $\mathbf{B}_1, ..., \mathbf{B}_N$ *and if all* $(\alpha_i, \gamma_i) \in \mathbf{IJ}_{\mathrm{f}}[\![B_i]\!]$ *are adjoined functions then*

$$\begin{aligned} u\, \boldsymbol{\alpha}_{ct}\, v &\Leftrightarrow u \leqslant_{ct} [\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 2(v) \\ &\Leftrightarrow [\![ct]\!]\,((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 1(u) \leqslant_{ct} v \end{aligned}$$

holds for all $u \in \mathbf{I}_{\mathrm{f}}[\![ct]\!]$ *and* $v \in \mathbf{J}_{\mathrm{f}}[\![ct]\!]$.

Proof. The proof is by structural induction on ct and we illustrate the case where $ct = ct' \to ct''$. As a shorthand we write (α, γ) instead of $((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N))$. The first condition becomes

$$\forall u' \in \mathbf{I}_{\mathrm{f}}[\![ct']\!],\ v' \in \mathbf{J}_{\mathrm{f}}[\![ct']\!]: \qquad u'\, \boldsymbol{\alpha}_{ct'}\, v' \Rightarrow u(u')\, \boldsymbol{\alpha}_{ct''}\, v(v'),$$

the second becomes

$$\begin{aligned} &\forall u', w' \in \mathbf{I}_{\mathrm{f}}[\![ct']\!]: \qquad u' \leqslant_{ct'} w' \\ &\quad \Rightarrow u(u') \leqslant_{ct''} [\![ct'']\!](\alpha, \gamma) \downarrow 2(v([\![ct']\!](\alpha, \gamma) \downarrow 1(w'))), \end{aligned}$$

and the third becomes

$$\begin{aligned} &\forall w', v' \in \mathbf{J}_{\mathrm{f}}[\![ct']\!]: \qquad w' \leqslant_{ct'} v' \\ &\quad \Rightarrow [\![ct'']\!](\alpha, \gamma) \downarrow 1(u([\![ct']\!](\alpha, \gamma) \downarrow 2(w'))) \leqslant_{ct''} v(v'). \end{aligned}$$

There are now four implications to show.

To prove that the first condition implies the second condition we assume the first condition and that u' and w' are such that

$$u' \leqslant_{ct'} w'.$$

By Proposition 10 we have

$$[\![ct']\!](\alpha, \gamma)\downarrow 1(u') \leqslant_{ct'} [\![ct']\!](\alpha, \gamma)\downarrow 1(w')$$

and by the induction hypothesis this is

$$u' \,\boldsymbol{\alpha}_{ct'}\, [\![ct']\!](\alpha, \gamma)\downarrow 1(w').$$

By the first condition we get

$$u(u') \,\boldsymbol{\alpha}_{ct''}\, v([\![ct']\!](\alpha, \gamma)\downarrow 1(w'))$$

and by the induction hypothesis we get

$$u(u') \leqslant_{ct''} [\![ct'']\!](\alpha, \gamma)\downarrow 2(v[\![ct']\!](\alpha, \gamma)\downarrow 1(w'))),$$

as was to be shown.

To prove that the second condition implies the first condition we assume the second condition and that u' and v' are chosen such that

$$u' \,\boldsymbol{\alpha}_{ct'}\, v'.$$

By the induction hypothesis this equals

$$u' \leqslant_{ct'} [\![ct']\!](\alpha, \gamma)\downarrow 2(v')$$

and using the second condition we get

$$u(u') \leqslant_{ct''} [\![ct'']\!](\alpha, \gamma)\downarrow 2(v([\![ct']\!](\alpha, \gamma)\downarrow 1([\![ct']\!](\alpha, \gamma)\downarrow 2(v')))).$$

Let us pause for a moment and note that

$$[\![ct']\!](\alpha, \gamma)\downarrow 2(v') \leqslant_{ct'} [\![ct']\!](\alpha, \gamma)\downarrow 2(v')$$

holds by Proposition 7 and $v' \in \mathbf{J}_{\mathrm{f}}[\![ct]\!]$ so that by the induction hypothesis

$$[\![ct']\!](\alpha, \gamma)\downarrow 1([\![ct']\!](\alpha, \gamma)\downarrow 2(v')) \leqslant_{ct'} v'.$$

But we have assumed that $v \in \mathbf{J}_{\mathrm{f}}[\![ct]\!]$ and by Proposition 10, $[\![ct'']\!](\alpha, \gamma)\downarrow 2$ is $\leqslant$-monotonic so we get

$$\begin{aligned} &[\![ct'']\!](\alpha, \gamma)\downarrow 2(v([\![ct']\!](\alpha, \gamma)\downarrow 1([\![ct']\!](\alpha, \gamma)\downarrow 2(v')))) \\ &\quad \leqslant_{ct''} [\![ct'']\!](\alpha, \gamma)\downarrow 2(v(v')). \end{aligned}$$

Continuing the argument we get by Proposition 7 that

$$u(u') \leqslant_{ct''} [\![ct'']\!](\alpha, \gamma) \downarrow 2(v(v'))$$

and by the induction hypothesis this amounts to

$$u(u')\, \boldsymbol{\alpha}_{ct''}\, v(v'),$$

as was desired.

We have now proved the equivalence of the first and second conditions. The proof that the first and third conditions are equivalent follows the same pattern and is therefore omitted. ∎

This result may be viewed as saying that when each (α_i, γ_i) is an adjoined pair then

$$[\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N))$$

is an adjoined pair with respect to the partial order of "safe approximation" (namely $\leqslant_{ct}$).

Returning to the existence of move we now have sufficient apparatus to show that the equations

$$\begin{aligned}
\text{move}_{A_i}[mv] &= \lambda l.l \\
\text{move}_{\mathbf{B}_i}[mv] &= mv_i \\
\text{move}_{ct'+ct''}[mv] &= \lambda l.\text{is}_1(l) \to \text{in}_1(\text{move}_{ct'}[mv](\text{out}_1(l))), \\
&\qquad\qquad \text{in}_2(\text{move}_{ct''}[mv](\text{out}_2(l))) \\
\text{move}_{ct' \times ct''}[mv] &= \lambda(l', l'').(\text{move}_{ct'}[mv](l'), \text{move}_{ct''}[mv](l'')) \\
\text{move}_{ct' \to ct''}[mv] &= \lambda l.\lambda ra. \bigsqcup \{\text{move}_{ct''}[mv](l(la)) \mid la\; \mathbf{mv}_{ct'}\; ra\}
\end{aligned}$$

do define a function when attention is restricted to the faithful parts of the interpretations.

THEOREM 12. *Let* **L** *be a type interpretation,* **R** *a* lattice *type interpretation, and mv a family of* strict *and* continuous *functions*

$$mv_i: \mathbf{L}(\mathbf{B}_i) \to \mathbf{R}(\mathbf{B}_i).$$

Then the equations for move make sense and define a function

$$\text{move}_{ct}[mv]: \mathbf{L}_f[\![ct]\!] \to \mathbf{R}_f[\![ct]\!],$$

that is,

strict, monotonic, $\leqslant$-monotonic, and faithful, *and satisfies*

$$u \ \mathbf{mv}_{ct} \ v \Leftrightarrow \text{move}_{ct}[mv](u) \leqslant_{ct} v$$

$$u \approx_{ct} \text{move}_{ct}[mv](u)$$

for all $u \in \mathbf{L}_{\mathrm{f}}[\![ct]\!]$ *and* $v \in \mathbf{R}_{\mathrm{f}}[\![ct]\!]$.

We leave the rather complex proof to the Appendix.
This theorem has some important implications.

COROLLARY. $\text{move}_{ct}[mv]$ *constructs the best safe approximation with respect to* $\mathbf{mv}_{ct}$.

Proof. For $u \in \mathbf{L}_{\mathrm{f}}[\![ct]\!]$ the claim that $\text{move}_{ct}[mv](u)$ is a safe approximation to u amounts to

$$u \ \mathbf{mv}_{ct} \ \text{move}_{ct}[mv](u).$$

But this follows from the double implication in the theorem by choosing $v = \text{move}_{ct}[mv](u)$ and observing that $v \leqslant_{ct} v$ follows by Proposition 7. That $\text{move}_{ct}[mv](u)$ is the best safe approximation means that all safe approximations to u also safely approximate $\text{move}_{ct}[mv](u)$, i.e.,

$$u \ \mathbf{mv}_{ct} \ v \Rightarrow \text{move}_{ct}[mv](u) \leqslant_{ct} v$$

but this follows from the double implication in the theorem. ▌

Furthermore we can compare the effects of $[\![ct]\!]$ and move_{ct}.

COROLLARY. *If* $(\alpha_i, \gamma_i)_i$ *are families of adjoined functions and ct mentions only* $\mathbf{B}_i$*'s among* $\mathbf{B}_1, ..., \mathbf{B}_N$ *then*

$$[\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 1 = \text{move}_{ct}[(\alpha_i)_i].$$

Proof. From the theorem we have, writing α for $(\alpha_i)_i$,

$$u \ \boldsymbol{\alpha}_{ct} \ v \Leftrightarrow \text{move}_{ct}[\alpha](u) \leqslant_{ct} v$$

and from Proposition 11 we have

$$u \ \boldsymbol{\alpha}_{ct} v \Leftrightarrow [\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 1(u) \leqslant_{ct} v.$$

Combining this we get

$$\text{move}_{ct}[\alpha](u) \leqslant_{ct} v \Leftrightarrow [\![ct]\!]((\alpha_1, \gamma_1), ..., (\alpha_N, \gamma_N)) \downarrow 1(u) \leqslant_{ct} v$$

and as $\leqslant_{ct}$ is a partial order the result follows. ▌

Induced Interpretations

We conclude this section by formulating the development of Abstract Interpretation in terms of interpretations. So let **L** be a faithful interpretation and $\mathbf{R}_0$ a lattice type interpretation. Here **L** may be thought of as the standard interpretation or some interpretation that expresses a program analysis whereas $\mathbf{R}_0$ is to be extended to a faithful lattice interpretation **R** that expresses some program analysis. The intended relation between **L** and **R** is based on a family *mv* of *strict* and *continuous* functions

$$mv_i \colon \mathbf{L}(\mathbf{B}_i) \to \mathbf{R}_0(\mathbf{B}_i).$$

The intention is that **R** should be a safe approximation to **L**, i.e.,

$$L \ \mathbf{mv} \ R,$$

and furthermore that **R** be as precise as possible, given that each $\mathbf{R}(\mathbf{B}_i)$ is $\mathbf{R}_0(\mathbf{B}_i)$, i.e.,

$$\mathbf{L} \ \mathbf{mv} \ \mathbf{R}' \Rightarrow \mathbf{R} \leqslant \mathbf{R}',$$

where each $\mathbf{R}'(\mathbf{B}_i)$ is also $\mathbf{R}_0(\mathbf{B}_i)$. To accomplish this we define the induced interpretation

$$\mathbf{R} = \text{induce}(\mathbf{L}, mv, \mathbf{R}_0)$$

by

$$\begin{array}{ll} \mathbf{R}(\mathbf{B}_i) = \mathbf{R}_0\,(\mathbf{B}_i) & \text{for all } \mathbf{B}_i \\ \mathbf{R}(f_i) = \text{move}_{ct_i}[mv](\mathbf{L}(f_i)) & \text{for all } f_i \text{ of type } ct_i. \end{array}$$

We then have the desired interpretation:

THEOREM 13. *Let* **L** *be a* faithful *interpretation,* $\mathbf{R}_0$ *a* lattice *type interpretation, mv a collection of* strict *and* continuous *functions, and* $\mathbf{R}'$ *a* faithful lattice *interpretation. Then*

induce $(\mathbf{L}, mv, \mathbf{R}_0)$ *is a* faithful lattice *interpretation,*

$$\mathbf{L} \ \mathbf{mv} \ \text{induce}(\mathbf{L}, mv, \mathbf{R}_0),$$

$$\mathbf{L} \ \mathbf{mv} \ \mathbf{R}' \Leftrightarrow \text{induce}(\mathbf{L}, mv, \mathbf{R}_0) \leqslant \mathbf{R}'.$$

This is a simple consequence of Theorem 12 and its first corollary and says that the induced interpretation is the best safe approximation to a given interpretation. This result is equally applicable to the definition of an analysis using the standard interpretation and to the definition of an analysis using a known analysis.

5. Applications

When applying the framework of the previous section one must make sure that only faithful interpretations need be considered. This is a semantic restriction to be fulfilled by the interpretations of the constants f_i of type ct_i. To ensure the fulfillment of this restriction it is therefore of interest to find a syntactic restriction on ct_i that will guarantee this, i.e., that will guarantee that $\mathbf{I}[\![ct_i]\!] = \mathbf{I}_f[\![ct_i]\!]$.

