INFORMATION AND COMPUTATION 99, 154–177 (1992)

Strictness Analysis via Abstract Interpretation for Recursively Defined Types

GERARD R. RENARDEL DE LAVALETTE

Department of Computing Science, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands

In this paper we consider a functional language with recursively defined types and a weak form of polymorphism. For this language a strictness analysis is developed, based on abstract interpretation in a category of complete algebraic lattices. © 1992 Academic Press, Inc.

1. INTRODUCTION

A function $f: X \to Y$ (X, Y partial orderings with bottom element $\perp$) is called *strict* if $f\perp = \perp$. Strictness is the semantical counterpart of *neededness*, defined as follows: subterm s (not in normal form) of term t is called needed if every reduction of t to normal form contains a contraction within s. (See Barendregt *et al.* (1986) for more information about neededness.)

Strictness analysis is an analysis technique for lazy functional programs. It is intended to improve efficiency of execution by detecting strict functions and strict subterms: this information can be used to replace lazy call-by-need evaluation by call-by-value evaluation. This is attractive since call-by-value evaluation is more efficient and can safely be done in parallel.

Abstract interpretation, first presented by Cousot and Cousot (1977, 1979) is a general theory for static analysis of programs. The idea behind it is to simplify the intended denotational interpretation of the language in question to an abstract interpretation, in such a way that the resulting denotation of a program preserves the property to be analysed and can be determined quickly. Several types of analysis lend themselves for this approach; besides strictness analysis one may think of type verification, program correctness, and termination.

A frequently studied application of abstract interpretation is strictness analysis. The first work in this direction was performed by Myeroft (1980, 1981) for flat domains and functions defined by first-order recursion equations. A generalisation to higher-order functions (with a type structure characterised by $\tau ::= c \mid \tau \to \tau$) is developed in Burn *et al.* (1986). Abramsky (1985) contains a generalisation to polymorphic types ($\tau ::= c \mid \alpha \mid \tau \to \tau$),

0890-5401/92 $5.00

obtained by adding free type variables: via an operational semantics, it is shown that strictness analysis is a polymorphic invariant. Generic polymorphism, obtained by adding type schemes where quantification over type variables is allowed (types $\tau ::= c \mid \alpha \mid \tau \to \tau$, type schemes $\sigma ::= \tau \mid \forall \alpha \cdot \sigma$), is briefly discussed in a remark at the end of Abramsky (1985).

A more general treatment of abstract interpretations is found in the work of Nielson (1984, 1986, 1988). In (Nielson, 1988), a general theory of abstract interpretations is developed for a language TMLb (with types $\tau ::= c \mid \alpha \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \tau$) which is comparable to the language in Burn *et al.* (1986). The strictness analysis of (Burn *et al.*, 1986) is treated in (Nielson, 1988) as an example. A difference between the denotational semantics used in (Burn *et al.*, 1986; Nielson, 1988) lies in the function spaces used for the interpretation of types $\sigma \to \tau$: in (Burn *et al.*, 1986) the functions are continuous, in (Nielson, 1988) they are monotonic. This has to do with the fact that the abstraction function for types $\sigma \to \tau$ does not preserve continuity, as is shown in a counterexample (Nielson, 1988, p. 88), but only monotonicity. In Lemma 4.4(i) of this paper, we show that continuity can be preserved, provided the abstraction functions preserve compactness; this settles a conjecture implicitly formulated in the Conclusion of (Nielson, 1988, p. 76).

In this paper, we extend strictness analysis, based on abstract interpretation in the style of (Burn *et al.*, 1986; Nielson, 1988) to a language with polymorphic and recursively defined types ($\tau ::= \alpha \mid \mu\Phi$; type constructors $\Phi ::= K \mid \to \mid \tau \mid \Phi\tau \mid \delta\alpha.\Phi$). The semantics of types $\mu\Phi$ (Φ a unary type constructor) is obtained via a well-known limit construction in the category of domains. The definition of the abstraction functions, which are used to define the relation between the concrete (i.e., intended) and the abstract interpretation, is rather sensitive: their properties should be preserved under the various operations on types, and at the same time guarantee some desired properties of the abstract interpretation defined in terms of abstractions. We use here the following definition (see 4.1.(i)) for abstractions f:

f is $\perp$-unique, preserves compactness, and has a continuous right inverse.

We give a survey of what follows. In Section 2, some preliminaries on cpo's, complete lattices, categories, and limit constructions are presented, with references for more details. The definition of the language **L** appears in Section 3.1 in the form of a context-free syntax and a derivation system for wellformedness, where the collection of constants of **L** acts as a parameter. A general definition of interpretations of **L** in some category of domains is given in Section 3.2, with the intended interpretation C of **L** as

an instance. In Section 4 another instance, the abstract interpretation A, is derived using the type interpretation abs (which is *almost* an interpretation in the sense of Section 3.2). Section 4 contains theory about abstractions and related notions and ends with the final theorem (4.17) which implies directly that strictness analysis based on A is safe. Section 5 concludes the paper with some remarks on variants of the methods used here and some suggestions for further work.