We therefore begin defining the predicates pure (abbreviated p), impure (abbreviated imp) and level preserving (abbreviated lp). These are defined structurally on types ct with the definitions of impure and level preserving being mutually recursive. The definitions are:

ct	$\mathrm{p}(ct)$	$\mathrm{imp}(ct)$	$\mathrm{lp}(ct)$
A_i	true	false	true
$\mathbf{B}_i$	false	true	true
$ct' \times ct''$	$\mathrm{p}(ct') \wedge \mathrm{p}(ct'')$	$\mathrm{imp}(ct') \wedge \mathrm{imp}(ct'')$	$\mathrm{lp}(ct') \wedge \mathrm{lp}(ct'')$
$ct' + ct''$	$\mathrm{p}(ct') \wedge \mathrm{p}(ct'')$	false	$\mathrm{lp}(ct') \wedge \mathrm{lp}(ct'')$
$ct' \rightarrow ct''$	$\mathrm{p}(ct') \wedge \mathrm{p}(ct'')$	$\mathrm{lp}(ct') \wedge \mathrm{imp}(ct'')$	$[\mathrm{p}(ct') \wedge \mathrm{lp}(ct'')] \vee [\mathrm{lp}(ct') \wedge \mathrm{imp}(ct'')]$

The motivation behind the definition of pure (i.e., $\mathrm{p}(ct)$) is that the type ct is pure if it contains no $\mathbf{B}_i$. The motivations for the predicates impure and level preserving are best explained by the following result.

Proposition 14. *For a type interpretation* $\mathbf{I}$ *and a type* ct *we have*

(1) *if* ct *is pure then it is level preserving and* $\approx_{ct}$ *and* $\leqslant_{ct}$ *are both equality,*

(2) *if* ct *is impure then it is level preserving and* $\approx_{ct}$ *is a relation that is always true and* $\leqslant_{ct}$ *is* $\sqsubseteq$,

(3) *if* ct *is level preserving then* $\mathbf{I}[\![ct]\!] = \mathbf{I}_f[\![ct]\!]$.

It is clear that the predicate level preserving can be used as the sufficient syntactic condition for ensuring that only faithful interpretations need be considered. Some examples of level preserving types are

$$A_1, \qquad \mathbf{B}_1, \qquad \mathbf{B}_1 \times \mathbf{B}_1 \times \mathbf{B}_1 \rightarrow \mathbf{B}_1, \qquad \mathbf{B}_1 \rightarrow (A_1 \rightarrow A_1 \rightarrow \mathbf{B}_1).$$

One may reformulate the condition for $ct' \rightarrow ct''$ to be level preserving to the requirement that ct' and ct'' be level preserving and that additionally ct' be pure or ct'' be impure. Concerning pure types one notes that they behave much as the A_i with respect to the definitions of $\approx_{ct}$ and $\leqslant_{ct}$ and, similarly, impure types behave much as the $\mathbf{B}_i$.

Proof. The proof is by structural induction on ct and we illustrate the case where $ct = ct' \rightarrow ct''$. Assume first that ct is pure, from which it follows that ct' and ct'' also are. By the induction hypothesis we have that ct'' is level preserving and this shows that ct also is. The predicate $f \approx_{ct} g$ amounts to

$$u \approx_{ct'} v \Rightarrow f(u) \approx_{ct''} g(v)$$

and since (by the induction hypothesis) both $\approx_{ct'}$ and $\approx_{ct''}$ are equality this reduces to $f = g$ so that also $\approx_{ct}$ is equality. In a similar way it is shown that $\leqslant_{ct}$ is equality.

Assume next that ct is impure but not necessarily pure. Clearly ct is also level preserving. Using the induction hypothesis the relation $f \approx_{ct} g$ amounts to

$$u \approx_{ct'} v \Rightarrow \text{true},$$

which always holds so that also $\approx_{ct}$ is a relation that is always true. Similarly $f \leqslant_{ct} g$ reduces to

$$u \leqslant_{ct'} v \Rightarrow f(u) \sqsubseteq g(v).$$

This implies that $f \sqsubseteq g$, i.e.,

$$\forall u \in \mathbf{I}[\![ct']\!] = \mathbf{I}_{\mathrm{f}}[\![ct']\!] : f(u) \sqsubseteq g(u),$$

since $\leqslant_{ct'}$ is reflexive, as follows from Proposition 7. Next suppose that $f \sqsubseteq g$ and $u \leqslant_{ct'} v$. By Proposition 7, $\mathbf{I}[\![ct']\!] = \mathbf{I}_{\mathrm{f}}[\![ct']\!]$, and monotonicity of f we get

$$f(u) \sqsubseteq f(v) \sqsubseteq g(v)$$

and this shows $f(u) \sqsubseteq g(v)$. It follows that $\leqslant_{ct}$ is $\sqsubseteq$.

Finally, assume that ct is level preserving but not necessarily pure or impure. There are two cases to consider so suppose first that ct' is pure and ct'' is level preserving. Then $f \approx_{ct} g$ reduces to

$$\forall u: \qquad f(u) \approx_{ct''} g(u)$$

as $\approx_{ct'}$ is equality and since $\approx_{ct''}$ is reflexive by Proposition 7 and the fact $\mathbf{I}[\![ct'']\!] = \mathbf{I}_{\mathrm{f}}[\![ct'']\!]$ it follows that $\approx_{ct}$ also is (over $\mathbf{I}[\![ct]\!]$). In a similar way it is shown that $\leqslant_{ct}$ is reflexive over $\mathbf{I}[\![ct]\!]$. This suffices to show that $\mathbf{I}[\![ct]\!] = \mathbf{I}_{\mathrm{f}}[\![ct]\!]$. Suppose next that ct' is level preserving and ct'' is impure. In fact ct is impure too and $\approx_{ct}$ and $\leqslant_{ct}$ are then reflexive as a consequence of (2). Again this suffices to show that $\mathbf{I}[\![ct]\!] = \mathbf{I}_{\mathrm{f}}[\![ct]\!]$. ▮

The motivation for the present work, as well as that of Nielson (1984, 1986a), is to formulate a general framework of Abstract Interpretation for denotational definitions. This means that it must be possible to formulate a wide class of program analyses using the framework. One may then consider how to implement the framework and this is currently being investigated. For this reason we shall concentrate on showing that actual analyses can be handled by showing that the setup used in various papers may be transformed to the present framework.

We begin with a brief comparison of the present development with respect to its forerunners (Nielson, 1984, 1986a). The predicate level preserving generalizes the predicate contravariantly pure used in Nielson (1984) to constrain the types ct_i of constants f_i. The definition of "contravariantly pure" (abbreviated cp) closely resembles that of "level preserving" with the exception that $\text{cp}(ct' \to ct'')$ is defined as $\text{p}(ct') \wedge \text{cp}(ct'')$. Proposition 3 is a simplified version of Theorem 3.3:14 of Nielson (1984) and a version of Proposition 7, but restricted to contravariantly pure types, is given by Lemma 4.2:3 of Nielson (1984). Finally, the definition of move generalizes the definition of view in Nielson (1984). This means that Theorem 13 is more general than its analogue in Nielson (1984) (which is Theorem 4.2:7). On the negative side we have not treated recursive domains and have only shown monotonicity and not continuity.

As a second example we consider the setup of Mycroft and Jones (1985). The language studied there is an untyped λ-calculus. Let Var be an unspecified countable set of variables x. Then the set Exp of expressions e is given by

$$e ::= x \mid \lambda x.e \mid e\ e.$$

In Mycroft and Jones (1985) the notion of an interpretation is defined, but to avoid confusion we prefer to call it a *model*. A model I is a triple consisting of cpo D and functions

$$\text{lam}\colon (D \to D) \to D$$

$$\text{app}\colon (D \times D) \to D.$$

Relative to a model I we define the cpo of environments

$$\text{Env}_D = \text{Var} \to D$$

and the semantic function

$$E_I\colon \text{Exp} \to \text{Env}_D \to D.$$

The semantic equations are

$$E_I[\![x]\!](\rho)=\rho(x)$$
$$E_I[\![\lambda x.e]\!](\rho)=\mathrm{lam}(\lambda d\in D.E_I[\![e]\!](\rho[d/x]))$$
$$E_I[\![e\ e']\!](\rho)=\mathrm{app}(E_I[\![e]\!](\rho), E_I[\![e']\!](\rho))$$

and this clearly defines E_I. In Mycroft and Jones (1985) various models are defined. One is a standard interpretation where D is the solution to the recursive domain equation

$$D=(D\rightarrow D)+Z+\{\mathrm{wrong}\}_\perp$$

and where Z is the integers. Another is a strictness analysis and the third model concerns the type analysis of Hindley and Milner.

To handle this in the present framework we must do two things: one is to define a syntactic translation from expressions e of Exp into expressions of TMLb and the other is to construct an interpretation **I** from a model I. Concerning the syntactic translation it seems natural to let $\mathbf{B}_1$ correspond to D and to let f_1 and f_2 correspond to lam and app, respectively. As for the variables x of Var there are two ways to view them: one is as elements of, e.g., A_1 and the other is to identify them with the variables of TMLb. We shall choose the former and hence Env_D will correspond to the type $A_1\rightarrow\mathbf{B}_1$. We then define the syntactic translation function $\mathscr{E}$:

$$\mathscr{E}[\![x]\!]=\lambda\rho\colon A_1\rightarrow\mathbf{B}_1.\rho(x)$$
$$\mathscr{E}[\![\lambda x.e]\!]=\lambda\rho\colon A_1\rightarrow\mathbf{B}_1.f_1(\lambda d\colon \mathbf{B}_1.\mathscr{E}[\![e]\!](\rho[d/x])),$$
$$\text{where } \rho[d/x] \text{ abbreviates } (\lambda v\colon A_1.f_3(v,x)\rightarrow d, \rho(v)))$$
$$\mathscr{E}[\![e\ e']\!]=\lambda\rho\colon A_1\rightarrow\mathbf{B}_1.f_2(\mathscr{E}[\![e]\!](\rho), \mathscr{E}[\![e']\!](\rho)).$$

Here we have assumed that ρ, d, and v are variables of TMLb and that f_3, of type $A_1\times A_1\rightarrow A_2$, where A_2 is the truth-values, stands for the test $\lambda(v,x).v=x$. To be fully precise we should have replaced the use of x on the right-hand sides by $\mathscr{V}[\![x]\!]$ where we had formally defined $\mathscr{V}[\![x_i]\!]=f_{3+i}$ for each x_i in Var. So the types ct_i of the constants f_i are

$$ct_1=(\mathbf{B}_1\rightarrow\mathbf{B}_1)\rightarrow\mathbf{B}_1$$
$$ct_2=(\mathbf{B}_1\times\mathbf{B}_1)\rightarrow\mathbf{B}_1$$
$$ct_3=A_1\times A_1\rightarrow A_2$$
$$ct_{3+i}=A_1\qquad\text{for}\quad i>0$$

and these are clearly seen to be level preserving. (In fact ct_1 and ct_2 are impure and the remaining are pure.)

Given a model $I = (D, \text{lam}, \text{app})$ we then specify an interpretation $\mathbf{I}$ by setting

$$\mathbf{I}(\mathbf{B}_1) = D$$

and, e.g., $\mathbf{I}(B_{1+i}) = D$ as well for $i \geqslant 1$, and by setting

$$\begin{aligned}
&\mathbf{I}(f_1) = \text{lam}\\
&\mathbf{I}(f_2) = \text{app}\\
&\mathbf{I}(f_3) = \lambda(u, v).u = v\\
\mathbf{I}(f_{3+i}) \quad &\text{an enumeration of the elements of Var.}
\end{aligned}$$

We take it for granted that $A_1 = \text{Var}_\perp$ and that $A_2 = \{\text{true, false}\}_\perp$. This interpretation is faithful by Proposition 14. It should be clear that $\mathscr{E}[\![e]\!]$ is always a closed expression of TMLb, i.e., no free variables, and that

$$\mathbf{I}[\![\mathscr{E}[\![e]\!]]\!](\perp) = E_I[\![e]\!](\perp).$$

One small point is that we have only required functions to be monotonic whereas in Mycroft and Jones (1985) they are in fact assumed to be continuous. Nonetheless it is correct to claim that this demonstrates that the setup of Mycroft and Jones (1985) is expressible in the present framework. As a specific example note that the Correctness Proposition of Mycroft and Jones (1985) corresponds to Proposition 3.

As the final example we consider the strictness analysis of Burn *et al.* (1986). The language studied there is a typed λ-calculus. The types t are given by the abstract syntax

$$t ::= A \mid t \to t,$$

where A is the type of atoms and $t \to t$ is a function space. The set Exp of expressions e is given by the abstract syntax

$$e ::= c^t \mid x^t \mid \lambda x^t.e \mid (e\ \ e) \mid \text{fix}^t\, e.$$

Here each c^t is a constant c of type t, each x^t is a variable x of type t, $\lambda x^t.e$ denotes function abstraction, $(e\ e)$ denotes application, and $\text{fix}^t\, e$ denotes an element of type t that is the fixed point of e. In Burn *et al.* (1986) this language is given two semantic definitions. One is

$$\text{sem}: \text{Exp} \to \text{Env} \to D,$$

which defines the ordinary semantics and where D is an infinite sum (indexed by the types t) and Env is the cpo of environments, i.e., mappings from variables to D. The other is

$$\text{tabs: Exp} \to \text{Env}' \to B,$$

which defines a strictness analysis and where also B is an infinite sum indexed by the types t and where Env$'$ contains mappings from variables to B. The definitions of sem and tabs take the form of semantic equations and the only difference is the way constants c^t are handled.

To handle this λ-calculus we follow the general approach of the previous example. In the present case we need to define two syntactic translations. The function $\mathscr{T}$ translates the types of the λ-calculus into types of TMLb. It is given by

$$\mathscr{T}[\![A]\!] = \mathbf{B}_1$$
$$\mathscr{T}[\![t_1 \to t_2]\!] = \mathscr{T}[\![t_1]\!] \to \mathscr{T}[\![t_2]\!]$$

and it is straightforward to verify that all $\mathscr{T}[\![t]\!]$ are impure and hence level preserving. Concerning expressions, we must decide how to treat variables and in this example we choose to identify them with the variables of TMLb. (To view them as elements of some type, e.g., A_1, we would have to extend TMLb with infinite sums and construct a type that is the sum of all $\mathscr{T}[\![t]\!]$.) A constant c^t will then be identified with a TMLb constant f_i of type $ct_i = \mathscr{T}[\![t]\!]$, where the index i is obtained from c. This motivates

$$\begin{aligned} \mathscr{E}[\![c^t]\!] &= f_i && \text{for } i \text{ obtained in a unique way from } c \\ \mathscr{E}[\![x^t]\!] &= x && \text{for each variable } x \\ \mathscr{E}[\![\lambda x^t . e]\!] &= \lambda x{:}\mathscr{T}[\![t]\!] . \mathscr{E}[\![e]\!] \\ \mathscr{E}[\![(e_1 e_2)]\!] &= \mathscr{E}[\![e_1]\!](\mathscr{E}[\![e_2]\!]) \\ \mathscr{E}[\![\text{fix}^t\, e]\!] &= \text{fix}_{\mathscr{T}[\![t]\!]}(\mathscr{E}[\![e]\!]). \end{aligned}$$

Clearly the expression e is closed iff $\mathscr{E}[\![e]\!]$ is and then e has type t iff $\mathscr{E}[\![e]\!]$ has type $\mathscr{T}[\![t]\!]$. In general for an expression e we define a type-environment tenv by setting

$$\text{tenv}(x) = \mathscr{T}[\![t]\!]$$

for each free x^t in e and then

$$\text{tenv} \vdash \mathscr{E}[\![e]\!] : \mathscr{T}[\![t_0]\!]$$

iff e has type t_0.

Next we define interpretations **S** and **T** corresponding to the semantic functions sem and tabs, respectively. These interpretations will be faithful, as required, because all $\mathscr{T}[\![t]\!]$ are level preserving. Concerning **S** we define $\mathbf{S}(\mathbf{B}_1)$ to be some unspecified cpo of atoms, just as in Burn *et al.* (1986), and for completeness we may interpret the $\mathbf{S}(\mathbf{B}_{1+i})$ similarly. If we write D_t for the summand of D that is indexed by t we essentially have $D_t = \mathbf{S}[\![\mathscr{T}[\![t]\!]]\!]$; the difference is that we have only insisted on monotonicity whereas Burn *et al.* (1986) insists on continuity. As for $\mathbf{S}(f_i)$ we do not specify these in detail but assume that they are interpreted as the corresponding c^i in Burn *et al.* (1986). It then follows that for a closed expression e we have

$$\text{sem}[\![e]\!](\bot) = \mathbf{S}[\![\mathscr{E}[\![e]\!]]\!](\bot)$$

and a straightforward generalization of this when e is not closed.

Concerning the strictness analysis we define $\mathbf{T}(\mathbf{B}_1)$, and for completeness also $\mathbf{T}(\mathbf{B}_{1+i})$, to be the cpo

$$\begin{tikzcd} \bullet\ 1 \arrow[d, dash] \\ \bullet\ 0 \end{tikzcd}$$

with elements 0 and 1 and ordered by $0 \sqsubseteq 1$. The intention is that 0 means "undefined" and 1 means "possibly defined." To formalize this we define

$$mv_1 : \mathbf{S}(\mathbf{B}_1) \to \mathbf{T}(\mathbf{B}_1)$$

by

$$mv_1(d) = \begin{cases} 0 & \text{if } d = \bot \\ 1 & \text{otherwise} \end{cases}$$

and similarly mv_{1+i}. Hence

$$x \ \mathbf{mv}_{\mathbf{B}_1} \ y$$

amounts to

$$y = 0 \Rightarrow x = \bot$$

and thus expresses the statement that y describes the "undefinedness" or "possibly definedness" of x. This motivates the demand that

$$\mathbf{S}[\![e]\!](\bot) \ \mathbf{mv}_{\mathscr{T}[\![t]\!]} \ \mathbf{T}[\![e]\!](\bot)$$

(for closed expressions e of type t) should express the relation between **S** and **T**. Using Theorem 13 we obtain this by defining

$$\mathbf{T} = \text{induce}\left(\mathbf{S}, mv, \begin{smallmatrix}\bullet\, 1\\ |\\ \bullet\, 0\end{smallmatrix}\right).$$

If we write B_t for the summand of B that corresponds to t we get

$$B_t = \mathbf{T}[\![\mathscr{T}[\![t]\!]]\!]$$

because monotonic functions on finite cpo's are continuous. Furthermore we have

$$\text{tabs}[\![e]\!](\bot) = \mathbf{T}[\![\mathscr{E}[\![e]\!]]\!](\bot)$$

whenever e is a closed expression and a straightforward generalization of this when e is not closed.

As an example consider an expression e of type $t = A \to A \to A$ and let us write $ce = \mathscr{E}[\![e]\!]$ and $ct = \mathscr{T}[\![tt]\!]$. Then

$$\mathbf{S}[\![ce]\!](\bot)\, \mathbf{mv}_{ct}\, \mathbf{T}[\![ce]\!](\bot)$$

holds by Theorem 13 and amounts to

$$mv_1(d) \sqsubseteq b \wedge mv_1(d') \sqsubseteq b' \Rightarrow mv_1(\mathbf{S}[\![ce]\!](\bot)(d)(d')) \sqsubseteq \mathbf{T}[\![ce]\!](\bot)(b)(b').$$

Suppose next that

$$\mathbf{T}[\![ce]\!](\bot)(0)(1) = 0.$$

Since $mv_1(d') \sqsubseteq 1$ holds for all d' and $mv_1(d) \sqsubseteq 0$ holds iff $d = \bot$ this gives

$$\forall d': \qquad \mathbf{S}[\![ce]\!](\bot)(\bot)(d') = \bot.$$

It follows that $\mathbf{S}[\![ce]\!](\bot)$ is strict in its first argument and this information may then be used as sketched in the Introduction. In Burn *et al.* (1986) the correctness of using tabs to infer strictness information about sem is expressed by the Soundness Theorem (Burn *et al.*, 1986, Theorem). A key ingredient in the proof is Lemma 8 of Burn *et al.* (1986), which corresponds to our Theorem 13 (using Proposition 3). Another ingredient in the proof is Proposition 3 of Burn *et al.* (1986), which corresponds to stating that $\text{move}_{ct' \to ct''}[mv](f)$ is strict iff f is.

6. The Collecting Semantics

The intuition behind the collecting interpretation **C** is that it is the most precise Abstract Interpretation[3] that is consistent with the standard interpretation. The consistency condition demands that **C** be *correct* with respect to **S**, i.e., that

$$\mathbf{S}\ \boldsymbol{\sigma}\ \mathbf{C}$$

for σ a suitable family of strict and continuous functions σ_i: $\mathbf{S}(\mathbf{B}_i) \to \mathbf{C}(\mathbf{B}_i)$. The preciseness condition demands that a faithful analysis **I** be correct with respect to **S** iff it is a safe approximation of **C**, i.e.,

$$(\exists\beta\colon \mathbf{S}\ \boldsymbol{\beta}\ \mathbf{I}) \Leftrightarrow (\exists\alpha\colon \mathbf{C}\ \boldsymbol{\alpha}\ \mathbf{I}),$$

where β ranges over families of strict and continuous functions and α ranges over families of lower adjoints. So far we have not assumed that a collecting interpretation exists although we have indicated that it would have $\mathbf{C}(\mathbf{B}_i) = \mathscr{P}(Z)$, or something isomorphic to it, if $\mathbf{S}(\mathbf{B}_i) = Z_\perp$. Then each α_i may be obtained from β_i by setting $\alpha_i(S) = \bigsqcup \{\beta_i(s) \mid s \in S\}$ and each β_i may be obtained from α_i by setting $\beta_i = \alpha_i \cdot \sigma_i$, where $\sigma_i(\perp) = \varnothing$ and otherwise $\sigma_i(s) = \{s\}$.

To enable a general definition of the collecting interpretation we must cover the theory of the relational power domain, $\mathscr{P}_\mathrm{R}$, also known as the lower or Hoare power domain. So let D be a cpo. A subset $X \subseteq D$ is *left-closed* iff

$$\forall d, d' \in D\colon \qquad d \sqsubseteq d' \in X \Rightarrow d \in X$$

and is *Scott-closed* iff

$$\forall Y \subseteq D\colon \qquad Y \subseteq X \wedge Y \text{ is a chain} \Rightarrow \bigsqcup Y \in X$$

and X is left-closed.

Given $X \subseteq D$ there exists a least (w.r.t. $\subseteq$) left-closed set containing X; it is called the left-closure of X and is given by

$$\mathrm{LC}(X) = \{d \in D \mid \exists d' \in X\colon\ d \sqsubseteq d'\}.$$

[3] Here the Abstract Interpretations range over those that can be proved correct by the methods of the present paper and so do not include "second-order" analyses like "live variables analysis."

Given $X \subseteq D$ there also exists a least (w.r.t. $\subseteq$) Scott-closed set containing X; it is called the Scott-closure of X and is given by

$$X^* = \bigcap \{X' \subseteq D \mid X \subseteq X' \wedge X' \text{ is Scott-closed}\}.$$

To see this note first that the intersection exists because D is a candidate for X' and therefore we do not take an empty intersection. Clearly $X \subseteq X^*$ and X^* is left-closed. To see that X^* is Scott-closed let Y be a chain such that $Y \subseteq X^*$. Then $Y \subseteq X'$ for all Scott-closed X' containing X and hence $\bigsqcup Y \in X'$ for all these X' so that $\bigsqcup Y \in X^*$.

DEFINITION. For a cpo D the relational power domain $\mathscr{P}_{\mathrm{R}}(D)$ is

$$(\{X \subseteq D \mid X \text{ Scott-closed and not empty}\}, \subseteq).$$

The associated singleton function σ: $D \to \mathscr{P}_{\mathrm{R}}(D)$ is

$$\lambda d.\{d\}^*.$$

PROPOSITION 15. *Let D be a cpo.*

(1) *$\mathscr{P}_{\mathrm{R}}(D)$ is a complete lattice with $\bigsqcup Y = (\bigcup Y)^* \cup \{\perp_D\}$ and $\perp = \{\perp_D\}$.*

(2) *$\sigma = \lambda d.\mathrm{LC}(\{d\})$ and is strict and continuous.*

(3) *Whenever β: $D \to L$ is strict and continuous and L is a complete lattice there exists precisely one completely additive function α: $\mathscr{P}_{\mathrm{R}}(D) \to L$ such that $\beta = \alpha \cdot \sigma$. It is given by $\alpha(S) = \bigsqcup \{\beta(s) \mid s \in S\}$ and is written β^σ.*

This is a version of a well-known result but for completeness a proof is given in the Appendix.

With this information we can now define the collecting interpretation **C** from a faithful standard interpretation **S**. Writing σ for the family

$$(\sigma_i\colon \mathbf{S}(B_i) \to \mathscr{P}_{\mathrm{R}}(\mathbf{S}(\mathbf{B}_i)))_i$$

of strict and continuous singleton functions we put

$$\mathbf{C} = \text{induce}(\mathbf{S}, \sigma, (\mathscr{P}_{\mathrm{R}}(\mathbf{S}(\mathbf{B}_i)))_i).$$

If $\mathbf{S}(\mathbf{B}_i) = Z_\perp$ we now have $\mathbf{C}(\mathbf{B}_i) = \{S \subseteq Z_\perp \mid \perp \in S\}$, which is isomorphic to the $\mathbf{C}(\mathbf{B}_i) = \mathscr{P}(Z)$ suggested earlier. It follows that the collecting interpretation will not distinguish between programs that produce the same set of integers as results even if one program may not terminate although the other one always does.

To prepare the ground for showing that $\mathbf{C}$ behaves as desired we first show some functor-like properties of inducing and how the $\mathbf{mv}$ relations compose.