Before we go to work, a final remark on practical aspects of strictness analysis. It is not hard to see that exact strictness analysis is as difficult as evaluation and therefore not interesting for static analysis intended for evaluation speedup. The approximative strictness analysis for a language with simple types as given, e.g., in Burn *et al.* (1986) is based on effective computations in finite complete lattices. However, as has been pointed out by several authors (see Abramsky, 1985; Hughes, 1987), this does not guarantee a feasible strictness analysis since it may take an amount of time which is exponential in the size of the program. From a computational point of view, the situation with respect to the strictness analysis developed here is even worse: due to the limit construction for the interpretation of the fixpoint constructor μ, the domains of the abstract interpretations are not even finite. So this strictness analysis does not lead directly to an algorithm, but should be seen as a theoretical basis upon which more efficient methods of analysis can be developed: not only strictness analysis, but also, e.g., backwards analysis in the sense of Hughes (1987).

2. Preliminaries

We list some well established facts about complete partial orderings, categories, and limit constructions. For more information and proofs we refer to the references given in the subsections and to Renardel de Lavalette (1988).

2.1. *Complete Partial Orderings*

(See Scott (1976, 1982) for more information.) $X = \langle X, \leqslant \rangle$ is a *complete partial ordering* (cpo) if it is a partial ordering with a least element $\bot$ where every $Y \in D(X)$ has a supremum (least upper bound) $\bigvee Y$. Here $D(X) = \{ Y \subseteq X \mid Y \neq \varnothing \wedge \forall xy \in Y \exists z \in Y (x \leqslant z \wedge y \leqslant z) \}$ is the collection of directed subsets of X.

Let X, Y be cpo's with $f: X \to Y$. f is called *strict* if $f\bot = \bot$, $\bot$-*unique* if $\forall x \in X (fx = \bot \leftrightarrow x = \bot)$, *continuous* if $f(\bigvee Z) = \bigvee f[Z]$ for all directed Z. The collection $[X \to Y] = \{ f: X \to Y \mid f \text{ continuous} \}$ is again a cpo.

A nice property of cpo's is that they allow the definition of a fixpoint

operator $\mathrm{fix} = \mathrm{fix}_X \in [[X \to X] \to X]$ (X some cpo) by $\mathrm{fix}(f) := \bigvee \{f^n \bot \mid n \in \omega\}$, satisfying $f(\mathrm{fix}(f)) = \mathrm{fix}(f)$ and $\forall x \in X(fx = x \to \mathrm{fix}(f) \leqslant x)$.

An element x of a cpo X is called *compact* (also called finite or isolated elsewhere) if $\forall Y \in D(X)(x \leqslant \bigvee Y \to \exists y \in Y(x \leqslant y))$. We put $c(X) = \{x \in X \mid x \text{ compact}\}$, $\Downarrow_c(x) = c(X) \cap \{y \mid y \leqslant x\}$. X is called *algebraic* if every element x is the supremum of $\Downarrow_c(x) \in D(X)$. A function f between cpo's is called *compactness-preserving* if fc is compact whenever c is.

A subset Y of a cpo X with $\exists x \in X \forall y \in Y(y \leqslant x)$ is called *bounded* (also called consistent); if every bounded set has a supremum, then X is *boundedly complete* (also called consistently complete).

A *domain* is an algebraic and boundedly complete cpo. If X, Y are domains, then so is $[X \to Y]$. A proof for this well-known fact runs as follows. Let $\mathbf{a} = a_1, ..., a_n$ and $\mathbf{b} = b_1, ..., b_n$ be finite sequences of elements of $c(X)$, resp. $c(Y)$. We define $\mathrm{Adm}(\mathbf{a}, \mathbf{b}) := \forall i, j \leqslant n((a_i \leqslant a_j \to b_i \leqslant b_j) \wedge \forall x \in X(a_i, a_j \leqslant x \to \exists k \leqslant n(a_i, a_j \leqslant a_k \leqslant x)))$ and $f_{\mathbf{ab}}(x) := \bigvee \{b_i \mid a_i \leqslant x\}$. Functions of the form $f_{\mathbf{ab}}$ are called *finite*, and *admissible* if also $\mathrm{Adm}(\mathbf{a}, \mathbf{b})$ holds. (We use these notions later on, in the proof of (c) in Lemma 4.4(ii).) Now the compact elements of $[X \to Y]$ are exactly the admissible finite ones, and this can be used to show that $[X \to Y]$ is algebraic. Preservation of boundedly completeness under $\to$ is straightforward.

2.2. *Complete Lattices, Embeddings*

A cpo X is called a *complete lattice* if each subset Z has a supremum. Then X has a top element $\top$ defined by $\bigvee X$, and for subsets Z the infimum $\bigwedge Z$ is defined by $\bigvee \{x \in X \mid \forall z \in Z(x \leqslant z)\}$. If $f: X \to Y$ satisfies $f(\bigvee Z) = \bigvee (fZ)$ for all $Z \subseteq X$, then f is called *additive*.