PROPOSITION 16. *Let* $\mathbf{L}$ *be a type interpretation and let* $\mathbf{R}$ *and* $\mathbf{R}'$ *be lattice type interpretations. Furthermore, let mv be a family of strict and continuous functions*

$$mv_i\colon \mathbf{L}(\mathbf{B}_i) \to \mathbf{R}(\mathbf{B}_i)$$

and let α *be a family of lower adjoints*

$$\alpha_i\colon\ \mathbf{R}(\mathbf{B}_i) \to \mathbf{R}'(\mathbf{B}_i).$$

For all types ct we have the functor-like equations

$$\mathrm{move}_{ct}[(\lambda l.l)_i] = \lambda l.l$$
$$\mathrm{move}_{ct}[\alpha\cdot mv] = \mathrm{move}_{ct}[\alpha]\cdot\mathrm{move}_{ct}[mv],$$

where $(\alpha\cdot mv)_i = \alpha_i\cdot mv_i$. *Furthermore we have the composition law*

$$(\boldsymbol{\alpha}\cdot\mathbf{mv})_{ct} \equiv \mathbf{mv}_{ct}\cdot\boldsymbol{\alpha}_{ct}$$

and the adjoinedness-like formula

$$l(\boldsymbol{\alpha}\cdot\mathbf{mv})_{ct}\, r' \Leftrightarrow \mathrm{move}_{ct}[mv](l)\,\boldsymbol{\alpha}_{ct}\, r'$$
$$\Leftrightarrow l\,\mathbf{mv}_{ct}\,[\![ct]\!](\alpha,\gamma)\downarrow 2(r').$$

Here γ *is the family of upper adjoints corresponding to* α *and we have restricted our attention to faithful parts of interpretations.*

Proof. The first equation for move follows from Proposition 10, the second corollary to Theorem 12, and the fact that $(\lambda l.l)_i$ is a family of lower adjoints. The second equation for move is proved by a structural induction on ct and we illustrate the case where $ct = ct' \to ct''$. We write (α,γ) for $((\alpha_1,\gamma_1),\ldots,(\alpha_N,\gamma_N))$, where each γ_i is the upper adjoint corresponding to α_i and where N is the maximal index i of any $\mathbf{B}_i$ occurring in ct. We then calculate as follows,

$$\mathrm{move}_{ct}[\alpha\cdot mv](f) =$$

$$\lambda ra.\bigsqcup\{\mathrm{move}_{ct''}[\alpha\cdot mv](f(la))\mid \mathrm{move}_{ct'}[\alpha\cdot mv](la) \leqslant_{ct'} ra\} = ,$$

where we have used Theorem 12. For the next step we use the induction hypothesis

$$\lambda ra.\bigsqcup \{\text{move}_{ct''}[\alpha](\text{move}_{ct''}[mv](f(la)))\mid \text{move}_{ct'}[\alpha](\text{move}_{ct'}[mv](la)) \leqslant_{ct'} ra\} =$$

and then one of the corollaries to Theorem 12 to get

$$\lambda ra.\bigsqcup \{[\![ct'']\!](\alpha,\gamma)\downarrow 1(\text{move}_{ct''}[mv](f(la)))\mid [\![ct']\!](\alpha,\gamma)\downarrow 1(\text{move}_{ct'}[mv](la)) \leqslant_{ct'} ra\} = .$$

Using Proposition 11 we may continue

$$\lambda ra.\bigsqcup \{[\![ct'']\!](\alpha,\gamma)\downarrow 1\,(\text{move}_{ct''}[mv](f(la)))\mid \text{move}_{ct'}[mv](la) \leqslant_{ct'} [\![ct']\!](\alpha,\gamma)\downarrow 2(ra)\} = ,$$

which equals

$$\left[\lambda ra'.\bigsqcup \{[\![ct'']\!](\alpha,\gamma)\downarrow 1\,(\text{move}_{ct''}[mv](f(la)))\mid \text{move}_{ct'}[mv](la) \leqslant_{ct'} ra'\}\right]\cdot [\![ct']\!](\alpha,\gamma)\downarrow 2 = .$$

The next step is to rewrite this to

$$[\![ct'']\!](\alpha,\gamma)\downarrow 1 \cdot\left[\lambda ra'.\bigsqcup \{\text{move}_{ct''}[mv](f(la))\mid \text{move}_{ct'}[mv](la) \leqslant_{ct'} ra'\}\right] \cdot [\![ct']\!](\alpha,\gamma)\downarrow 2 =$$

and we shall justify this shortly. But having done so we may continue

$$[\![ct'']\!](\alpha,\gamma)\downarrow 1 \cdot \text{move}_{ct}[mv](f)\cdot [\![ct']\!](\alpha,\gamma)\downarrow 2 = [\![ct]\!](\alpha,\gamma)\downarrow 1\,(\text{move}_{ct}[\![mv]\!](f)),$$

which shows the result.

It now remains to show the correctness of the step that was not justified above. For this write

$$\begin{aligned} Y &= \{\text{move}_{ct''}[mv](fla)) \mid \text{move}_{ct'}[mv](la) \leqslant_{ct'} ra'\} \\ (\text{abs}, \text{con}) &= [\![ct'']\!](\alpha, \gamma) \\ Z &= \{\text{abs}(y) \mid y \in Y\}. \end{aligned}$$

Since $f \in \mathbf{L}_f[\![ct]\!]$ and $ra' \in \mathbf{R}_f[\![ct']\!]$ it follows by the reasoning in the proof of Theorem 12 that Y is a non-empty and faithful set. Similarly abs and Z are faithful, and we are claiming that

$$\text{abs}\left(\bigsqcup Y\right) = \bigsqcup Z,$$

where by Proposition 8 the $\bigsqcup$ are the least upper bound operators with respect to $\leqslant_{ct''}$. Hence we are claiming that

$$\begin{aligned} &\forall z \in Z: && z \leqslant_{ct''} \text{abs}\left(\bigsqcup Y\right) \\ &(\forall z \in Z: && z \leqslant_{ct''} u) \Rightarrow \text{abs}\left(\bigsqcup Y\right) \leqslant_{ct''} u. \end{aligned}$$

But the first result is immediate as each $z \in Z$ is of the form abs(y) for $y \in Y$ and $y \leqslant_{ct''} \bigsqcup Y$ follows by Proposition 8 and abs is $\leqslant$-monotonic by Proposition 10. Next suppose that

$$\forall z \in Z\colon\ z \leqslant_{ct''} u.$$

By Proposition 11 and the definition of Z this amounts to

$$\forall y \in Y\colon\ y \leqslant_{ct''} \text{con}(u)$$

so that

$$\bigsqcup Y \leqslant_{ct''} \text{con}(u),$$

from which we get the desired

$$\text{abs}\left(\bigsqcup Y\right) \leqslant_{ct''} u.$$

In fact we could have justified this step more tersely by just saying that $\bigsqcup$ is the least upper bound operator with respect to $\leqslant_{ct''}$ and abs is a lower adjoint by Proposition 11, and hence it is completely additive with respect to $\leqslant_{ct''}$.

We have now proved the equations for move and turn our attention towards the composition law and the adjoinedness-like formula. We calculate

$$
\begin{array}{ll}
l(\boldsymbol{\alpha}\cdot\mathbf{mv})_{ct}\, r' & \\
\Updownarrow & \text{by Theorem 12} \\
\text{move}_{ct}[\alpha\cdot mv](l)\leqslant_{ct} r' & \\
\Updownarrow & \text{by the equations for move} \\
[\![ct]\!](\alpha,\gamma)\downarrow 1(\text{move}_{ct}[mv](l))\leqslant_{ct} r' & \\
\Updownarrow & \text{by Proposition 11} \\
\text{move}_{ct}[mv](l)\leqslant_{ct}[\![ct]\!](\alpha,\gamma)\downarrow 2(r'). &
\end{array}
$$

By Theorem 12 this is equivalent to

$$l\,\mathbf{mv}_{ct}\,[\![ct]\!](\alpha,\gamma)\downarrow 2(r')$$

and by Proposition 11 to

$$\text{move}_{ct}[mv](l)\,\boldsymbol{\alpha}_{ct}\, r'$$

and this proves the adjoinedness-like formula. For the composition law we continue the calculation:

$$
\begin{array}{ll}
\text{move}_{ct}[mv](l)\leqslant_{ct}[\![ct]\!](\alpha,\gamma)\downarrow 2(r') & \\
\Updownarrow & \text{by Proposition 7} \\
\exists r:\quad \text{move}_{ct}[mv](l)\leqslant_{ct} r \wedge r\leqslant_{ct}[\![ct]\!](\alpha,\gamma)\downarrow 2(r') & \\
\Updownarrow & \text{by Proposition 11 and Theorem 12} \\
\exists r:\quad l\,\mathbf{mv}_{ct}\, r \wedge r\,\boldsymbol{\alpha}_{ct}\, r' & \\
\Updownarrow & \\
l(\mathbf{mv}_{ct}\cdot\boldsymbol{\alpha}_{ct})\, r'. &
\end{array}
$$

This completes the proof of the composition law. ▌

The intuitive contents of Proposition 16 is more clearly expressed when reformulating it for interpretations.

THEOREM 17. *Let* **L** *be a faithful interpretation, let* **R** *and* **R**$'$ *be faithful lattice interpretations, and let mv and* α *be as above. Then inducing behaves much like a functor, i.e.,*

$$\text{induce}(\mathbf{L}, (\lambda l.l)_i, (\mathbf{L}(\mathbf{B}_i))_i)=\mathbf{L}$$

$$\text{induce}(\text{induce}(\mathbf{L}, mv, (\mathbf{R}(\mathbf{B}_i))_i), \alpha, (\mathbf{R}'(\mathbf{B}_i))_i)=$$

$$\text{induce}(\mathbf{L}, \alpha\cdot mv, (R'(\mathbf{B}_i))_i).$$

Furthermore correctness and safeness compose, i.e.,

$$\mathbf{mv}' \equiv (\boldsymbol{\alpha} \cdot \mathbf{mv}) \qquad \textit{where} \quad mv' = mv \cdot \alpha$$

and inducing gives a result that is as precise as possible, i.e.,

$$\mathbf{L}(\boldsymbol{\alpha} \cdot \mathbf{mv})\, \mathbf{R}' \Leftrightarrow \mathrm{induce}(\mathbf{L}, \mathrm{mv}, (\mathbf{R}(\mathbf{B}_i))_i)\, \boldsymbol{\alpha}\, \mathbf{R}'.$$

The functor-like properties of inducing means that an Abstract Interpretation may be built by inducing in small steps and still give the same result as if induced in one big step. This supports a "stepwise coarsening" methodology for the design of Abstract Interpretations. The composition of correctness and safeness implies that if **R** is safe/correct with respect to **L** (i.e., **L mv R**) and if **R**′ is safe with respect to **R** (i.e., **R α R**′) then also **R**′ is safe/correct with respect to **L** (i.e., **L α·mv R**′). The final result of the theorem says that inducing does not lose information and it is a strengthening of the final statement in Theorem 13. The proof of Theorem 17 is an easy consequence of Proposition 16 and is therefore omitted: it merely amounts to a "componentwise" application of Proposition 16 (where the "components" are the f_i).

We can now prove the desired result about the collecting interpretation.

THEOREM 18. *The collecting interpretation* **C** *is a faithful lattice interpretation that is correct with respect to* S, *i.e.,*

$$\mathbf{S}\ \boldsymbol{\sigma}\ \mathbf{C},$$

and that is as precise as possible, i.e.,

$$(\exists \beta: \mathbf{S}\, \boldsymbol{\beta}\, \mathbf{I}) \Leftrightarrow (\exists \alpha:\ \mathbf{C}\ \boldsymbol{\alpha}\ \mathbf{I}),$$

where each β_i is strict and continuous and each α_i is a lower adjoint.

Proof. The first half of the result is a consequence of Theorem 13 so consider the second half. If **C α I** it follows from **S σ C** and Theorem 17 that **S β I** with $\beta = \alpha \cdot \sigma$. Conversely suppose that **S β I**. Define α as designated in Proposition 15, i.e., $\alpha_i = \beta_i^{\sigma_i}$, and note that α then is a family of lower adjoints. Then $\beta = \alpha \cdot \sigma$ so **S α·σ I** and the last statement in Theorem 17 gives

$$\mathrm{induce}(\mathbf{S}, \boldsymbol{\sigma}, (\mathbf{C}(\mathbf{B}_i))_i)\, \boldsymbol{\alpha}\, \mathbf{I},$$

which is the desired **C α I**. ▌

Returning to the application in Section 5 to the strictness analysis of Burn *et al.* (1986), this theorem answers a problem left open in that paper, namely how to construct a collecting semantics.

We conclude with a few examples of the collecting interpretation. Suppose that the standard interpretation has $\mathbf{S}(\mathbf{B}_1) = Z_\perp$ and interprets f_1: $\mathbf{B}_1 \times \mathbf{B}_1 \to \mathbf{B}_1$ as $\mathbf{S}(f_1) = \lambda(u, v).u + v$, where $+$ is strict in each of its arguments. Then

$$\begin{aligned} \mathbf{C}(f_1) &= \mathrm{move}_{\mathbf{B}_1 \times \mathbf{B}_1 \to \mathbf{B}_1}[\sigma](+) \\ &= \lambda(x, y).\bigsqcup \{\mathrm{move}_{\mathbf{B}_1}[\sigma](u+v) \mid \mathrm{move}_{\mathbf{B}_1 \times \mathbf{B}_1}[\sigma](u, v) \leqslant_{\mathbf{B}_1 \times \mathbf{B}_1} (x, y)\} \\ &= \lambda(x, y).\bigsqcup \{\sigma(u+v) \mid \sigma(u) \sqsubseteq x \wedge \sigma(v) \sqsubseteq y\} \\ &= \lambda(x, y).\{u+v \mid u \in x \wedge v \in y\} \end{aligned}$$

so $\mathbf{C}(f_1)$ is the element-wise application of $+$.

For a more involved example consider the function twice defined by the TMLb expression

$$\lambda x\colon \mathbf{B}_1 \to \mathbf{B}_1.\lambda y\colon \mathbf{B}_1.x(x(y)).$$

First note that

$$\begin{aligned} \mathrm{move}_{\mathbf{B}_1 \to \mathbf{B}_1}[\sigma](f) &= \lambda x.\bigsqcup \{\sigma(f(u)) \mid \sigma(u) \sqsubseteq x\} \\ &= \lambda x.\{f(u) \mid u \in x\} \cup \{\perp\}, \end{aligned}$$

which equals $\lambda x.\{f(u) \mid u \in x\}$ if f is strict. We shall write $\mathrm{P}_\mathrm{R}(f)$ for this, i.e., $\mathscr{P}_\mathrm{R}(f)(x) = \{f(u) \mid u \in x\} \cup \{\perp\}$, since this is the standard way of extending $\mathscr{P}_\mathrm{R}$ to a functor (Arbib and Manes, 1975). We then have, omitting environments,

$$\begin{aligned} \mathbf{C}[\![\mathrm{twice}]\!] &= \mathrm{move}_{(\mathbf{B}_1 \to \mathbf{B}_1) \to (\mathbf{B}_1 \to \mathbf{B}_1)}[\sigma](\mathbf{S}[\![\mathrm{twice}]\!]) \\ &= \lambda g.\bigsqcup \{\mathscr{P}_\mathrm{R}(\mathbf{S}[\![\mathrm{twice}]\!](f)) \mid \mathscr{P}_\mathrm{R}(f) \leqslant_{\mathbf{B}_1 \to \mathbf{B}_1} g\} \\ &= \lambda g.\bigsqcup \{\mathscr{P}_\mathrm{R}(f \cdot f) \mid \mathscr{P}_\mathrm{R}(f) \sqsubseteq g\} \\ &= \lambda g.\bigsqcup \{\mathscr{P}_\mathrm{R}(f) \cdot \mathscr{P}_\mathrm{R}(f) \mid \mathscr{P}_\mathrm{R}(f) \sqsubseteq g\}. \end{aligned}$$

So $\mathbf{C}[\![\mathrm{twice}]\!](g)$ combines the effects of the "primitives" less than g; this is somewhat similar to the use of the function lin in Nielson (1984), where the "primitives" are the irreducible elements.

Another way to write the result is

$$\begin{aligned} \mathbf{C}[\![\mathrm{twice}]\!] &= \lambda g.\lambda x.\{\perp\} \cup \bigcup \{\mathscr{P}_\mathrm{R}(f)(\mathscr{P}_\mathrm{R}(f)(x)) \mid \forall y\colon \mathscr{P}_\mathrm{R}(f)(y) \subseteq g(y)\} \\ &= \lambda g.\lambda x.\{\perp\} \cup \{f(f(u)) \mid u \in x \wedge \forall y\colon \mathscr{P}_\mathrm{R}(f)(y) \subseteq g(y)\} \end{aligned}$$

and one could avoid the explicit "$\{\perp\} \cup$" if g, and hence f, is strict.

Clearly $\mathbf{C}[\![\text{twice}]\!] \sqsubseteq \lambda g.g \cdot g$ but we do not have the converse inequality. To see this let

$$\begin{aligned} g(\{\perp\}) &= \{\perp\} & & \\ g(\{\perp, u\}) &= \{\perp\} & & \text{for } u \in Z \\ g(x) &= x & & \text{otherwise} \end{aligned}$$

and note that $\forall y: \mathscr{P}_{\mathrm{R}}(f)(y) \subseteq g(y)$ then reduces to $f = \perp$. Then $\mathbf{C}[\![\text{twice}]\!](g) = \perp$, which clearly differs from $g \cdot g$. This may be somewhat surprising but is just an instance of the general phenomenon in Abstract Interpretation that the induced version may be more precise than expected (see Nielson, 1986b, 1984).

7. Conclusion

The motivation behind the present work as well as that of its forerunners (Nielson, 1984, 1986a) is to formulate a general theory of program analyses that is based on Abstract Interpretation and denotational language definitions. Such a development potentially has two benefits. One is that the theoretical justification of correctness need not be performed for each analysis. We believe that our examples show that this goal has been achieved. The present development is already contained in the development of Nielson (1984, 1986a) except that the syntactic restriction "contravariantly pure" has now been weakened to the syntactic restriction "level preserving" or the semantic restriction "faithful." The other potential benefit is that it might be possible to construct a system that facilitates performing program analyses whose results are guaranteed to be correct. This possibility is currently being explored.

On a more technical side the present development should be extended to allow recursive domains; i.e., the types of TMLb should be given by

$$ct ::= A_i \mid \mathbf{B}_i \mid ct \times ct \mid ct + ct \mid ct \rightarrow ct \mid \text{rec } X.ct \mid X.$$

Before this can be accomplished we must replace monotonic function space by continuous function space as otherwise recursive domain equations need not have solutions. But this boils down to requiring $\text{move}_{ct' \rightarrow ct''}[mv]$ to preserve continuity and the remark in the Appendix shows that this does not hold in general. So one would search for conditions on the mv strong enough to guarantee this. One candidate is the backward continuity of Barbuti and Martelli (1983) but their proof, Theorem A.3, is not convincing. Another candidate might be a requirement that mv map compact (or finite or isolated; Stoy, 1977) elements to compact elements but then one would probably need to identify the compact elements in $\mathbf{I}_{\mathrm{f}}[\![ct' \rightarrow ct'']\!]$. However, the present development should be sufficiently general that we can handle a restricted version of recursive domains where rec $X.ct$ does not allow ct to contain function spaces and this would suffice for lists, trees, etc.

APPENDIX

In this appendix we give the proofs promised in the main text. The first result to be proved is Theorem 12 and to do so it is convenient first to state a strengthening of some of the results from Proposition 7.

LEMMA. *The following equations hold in general upon fathful parts of interpretations*:

$$\leqslant_{ct} \cdot \approx_{ct} = \approx_{ct} \qquad \text{i.e., } \forall u \in \mathbf{I}_f[\![ct]\!], v \in \mathbf{J}_f[\![ct]\!]:$$
$$(\exists w \in \mathbf{I}_f[\![ct]\!]: u \leqslant_{ct} w \approx_{ct} v) \Leftrightarrow u \approx_{ct} v$$
$$\approx_{ct} \cdot \leqslant_{ct} = \approx_{ct} \qquad \text{i.e., } \forall u \in \mathbf{I}_f[\![ct]\!], v \in \mathbf{J}_f[\![ct]\!]:$$
$$(\exists w \in \mathbf{J}_f[\![ct]\!]: u \approx_{ct} w \leqslant_{ct} v) \Leftrightarrow u \approx_{ct} v$$
$$\approx_{ct} \cdot \approx_{ct} = \approx_{ct} \qquad \text{i.e., } \forall u \in \mathbf{I}_f[\![ct]\!], v \in \mathbf{J}_f[\![ct]\!]:$$
$$(\exists w \in \mathbf{K}_f[\![ct]\!]: u \approx_{ct} w \approx_{ct} v) \Leftrightarrow u \approx_{ct} v.$$

Note. In Proposition 7 it was assumed that **I**, **J**, and **K** were the same.

Proof. Assume for a moment that the third equation has been proved. If $u \leqslant_{ct} w \approx_{ct} v$ we have $u \approx_{ct} w$ by Proposition 7 and hence $u \approx_{ct} v$ by the third equation. Conversely if $u \approx_{ct} v$ we have $u \leqslant_{ct} u \approx_{ct} u$ by Proposition 7. This shows that the first equation follows from the third and in a similar way it is shown that the second equation follows from the third. The third equation will be a consequence of the result

$$u \approx_{ct} w \approx_{ct} v \Rightarrow u \approx_{ct} v \qquad (*)$$

$$v \approx_{ct} [\![ct]\!](\bot, \bot) \downarrow 1(v), \qquad (\#)$$

where $(\bot, \bot)$ abbreviates $((\bot, \bot), ..., (\bot, \bot))$. To see this, note that $(*)$ is one-half of the result and that if $u \approx_{ct} v$ then

$$u \approx_{ct} v \approx_{ct} [\![ct]\!](\bot, \bot) \downarrow 1(v)$$

by $(\#)$ and we then get

$$u \approx_{ct} w \approx_{ct} v \qquad \text{for} \quad w = [\![ct]\!](\bot, \bot) \downarrow 1(v)$$

using $(*)$ and the symmetry of $\approx_{ct}$.

We prove the conjunction of $(*)$ and $(\#)$ by structural induction on *ct*.

Case $ct = A_i$. This is straightforward as $\approx_{ct}$ is $=$ and $[\![ct]\!](\bot, \bot)\downarrow 1$ is $\lambda l.l$.

Case $ct = \mathbf{B}_i$. This is straightforward as $\approx_{ct}$ is the relation that is always true.

Case $ct = ct' \times ct''$. This follows from the induction hypothesis due to the componentwise definition of $\approx_{ct}$ and $[\![ct]\!](\bot, \bot)\downarrow 1$.

Case $ct = ct' + ct''$. This follows from the induction hypothesis by case analysis, e.g., on whether $\mathrm{is}_1(v)$ is $\bot$, true, or false.

Case $ct = ct' \to ct''$. We first prove $(*)$ so suppose that

$$f \approx_{ct} g \approx_{ct} h$$

and show $f \approx_{ct} h$. For this let $u \approx_{ct'} v$ and note that for $w = [\![ct']\!](\bot, \bot)\downarrow 1(v)$ it follows from the induction hypothesis that

$$u \approx_{ct'} w \approx_{ct'} v.$$

From this we have

$$f(u) \approx_{ct''} g(w) \approx_{ct''} h(w)$$

and by $(*)$ of the induction hypothesis we get $f(u) \approx_{ct''} h(w)$. This completes the proof of $(*)$ and for $(\#)$ note that

$$[\![ct]\!](\bot, \bot)\downarrow 1(h) = [\![ct'']\!](\bot, \bot)\downarrow 1 \cdot h \cdot [\![ct']\!](\bot, \bot)\downarrow 1$$

since $[\![ct]\!](\bot, \bot)\downarrow 2 = [\![ct]\!](\bot, \bot)\downarrow 1$. To show

$$h \approx_{ct} [\![ct]\!](\bot, \bot)\downarrow 1(h)$$

let $u \approx_{ct'} v$. By the induction hypothesis $(\#)$

$$v \approx_{ct'} [\![ct']\!](\bot, \bot)\downarrow 1(v)$$

so that by $(*)$

$$u \approx_{ct'} [\![ct']\!](\bot, \bot)\downarrow 1(v).$$

Since $h \approx_{ct} h$ we have

$$h(u) \approx_{ct''} h([\![ct']\!](\bot, \bot)\downarrow 1(v))$$

and by the induction hypothesis we get

$$h(u) \approx_{ct''} [\![ct'']\!](\perp, \perp) \downarrow 1(h([\![ct']\!](\perp, \perp) \downarrow 1(v))).$$

This completes the proof of (#). ▮

THEOREM 12. *Let* **L** *be a type interpretation,* **R** *a* lattice *type interpretation, and mv a family of* strict *and* continuous *functions* $mv_i: \mathbf{L}(\mathbf{B}_i) \to \mathbf{R}(\mathbf{B}_i)$. *Then the equations*

$$\begin{aligned}
\text{move}_{A_i}[mv] &= \lambda l.l\\
\text{move}_{\mathbf{B}_i}[mv] &= mv_i\\
\text{move}_{ct \times ct'}[mv] &= \lambda(l, l').(\text{move}_{ct}[mv](l), \text{move}_{ct'}[mv](l'))\\
\text{move}_{ct + ct'}[mv] &= \lambda l.\text{is}_1(l) \to \text{in}_1(\text{move}_{ct}[mv](\text{out}_1(l))),\\
&\qquad\qquad \text{in}_2(\text{move}_{ct'}[mv](\text{out}_2(l)))\\
\text{move}_{ct \to ct'}[mv] &= \lambda l.\lambda ra.\bigsqcup\{\text{move}_{ct'}[mv](l(la)) \mid la\ \mathbf{mv}_{ct}\ ra\}
\end{aligned}$$

make sense and define a function

$$\text{move}_{ct}[mv]: \mathbf{L}_{\mathrm{f}}[\![ct]\!] \to \mathbf{R}_{\mathrm{f}}[\![ct]\!]$$

that is strict and monotonic, $\leqslant$*-monotonic, and faithful and satisfies*

$$u\ \mathbf{mv}_{ct}\ v \Leftrightarrow \text{move}_{ct}[mv](u) \leqslant_{ct} v$$

as well as

$$u \approx_{ct} \text{move}_{ct}[mv](u)$$

for all $u \in \mathbf{L}_{\mathrm{f}}[\![ct]\!]$ *and* $v \in \mathbf{R}_{\mathrm{f}}[\![ct]\!]$.

Proof. The proof will be by structural induction on *ct* but to be able to conduct the proof for function space we need a stronger induction hypothesis. This is obtained by also claiming that

$$\begin{aligned}
\text{down}_{A_i}[mv](l, r) &= r\\
\text{down}_{\mathbf{B}_i}[mv](l, r) &= l\\
\text{down}_{ct \times ct'}[mv](l, r) &= (\text{down}_{ct}[mv](l \downarrow 1, r \downarrow 1), \text{down}_{ct'}[mv](l \downarrow 2, r \downarrow 2))\\
\text{down}_{ct + ct'}[mv](l, r) &= \text{is}_1(r) \to \text{in}_1(\text{down}_{ct}[mv](\text{out}_1(l), \text{out}_1(r)),\\
&\qquad\qquad \text{in}_2(\text{down}_{ct'}[mv](\text{out}_2(l), \text{out}_2(r))\\
\text{down}_{ct \to ct'}[mv](l, r) &= \lambda la.\text{down}_{ct'}[mv](l(la), r(\text{move}_{ct}[mv](la)))
\end{aligned}$$

define a function

$$\text{down}_{ct}[mv]\colon \mathbf{L}_{\text{f}}[\![ct]\!] \times \mathbf{R}_{\text{f}}[\![ct]\!] \to \mathbf{L}_{\text{f}}[\![ct]\!]$$

that is monotonic, $\leqslant$-monotonic, and faithful in each argument and satisfies

$$l \approx_{ct} r \Rightarrow l \approx_{ct} \text{down}_{ct}[mv](l, r) \approx_{ct} r$$

as well as

$$\text{move}_{ct}[mv](l) \sqsubseteq r \Rightarrow l \sqsubseteq \text{down}_{ct}[mv](l, r) \wedge \text{down}_{ct}[mv](l, r)\, \mathbf{mv}_{ct}\, r.$$

Case $ct = A_i$. Clearly $\lambda l.l$ is a function of the stated functionality and it is strict, monotonic, and $\leqslant$-monotonic (as $\leqslant$ is $=$) and faithful (as $\approx$ is $=$). The double implication

$$u\, \mathbf{mv}_{ct}\, v \Leftrightarrow \text{move}_{ct}[\text{mv}](u) \leqslant_{ct} v$$

reduces to $u = v \Leftrightarrow u = v$, which clearly holds. Analogously

$$u \approx_{ct} \text{move}_{ct}[mv](u)$$

reduces to $u = u$, which is trivially true.

The function $\lambda(l, r).r$ has the stated functionality because $\mathbf{L}_{\text{f}}[\![A_i]\!] = \mathbf{R}_{\text{f}}[\![A_i]\!]$. Clearly it is monotonic in each argument and it is $\leqslant$-monotonic in each argument because $\leqslant$ is $=$. Similarly it is faithful in each argument as $\approx$ is $=$. The implication

$$l \approx_{ct} r \Rightarrow l \approx_{ct} \text{down}_{ct}[mv](l, r) \approx_{ct} r$$

reduces to $l = r \Rightarrow l = r = r$, which holds. Analogously the implication

$$\text{move}_{ct}[mv](l) \sqsubseteq r \Rightarrow l \sqsubseteq \text{down}_{ct}[mv](l, r) \wedge \text{down}_{ct}[mv](l, r)\, \mathbf{mv}_{ct}\, r$$

reduces to $l \sqsubseteq r \Rightarrow l \sqsubseteq r \wedge r = r$, which is also the case. (Note that this result would not hold if we had defined $\text{down}_{A_i}[mv](l, r)$ to be l.)

Case $ct = \mathbf{B}_i$. Clearly mv_i is a strict and monotonic function of the stated functionality. It is also $\leqslant$-monotonic and faithful as $\leqslant$ is $\sqsubseteq$ and $\approx$ is always true. The condition

$$u\, \mathbf{mv}_{ct}\, v \Leftrightarrow \text{move}_{ct}[mv](u) \leqslant_{ct} v$$

reduces to $mv_i(u) \sqsubseteq v \Leftrightarrow mv_i(u) \sqsubseteq v$, which clearly holds, and

$$u \approx_{ct} \text{move}_{ct}[mv](u)$$

is true because $\approx$ is constantly true.

The function $\lambda(l, r).l$ has the stated functionality and is monotonic in each argument. It is clearly $\leqslant$-monotonic in its left argument and also in its right argument. Similarly it is faithful in both arguments. The condition

$$l \approx_{ct} r \Rightarrow l \approx_{ct} \mathrm{down}_{ct}[mv](l, r) \approx_{ct} r$$

is true because $\approx$ is constantly true. The condition

$$\mathrm{move}_{ct}[mv](l) \sqsubseteq r \Rightarrow l \sqsubseteq \mathrm{down}_{ct}[mv](l, r) \wedge \mathrm{down}_{ct}[mv](l, r)\, \mathbf{mv}_{ct}\, r$$

reduces to $mv_i(l) \sqsubseteq r \Rightarrow l \sqsubseteq l \wedge mv_i(l) \sqsubseteq r$, which clearly holds. (Note that this result would not hold if we had defined $\mathrm{down}_{\mathbf{B}_i}[mv](l, r)$ to, e.g., $\perp$ in $\mathbf{L}_{\mathrm{f}}[\![\mathbf{B}_i]\!]$.)

Case $ct = ct' \times ct''$. The equation for $\mathrm{move}_{ct}[mv]$ defines a function of the stated functionality given that the induction hypothesis holds for $\mathrm{move}_{ct'}$ and $\mathrm{move}_{ct''}$. Also, $\mathrm{move}_{ct}[mv]$ is strict and monotonic, $\leqslant$-monotonic, and faithful because of the componentwise definition of $\leqslant_{ct}$ (in terms of $\leqslant_{ct'}$ and $\leqslant_{ct''}$) and $\approx_{ct}$ (in terms of $\approx_{ct'}$ and $\approx_{ct''}$). It is then easy to see that the two conditions on $\mathrm{move}_{ct}[mv]$ are indeed true.

The equation for $\mathrm{down}_{ct}[mv]$ defines a function of the stated functionality given that the induction hypothesis holds for $\mathrm{down}_{ct'}$ and $\mathrm{down}_{ct''}$. Monotonicity, $\leqslant$-monotonicity, and faithfulness in each argument follow from the induction hypothesis and the componentwise definition of $\leqslant_{ct}$ and $\approx_{ct}$. The condition

$$l \approx_{ct} r \Rightarrow l \approx_{ct} \mathrm{down}_{ct}[mv](l, r) \approx_{ct} r$$

holds by the induction hypothesis and the componentwise definition of $\approx_{ct}$. For the condition

$$\mathrm{move}_{ct}[mv](l) \sqsubseteq r \Rightarrow l \sqsubseteq \mathrm{down}_{ct}[mv](l, r) \wedge \mathrm{down}_{ct}[mv](l, r)\, \mathbf{mv}_{ct}\, r$$

note that

$$\begin{aligned} \mathrm{move}_{ct}[mv](l) \sqsubseteq r \quad & \text{iff } \mathrm{move}_{ct'}[mv](l \downarrow 1) \sqsubseteq r \downarrow 1 \\ & \text{and } \mathrm{move}_{ct''}[mv](l \downarrow 2) \sqsubseteq r \downarrow 2 \end{aligned}$$

and

$$\begin{aligned} l \sqsubseteq \mathrm{down}_{ct}[mv](l, r) \quad & \text{iff } l \downarrow 1 \sqsubseteq \mathrm{down}_{ct'}[mv](l \downarrow 1, r \downarrow 1) \\ & \text{and } l \downarrow 2 \sqsubseteq \mathrm{down}_{ct''}[mv](l \downarrow 2, r \downarrow 2) \end{aligned}$$

and

$$\text{down}_{ct}[mv](l, r)\ \mathbf{mv}_{ct}\ r \qquad \text{iff } \text{down}_{ct'}[mv](l\downarrow 1, r\downarrow 1)\ \mathbf{mv}_{ct'}\ r\downarrow 1$$
$$\text{and } \text{down}_{ct''}[mv](l\downarrow 2, r\downarrow 2)\ \mathbf{mv}_{ct''}\ r\downarrow 2$$

so that the condition follows from the inductive hypothesis.

Case $ct = ct' + ct''$. The equation for $\text{move}_{ct}[mv]$ defines a function of the stated functionality given that the induction hypothesis holds upon $\text{move}_{ct'}$ and $\text{move}_{ct''}$. Clearly the function is strict and monotonic. For $\leqslant$-monotonicity and faithfulness let Q be $\leqslant$ or $\approx$ as appropriate. If $l_1\ Q_{ct} l_2$ we consider three cases depending on l_1. When $l_1 = \bot$ also $l_2 = \bot$ so $\text{move}_{ct}[mv](l_1) = \text{move}_{ct}[mv](l_2)$ and $\text{move}_{ct}[mv](l_1)\ Q_{ct}\ \text{move}_{ct}[mv](l_2)$ follows because Q (i.e., $\leqslant$ or $\approx$) is reflexive by Proposition 7. When $l_1 = \text{in}_1(l_1')$ also $l_2 = \text{in}_1(l_2')$ for some l_2' such that $l_1'\ Q_{ct'}\ l_2'$. By the induction hypothesis we get

$$\text{move}_{ct'}[mv](l_1')\ Q_{ct'}\ \text{move}_{ct'}[mv](l_2')$$

and the result then easily follows. When $l_1 = \text{in}_2(l_1'')$ the proof is similar. The conditions

$$u\ \mathbf{mv}_{ct}\ v \Leftrightarrow \text{move}_{ct}[mv](u) \leqslant_{ct} v$$
$$u \approx_{ct} \text{move}_{ct}[mv](u)$$

are shown by a similar case analysis upon u.

The equation for $\text{down}_{ct}[mv]$ defines a function of the stated functionality given that the induction hypothesis holds for $\text{down}_{ct'}$ and $\text{down}_{ct''}$. Monotonicity in the left and right arguments are straightforward as is $\leqslant$-monotoncity in the left argument. As for $\leqslant$-monotonicity in the right argument note that $r_1 \leqslant_{ct} r_2$ implies that $\text{is}_1(r_1) = \text{is}_1(r_2)$ and the result easily follows from the induction hypothesis. A similar argument shows that it is faithful in each argument. The condition

$$l \approx_{ct} r \Rightarrow l \approx_{ct} \text{down}_{ct}[mv](l, r) \approx_{ct} r$$

is proved by cases of l, assuming that $l \approx_{ct} r$. If $l = \bot$ also $r = \bot$ and the result follows because $\text{down}_{ct}[mv](l, r) = \bot$. If $l = \text{in}_1(l')$ there is an r' such that $l' \approx_{ct'} r'$ and $r = \text{in}_1(r')$. Hence $l' \approx_{ct'} \text{down}_{ct'}[mv](l', r') \approx_{ct'} r'$ follows by the induction hypothesis. From this the result easily follows and the case $l = \text{in}_2(l'')$ is similar. Finally, assume that

$$\text{move}_{ct}[mv](l) \sqsubseteq r \qquad \text{(i)}$$

and show that

$$l \sqsubseteq \text{down}_{ct}[mv](l, r) \tag{ii}$$

$$\text{down}_{ct}[mv](l, r)\, \mathbf{mv}_{ct}\, r. \tag{iii}$$

If $l = \bot$ we clearly have (ii). Then (iii) is immediate if $r = \bot$ as then $\text{down}_{ct}[mv](l, r) = \bot$. If $r = \text{in}_1(r')$ then condition (iii) reduces to $\text{down}_{ct'}[mv](\bot, r')\, \mathbf{mv}_{ct'}\, r'$, which holds by the induction hypothesis because $\text{move}_{ct'}[mv](\bot) \sqsubseteq r'$ as $\text{move}_{ct'}[mv](\bot) = \bot$. (Note that it is here we need each mv_i to be strict.) The case $r = \text{in}_2(r'')$ is similar. Next assume $l = \text{in}_1(l')$. Then (i) implies that there is r' such that $r = \text{in}_1(r')$ and $\text{move}_{ct'}[mv](l') \sqsubseteq r'$. Hence (ii) and (iii) follow from the induction hypothesis for $\text{down}_{ct'}[mv]$. The case $l = \text{in}_2(l'')$ is similar.

Case $ct = ct' \to ct''$. We now come to the case where the assumptions about faithfulness will pay off by allowing us to show that the least upper bound in the equation for $\text{move}_{ct}[mv]$ does exist. We begin by stating an auxiliary result along the lines of the lemma preceding Theorem 12 in this appendix.

FACT. *$u\, \mathbf{mv}_{ct'}\, v \approx_{ct'} w$ implies $u \approx_{ct'} w$.*

Proof. If $u\, \mathbf{mv}_{ct'}\, v$ it follows by the induction hypothesis that $\text{move}_{ct'}[mv](u) \leqslant_{ct'} v$. By the lemma we get $\text{move}_{ct'}[mv](u) \approx_{ct'} w$ if also $v \approx_{ct'} w$. By the induction hypothesis we also have $u \approx_{ct'} \text{move}_{ct'}[mv](u)$ so that $u \approx_{ct'} w$ follows by the lemma.

Given $lf \in \mathbf{L}_f[\![ct]\!]$ and $ra \in \mathbf{R}_f[\![ct']\!]$ we must show that the set

$$Z^{lf}_{ra} = \{\text{move}_{ct''}[mv](lf(la)) \mid la\, \mathbf{mv}_{ct'}\, ra\}$$

has a least upper bound in $\mathbf{R}_f[\![ct'']\!]$. For this it is convenient to name the sets

$$Y^{lf}_{ra} = \{lf(la) \mid la\, \mathbf{mv}_{ct'}\, ra\} \subseteq \mathbf{L}_f[\![ct'']\!]$$

$$X_{ra} = \{la \mid la\, \mathbf{mv}_{ct'}\, ra\} \subseteq \mathbf{L}_f[\![ct']\!].$$

First note that X_{ra} is not empty because it contains $\text{down}_{ct'}[mv](\bot, ra)$ as follows from the induction hypothesis for $\text{down}_{ct'}$. Also, X_{ra} is a faithful set for if $la\, \mathbf{mv}_{ct'}\, ra$ we get $la \approx_{ct'} ra$ by the above fact and if also $la'\, \mathbf{mv}_{ct'}\, ra$ we have $la' \approx_{ct'} ra$ and hence $la \approx_{ct'} la'$ by the lemma and symmetry of $\approx_{ct'}$. Next Y^{lf}_{ra} is also a non-empty and faithful set because lf is faithful. Finally, Z^{lf}_{ra} is non-empty and faithful because $\text{move}_{ct''}[mv]$ is faithful. Hence $\bigsqcup Z^{lf}_{ra}$ exists in $\mathbf{R}_f[\![ct'']\!]$ by Proposition 9. Thus $\text{move}_{ct}[mv](lf)(ra)$ is well-defined and is an element of $\mathbf{R}_f[\![ct'']\!]$.

To show that $\text{move}_{ct}[mv](lf)$ is an element of $\mathbf{R}_f[\![ct]\!]$ we must show that $\lambda ra.\bigsqcup Z^{lf}_{ra}$ is monotonic, $\leqslant$-monotonic, and faithful. Suppose that $ra \sqsubseteq rb$ and that $la \in X_{ra}$. Then $\text{move}_{ct'}[mv](la) \leqslant_{ct'} ra$ so that $\text{move}_{ct'}[mv](la) \sqsubseteq ra$ by Proposition 7 and hence $\text{move}_{ct'}[mv](la) \sqsubseteq rb$. Define $lb = \text{down}_{ct'}[mv](la, rb)$ and note that $la \sqsubseteq lb$ and $lb \in X_{rb}$ follows from the induction hypothesis for $\text{down}_{ct'}$. It follows that $\bigsqcup Z^{lf}_{ra} \sqsubseteq \bigsqcup Z^{lf}_{rb}$ because lf and $\text{move}_{ct''}[mv]$ are monotonic. Next suppose that $ra \leqslant_{ct'} rb$ and that $la \in X_{ra}$. Then $la \in X_{rb}$ follows by

$$la\ \mathbf{mv}_{ct'}\ ra \Leftrightarrow \text{move}_{ct'}[mv](la) \leqslant_{ct'} ra$$

and Proposition 7. It follows that $X_{ra} \subseteq X_{rb}$ and hence $Z^{lf}_{ra} \subseteq Z^{lf}_{rb}$. Then $\bigsqcup Z^{lf}_{rb}$ is an upper bound of Z^{lf}_{ra} w.r.t. $\leqslant_{ct''}$ and hence $\bigsqcup Z^{lf}_{ra} \leqslant_{ct''} \bigsqcup Z^{lf}_{rb}$. Finally suppose that $ra \approx_{ct'} rb$. Then each $la \in X_{ra}$ and $lb \in X_{rb}$ satisfy $la \approx_{ct'} lb$ by the above fact. It is possible to choose elements $rx \in Z^{lf}_{ra}$ and $ry \in Z^{lf}_{rb}$ and it follows that $rx \approx_{ct''} ry$. As $rx \approx_{ct''} \bigsqcup Z^{lf}_{ra}$ and $ry \approx_{ct''} \bigsqcup Z^{lf}_{rb}$ by Proposition 8 we get $\bigsqcup Z^{lf}_{ra} \approx_{ct''} \bigsqcup Z^{lf}_{rb}$ by Proposition 7.

Clearly $\text{move}_{ct}[mv]$ is a strict function. For monotonicity suppose that $lf \sqsubseteq lg$. Then for all ra

$$\forall x \in Y^{lf}_{ra}\colon \exists y \in Y^{lg}_{ra}\colon \qquad x \sqsubseteq y$$

and a similar condition relates Z^{lf}_{ra} and Z^{lg}_{rb}. It follows that $\lambda ra.\bigsqcup Z^{lf}_{ra} \sqsubseteq \lambda ra.\bigsqcup Z^{lg}_{ra}$. For $\leqslant$-monotonicity suppose that $lf \leqslant_{ct} lg$ and that $ra \leqslant_{ct'} rb$. Then

$$\forall x \in X_{ra}\colon \exists y \in X_{rb}\colon \qquad x \leqslant_{ct'} y$$

and a similar condition relates Z^{lf}_{ra} and Z^{lg}_{rb}. But then $\bigsqcup Z^{lg}_{rb}$ is an upper bound of Z^{lf}_{ra} w.r.t. $\leqslant_{ct''}$ and $\bigsqcup Z^{lf}_{ra} \leqslant_{ct''} \bigsqcup Z^{lg}_{rb}$ follows. This shows $\lambda ra.\bigsqcup Z^{lf}_{ra} \leqslant_{ct} \lambda ra.\bigsqcup Z^{lg}_{ra}$. Finally, for faithfulness, suppose that $lf \approx_{ct} lg$ and that $ra \approx_{ct'} rb$. Then

$$\forall x \in X_{ra}\colon \exists y \in X_{rb}\colon \qquad x \approx_{ct'} y$$

and a similar condition relates Z^{lf}_{ra} and Z^{lg}_{rb}. It follows that $\bigsqcup Z^{lf}_{ra} \approx_{ct''} \bigsqcup Z^{lf}_{rb}$ and hence $\lambda ra.\bigsqcup Z^{lf}_{ra} \approx_{ct} \lambda ra\,.\bigsqcup Z^{lg}_{ra}$.

For the implication concerning $\text{move}_{ct}[mv]$ we note first that $lf\ \mathbf{mv}_{ct}\ rf$ amounts to

$$la\ \mathbf{mv}_{ct'}\ rb \Rightarrow \text{move}_{ct''}[mv](lf(la)) \leqslant_{ct''} rf(rb),$$

using the induction hypothesis for $\text{move}_{ct''}$. Similarly $\text{move}_{ct}[mv](lf) \leqslant_{ct} rf$ amounts to

$$ra \leqslant_{ct'} rb \Rightarrow (la\ \mathbf{mv}_{ct'}\ ra \Rightarrow \text{move}_{ct''}[mv](lf(la)) \leqslant_{ct''} rf(rb)),$$

using the fact that $\bigsqcup\{...\}$ is the least upper bound of $\{...\}$ w.r.t. $\leqslant_{ct''}$ whenever $\{...\}$ is non-empty and faithful. It is then immediate that the second condition implies the first as one may choose $ra = rb$. That the first condition implies the second follows from

$$la\ \mathbf{mv}_{ct'}\ ra \leqslant_{ct'} rb \Rightarrow la\ \mathbf{mv}_{ct'}\ rb,$$

which was shown previously.

Finally $lf \approx_{ct} \text{move}_{ct}[mv](lf)$ may be shown as follows. Let $la \approx_{ct'} ra$ and choose lb such that $lb\ \mathbf{mv}_{ct'}\ ra$. Then

$$la \approx_{ct'} lb$$

by a previous fact. Then

$$lf(la) \approx_{ct''} lf(lb)$$

and by the induction hypothesis for $\text{move}_{ct''}[mv]$ we get

$$lf(la) \approx_{ct''} \text{move}_{ct''}[mv](lf(lb)).$$

But $\text{move}_{ct}[mv](lf)(ra) \approx_{ct''} \text{move}[mv](lf(lb))$ follows by Proposition 8 so that

$$lf(la) \approx_{ct''} \text{move}_{ct}[mv](lf)(ra),$$

as was to be shown.

We now turn our attention towards down_{ct}. Clearly $\text{down}_{ct}[mv](lf, rf)(la)$ is well-defined and is an element of $\mathbf{L}_f[\![ct'']\!]$. To prove that $\text{down}_{ct}[mv](lf, rf)$ is an element of $\mathbf{L}_f[\![ct]\!]$ we must show that

$$\lambda la.\text{down}_{ct''}[mv](lf(la), rf(\text{move}_{ct'}[mv](la)))$$

is monotonic, $\leqslant$-monotonic, and faithful. Let Q be any one of $\sqsubseteq$, $\leqslant$, or $\approx$. If $la\ Q_{ct'}\ lb$ then

$$lf(la)\ Q_{ct''}\ lf(lb)$$

by assumptions about lf, and

$$rf(\text{move}_{ct'}[mv](la))\ Q_{ct''}\ rf(\text{move}_{ct'}[mv](lb))$$

by assumptions about rf and $\text{move}_{ct'}[mv]$. Since $\text{down}_{ct''}[mv]$ preserves Q in each argument we have

$$\text{down}_{ct''}[mv](lf(la), rf(\text{move}_{ct'}[mv](la)))$$
$$Q_{ct''}$$
$$\text{down}_{ct''}[mv](lf(lb), rf(\text{move}_{ct'}[mv](la)))$$
$$Q_{ct''}$$
$$\text{down}_{ct''}[mv](lf(lb), rf(\text{move}_{ct'}[mv](lb))),$$

from which the result follows as $Q_{ct''}$ is transitive.

Next we must show that $\text{down}_{ct}[mv]$ is monotonic, $\leqslant$-monotonic, and faithful in each argument. Let Q be $\sqsubseteq$, $\leqslant$, or $\approx$ as appropriate and suppose that $lf\ Q_{ct}\ lg$ and $la\ Q_{ct'}\ lb$. It was argued above that

$$\text{down}_{ct''}[mv](lf(la), rf(\text{move}_{ct'}[mv](la)))$$
$$Q_{ct''}$$
$$\text{down}_{ct''}[mv](lf(lb), rf(\text{move}_{ct'}[mv](lb))).$$

Since $\text{down}_{ct''}$ preserves Q in its left argument we may extend this with

$$Q_{ct''}$$
$$\text{down}_{ct''}[mv](lg(lb), rf(\text{move}_{ct'}[mv](lb))).$$

Since Q is transitive it follows that $\text{down}_{ct}[mv]$ preserves Q in its left argument. Preservation in its right argument is shown in a similar way.

Suppose next that $lf \approx_{ct} rf$ and let us show that

$$lf \approx_{ct} \text{down}_{ct}[mv](lf, rf) \approx_{ct} rf.$$

Actually,, by the lemma it suffices to prove the left half. So suppose that $la \approx_{ct'} lb$. Then

$$lb \approx_{ct'} \text{move}_{ct'}[mv](lb)$$

by the inductive hypothesis so that

$$lf(lb) \approx_{ct''} rf(\text{move}_{ct'}[mv](lb)).$$

Hence

$$lf(lb) \approx_{ct''} \text{down}_{ct''}[mv](lf(lb), rf(\text{move}_{ct'}[mv](lb)))$$

follows by the induction hypothesis. Since $la \approx_{ct'} lb$ we get $lf(la) \approx_{ct''} lf(lb)$ so that

$$lf(la) \approx_{ct''} \text{down}_{ct''}[mv](lf(lb), rf(\text{move}_{ct'}[mv](lb))),$$

as was to be shown.

Finally suppose that $\mathrm{move}_{ct}[mv](lf) \sqsubseteq rf$, i.e., that

$$\mathrm{move}_{ct'}[mv](la) \leqslant_{ct'} ra \Rightarrow \mathrm{move}_{ct''}[mv](lf(la)) \sqsubseteq rf(ra).$$

By setting $ra = \mathrm{move}_{ct'}[mv](la)$ and using the fact that $\leqslant_{ct'}$ is reflexive we have

$$\mathrm{move}_{ct''}[mv](lf(la)) \sqsubseteq rf(\mathrm{move}_{ct'}[mv](la)). \qquad (*)$$

Using the induction hypothesis we have

$$lf(la) \sqsubseteq \mathrm{down}_{ct''}[mv](lf(la), rf(\mathrm{move}_{ct'}[mv](la))).$$

Since this holds for all $la \in \mathbf{L}_{\mathrm{f}}[\![ct']\!]$ we have

$$lf \sqsubseteq \mathrm{down}_{ct}[mv](lf, rf)$$

and this was the first conjunct we had to show. To show the other conjunct we assume that

$$\mathrm{move}_{ct'}[mv](la) \leqslant_{ct'} ra \qquad (**)$$

and must show

$$\mathrm{move}_{ct''}[mv](\mathrm{down}_{ct''}[mv](lf(la), rf(\mathrm{move}_{ct'}[mv](la))) \leqslant_{ct''} rf(ra).$$

From $(*)$ and the induction hypothesis we get

$$\begin{aligned}&\mathrm{move}_{ct''}[mv](\mathrm{down}_{ct''}[mv](lf(la), rf(\mathrm{move}_{ct'}[mv](la))))\\ &\quad \leqslant_{ct''} rf(\mathrm{move}_{ct'}[mv](la))\end{aligned}$$

but by $(**)$ we may continue

$$\leqslant_{ct''} rf(ra),$$

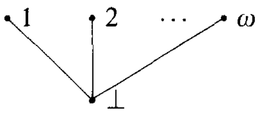
which shows the result. ▌

Remark. To stay within the traditional theory of denotational semantics we should have worked with continuous functions rather than monotonic functions. In particular we should show that $\mathrm{move}_{ct}[mv](f)$ is continuous whenever f is. Unfortunately this need not be so. For an example of this let

$$ct = \mathbf{B}_1 \rightarrow \mathbf{B}_2,$$

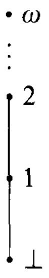
where

$$\mathbf{L}(\mathbf{B}_1) = \{1, 2, ..., \omega\}_\perp =$$



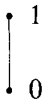
and

$$\mathbf{R}(\mathbf{B}_1) = (\{\perp, 1, 2, ..., \omega\}, \leqslant) =$$



and

$$\mathbf{L}(\mathbf{B}_2) = \mathbf{R}(\mathbf{B}_2) =$$



Furthermore, let

$$mv_i = \lambda x.x$$

$$f = \lambda x.(x = \omega) \rightarrow 1, 0 \quad \in \mathbf{L}(\mathbf{B}_1) \rightarrow \mathbf{L}(\mathbf{B}_2)$$

and note that this defines strict and continuous functions. Then

$$\text{move}_{ct}[mv](f) = \lambda x.(x = \omega) \rightarrow 1, 0 \quad \in \mathbf{R}(\mathbf{B}_1) \rightarrow \mathbf{R}(\mathbf{B}_2),$$

which is not continuous. (In the case where all $\mathbf{R}(\mathbf{B}_i)$ are finite we do have continuity, as is implied by monotonicity.)

PROPOSITION 15. *Let D be a cpo.*

(1) *$\mathscr{P}_\mathrm{R}(D)$ is a complete lattice with $\bigsqcup \mathscr{Y} = (\bigcup \mathscr{Y})^* \cup \{\perp_D\}$ and $\perp = \{\perp_D\}$.*

(2) *$\sigma = \lambda d.\mathrm{LC}(\{d\})$ and is strict and continuous.*

(3) *Whenever $\beta: D \rightarrow L$ is strict and continuous and L is a complete lattice there exists precisely one completely additive function $\alpha: \mathscr{P}_\mathrm{R}(D) \rightarrow L$ such that $\beta = \alpha \cdot \sigma$. It is given by $\alpha(S) = \bigsqcup \{\beta(s) \mid s \in S\}$ and is written β^σ.*

Proof. We first show that $\mathscr{P}_R(D)$ is a complete lattice. For this let $\mathscr{Y}$ be some subset of $\mathscr{P}_R(D)$. Clearly $(\bigcup \mathscr{Y})^* \cup \{\perp_D\}$ is an element of $\mathscr{P}_R(D)$ and is an upper bound of $\mathscr{Y}$. Next let $Y \in \mathscr{P}_R(D)$ be an upper bound of $\mathscr{Y}$. We have $\bigcup \mathscr{Y} \subseteq Y$ and $\perp_D \in Y$ so that $(\bigcup \mathscr{Y})^* \cup \{\perp_D\} \subseteq Y$. This shows that $\mathscr{P}_R(D)$ is a complete lattice with $\bigsqcup \mathscr{Y}$ as stated. Then also $\perp = \bigsqcup \varnothing = \{\perp_D\}$ follows.

To see that $\sigma = \lambda d.\text{LC}(\{d\})$ note that $\text{LC}(\{d\})$ is already Scott-closed: if $Y \subseteq \text{LC}(\{d\})$ is a chain then d is an upper bound of Y and hence $\bigsqcup Y \sqsubseteq d$ so that $\bigsqcup Y \in \text{LC}(\{d\})$. To see that σ is continuous let $Y \subseteq D$ be a non-empty chain. Then

$$\bigsqcup \{\sigma(y) \mid y \in Y\} = \left(\bigcup_{y \in Y} \text{LC}(\{y\})\right)^*$$

and we must show

$$\left(\bigcup_{y \in Y} \text{LC}(\{y\})\right)^* = \text{LC}\left(\left\{\bigsqcup Y\right\}\right).$$

Clearly $\subseteq$ holds so consider $\supseteq$. It suffices to show that

$$\bigsqcup Y \in \left(\bigcup_{y \in Y} \text{LC}(\{y\})\right)^*$$

and this follows from

$$Y \subseteq \bigcup_{y \in Y} \text{LC}(\{y\})$$

and the Scott-closedness of $(\bigcup_{y \in Y} \text{LC}(\{y\}))^*$. That σ is strict is evident.

Finally, let L be a complete lattice, let β: $D \to L$ be a strict and continuous function, and let α: $\mathscr{P}_R(D) \to L$ be as stated in (3). That α is completely additive amounts to showing

$$\alpha\left(\bigsqcup \mathscr{Y}\right) = \bigsqcup \{\alpha(Y) \mid Y \in \mathscr{Y}\}$$

for $\mathscr{Y} \subseteq \mathscr{P}_R(D)$. Since β is strict we have

$$\alpha\left(\bigsqcup \mathscr{Y}\right) = \bigsqcup \left\{\beta(s) \mid s \in \left(\bigcup \mathscr{Y}\right)^*\right\}$$

and we have

$$\bigsqcup \{\alpha(Y) \mid y \in \mathscr{Y}\} = \bigsqcup \left\{\bigsqcup \{\beta(s) \mid s \in Y\} \mid Y \in \mathscr{Y}\right\}$$
$$= \bigsqcup \{\beta(s) \mid s \in Y \in \mathscr{Y}\}$$
$$= \bigsqcup \{\beta(s) \mid s \in \bigcup \mathscr{Y}\}.$$

From this $\alpha(\bigsqcup \mathscr{Y}) \sqsupseteq \bigsqcup \{\alpha(Y) \mid Y \in \mathscr{Y}\}$ easily follows so consider $\alpha(\bigsqcup \mathscr{Y}) \sqsubseteq \bigsqcup \{\alpha(Y) \mid Y \in \mathscr{Y}\}$. It suffices to show that

$$\beta(s) \sqsubseteq \bigsqcup \left\{\beta(s') \mid s' \in \bigcup \mathscr{Y}\right\}$$

whenever $s \in (\bigcup \mathscr{Y})^*$. It follows from Markowsky (1976, Sect. 6) that $(\bigcup \mathscr{Y})^* = W_\delta$ for some ordinal number δ where

$$W_0 = \text{LC}\left(\bigcup \mathscr{Y}\right)$$
$$W_\lambda = \text{LC}\left(\left\{\bigsqcup C \mid C \text{ is a chain and } C \subseteq U_{\gamma<\lambda} W_\gamma\right\}\right) \qquad \text{for} \quad \lambda > 0.$$

It thus suffices to show that

$$s \in W_\lambda \Rightarrow \beta(s) \sqsubseteq \bigsqcup \left\{\beta(s') \mid s' \in \bigcup \mathscr{Y}\right\}$$

by transfinite induction on λ. For $\lambda = 0$ this is evident and for $\lambda > 0$ we have

$$\beta(s) \sqsubseteq \beta\left(\bigsqcup C\right) = \bigsqcup \{\beta(s') \mid s' \in C\} \qquad \text{for } C \text{ as above}$$
$$\sqsubseteq \bigsqcup \left\{\beta(s') \mid s' \in \bigcup_{\gamma<\lambda} W_\gamma\right\}$$
$$\sqsubseteq \bigsqcup \left\{\beta(s') \mid s' \in \bigcup \mathscr{Y}\right\}$$

This proves that α is completely additive. To see that $\beta = \alpha \cdot \sigma$ note that

$$\alpha(\sigma(d)) = \bigsqcup \{\beta(s) \mid s \sqsubseteq d\} = \beta(d).$$

It remains to show that $\alpha' = \alpha$ whenever α' is a completely additive function such that $\alpha' \cdot \sigma = \beta$. For $Y \in \mathscr{P}_{\mathrm{R}}(D)$ we calculate

$$\begin{aligned}
\alpha'(Y) &= \alpha'\left(\bigsqcup \{\mathrm{LC}(\{y\}) \mid y \in Y\}\right) \\
&= \alpha'\left(\bigsqcup \{\sigma(y) \mid y \in Y\}\right) \\
&= \bigsqcup \{\alpha'(\sigma(y)) \mid y \in Y\} \\
&= \bigsqcup \{\beta(y) \mid y \in Y\} \\
&= \alpha(Y)
\end{aligned}$$

and this shows the result. ▮

Acknowledgments

This research is part of the PSI-project supported by the Danish Natural Science Research Council and most of the work reported here was performed while the author was at the Institute of Electronic Systems at Aalborg University Centre. The present version benefits from the useful comments of the referees.

Received: July 23, 1986; accepted: June 12, 1987

References

Abramsky, S. (1985), Strictness analysis via logical relations, unpublished manuscript.

Arbib, M. A., and Manes, E. G. (1975), "Arrows, Structures and Functors: The Categorical Imperative," Academic Press, Orlano, FL.

Barbuti, R., and Martelli, A., (1983), A structured approach to static semantics correctness, *Sci. Comput. Programming* **3**, 279–311.

Burn, G. L., Hankin, C. L. and Abramsky, S. (1986), Strictness analysis for higher order functions, *Sci. Comput. Programming* **7**, 249–278; also see Report 85/6, Department of Computing, Imperial College, 1985.

Cousot, P., and Cousot, R. (1977), Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, *in* "Conf. Record, 4th ACM Symposium on Principles of Programming Languages."

Cousot, P., and Cousot, R. (1979), Systematic design of program analysis frameworks, *in* "Conf. Record, 6th ACM Symposium on Principles of Programming Languages."

Gordon, M. J. C. (1979), "The Denotational Description of Programming Languages: An Introduction," Springer-Verlag, New York/Berlin.

Halmos, P. R. (1960), "Naive Set Theory," Van Nostrand, Princeton, NJ.

HUGHES, J. (1986), Strictness detection in non-flat domains, *in* "Proceedings, 'Programs as Data Objects,'" Lecture Notes in Computer Science, Vol. 217, Springer-Verlag, New York/Berlin.

JONES, N. D. AND MUCHNICK, S. S. (1981), Complexity of flow analysis, inductive assertion synthesis and a language due to Dijkstra, *in* "Program Flow Analysis: Theory and Applications" (S. S. Muchnick and N. D. Jones, Eds.), Prentice–Hall, Englewood Cliffs, NJ.

JONES, N. D., AND MYCROFT, A. (1986), Data flow analysis of applicative programs using minimal function graphs, *in* "Conf. Record, 13th ACM Symposium on Principles of Programming Languages."

MACLANE, S. (1971), "Categories for the Working Mathematician," Springer-Verlag, New York/Berlin.

MARKOWSKY, G. (1976), Chain-complete posets and directed sets with applications, *Algebra Universalis* **6**, 53–68.

MAURER, D. (1986), Strictness computation using generalized λ-expressions, *in* "Proceedings, 'Programs as Data Objects,'" Lecture Notes in Computer Science, Vol. 217, Springer-Verlag, New York/Berlin.

MILNE, R., AND STRACHEY, C. (1976), "A Theory of Programming Language Semantics," Chapman & Hall, London.

MOSSES, P. D. (1979), "SIS–Semantics Implementation System: Reference Manual and User Guide," DAIMI Report No. MD-30, Denmark.

MYCROFT, A. (1981), "Abstract Interpretation and Optimizing Transformations for Applicative Programs," Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.

MYCROFT, A., AND JONES, N. D. (1986), A relational framework for abstract interpretation, *in* "Proceedings, 'Programs as Data Objects,'" Lecture Notes in Computer Science, Vol. 217, Springer-Verlag, New York/Berlin.

MYCROFT, A., AND NIELSON, F. (1983), Strong abstract interpretation using power domains, *in* "Proceedings, 10th ICALP," Lecture Notes in Computer Science, Vol. 154, Springer-Verlag, New York/Berlin.

NIELSON, F. (1982), A denotational framework for data flow analysis, *Acta Inform.* **18**.

NIELSON, F. (1984), "Abstract Interpretation Using Domain Theory," Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.

NIELSON, F. (1986a), Abstract interpretation of denotational definitions, *in* "Proceedings, STACS 1986," Lecture Notes in Computer Science, Vol. 210, Springer-Verlag, New York/Berlin.

NIELSON, F. (1986b), Expected forms of data flow analyses, *in* "Proceedings, 'Programs as Data Objects," Lecture Notes in Computer Science, Vol. 217; Springer-Verlag, New York/Berlin.

NIELSON, F. (1986c), "Strictness Analysis and Denotational Abstract Interpretation," Research Report R86-9A, Aalborg University Centre.

NIELSON, F. (1987), Strictness analysis and denotational abstract interpretation (extended abstract), *in* "Proceedings, 14th POPL, ACM."

PLOTKIN, G. D. (1973), Lambda-definability and logical relations, Edinburgh AI memo.

REYNOLDS, J. C. (1974), On the relation between direct and continuation semantics, *in* "Proceedings, 2nd ICALP," Lecture Notes in Computer Science, Vol. 14, Springer-Verlag, New York/Berlin.

J. E. STOY, (1977), "Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory," MIT Press, Cambridge, MA.

Printed in Belgium