Let X, Y be cpo's, $\phi \in [X \to Y]$. ϕ is called an *embedding* if (i) $\forall xy \in X(x \leqslant y \leftrightarrow \phi x \leqslant \phi y)$ and (ii) $\forall y \in Y(\{x \mid \phi x \leqslant y\} \in D(X))$. Embeddings ϕ have a continuous left inverse ϕ^{L} defined by $\phi^{\mathrm{L}}(y) = \bigvee \{x \mid \phi x \leqslant y\}$ satisfying besides (iii) $\phi^{\mathrm{L}} \cdot \phi = \mathrm{id}_X$ also (iv) $\phi \cdot \phi^{\mathrm{L}} \leqslant \mathrm{id}_Y$. Conversely, if ϕ has a left inverse ϕ^{L} satisfying (iii) and (iv) then ϕ is an embedding. Pairs $\langle \phi, \phi^{\mathrm{L}} \rangle$ for which (iii) and (iv) hold are called *projection pairs.*

If X is a complete lattice, then condition (ii) is superfluous (follows from (i)) and embeddings are additive (for $\bigvee \{\phi z \mid z \in Z\} \leqslant \phi(\bigvee Z) = \phi(\bigvee \{\phi^{\mathrm{L}}(\phi z) \mid z \in Z\}) \leqslant (\phi \cdot \phi^{\mathrm{L}})(\bigvee \{\phi z \mid z \in Z\}) \leqslant \bigvee \{\phi z \mid z \in Z\}$).

Some examples of complete lattices:

$\underline{\bot} := \langle \{\bot\}, \leqslant \rangle$

$\underline{2} := \langle \{\bot, \top\}, \leqslant \rangle$, with $\bot < \top$

$\underline{3} := \langle \{\bot, *, \top\}, \leqslant \rangle$, with $\bot < * < \top$

$\underline{B} := \langle \{\bot, \text{true}, \text{false}, \top\}, \leqslant \rangle$, with $x \leqslant y$ iff ($x = \bot$ or $x = y$ or $y = \top$)

$\underline{N} := \langle \{\bot, 0, 1, 2, 3, ..., \top\}, \leqslant \rangle$, with $x \leqslant y$ iff ($x = \bot$ or $x = y$ or $y = \top$)

Besides $\rightarrow$, we have the binary operations $\times$ (product) on cpo's and $+$ (separated sum) on complete lattices. $\times$ is as usual, and $X+Y$ is defined as the disjoint union of X and Y extended with a top and a bottom element.

2.3. *Categories, Functors*

(See, e.g., (Arbib and Manes, 1975; Mac Lane, 1971) for more information.) Categories $\mathbb{C}$, consisting of objects $\mathrm{Obj}(\mathbb{C})$ and arrows $\mathrm{Ar}(\mathbb{C})$, are defined as usual, together with functors between them. Composition of arrows f and g is denoted by $f \cdot g$. Isomorphic objects in a category shall be identified. Sometimes we identify objects C with their arrows 1_C, and this leads to $\mathrm{Obj}(\mathbb{C}) \subseteq \mathrm{Ar}(\mathbb{C}) = \mathbb{C}$. Given two categories $\mathbb{C}$ and $\mathbb{C}'$, the product category is denoted by $\mathbb{C} * \mathbb{C}'$, and for $\mathbb{C} * \cdots * \mathbb{C}$ (n times) we write $\mathbb{C}^n$. $\mathbb{F}(\mathbb{C}, \mathbb{C}')$ is the collection of all functors from $\mathbb{C}$ to $\mathbb{C}'$. A mapping $H: \mathrm{Obj}(\mathbb{C}) \rightarrow \mathrm{Ar}(\mathbb{C}')$ is a *natural transformation* between $F, G \in \mathbb{F}(\mathbb{C}, \mathbb{C}')$ if $G(f) \cdot H(\mathrm{dom}(f)) = H(\mathrm{cod}(f)) \cdot F(f)$ for all $f \in \mathrm{Ar}(\mathbb{C})$.

In the sequel, we use the categories $\mathbb{A}$ and $\mathbb{A}_e$, both having the collection of algebraic complete lattices as objects; the arrows of $\mathbb{A}$ are the continuous functions, the arrows of $\mathbb{A}_e$ are embeddings. $\mathbb{A}$ and $\mathbb{A}_e$ are full subcategories of the corresponding categories $\mathbb{D}$ and $\mathbb{D}_e$ of domains.

A functor $F \in \mathbb{F}(\mathbb{D}^n, \mathbb{D})$ are called monotonic if it preserves $\leqslant$; such a functor is also a functor from $\mathbb{D}_e^n$ to $\mathbb{D}_e$, with $F(\langle \phi_1, \psi_1 \rangle, \ldots, \langle \phi_n, \psi_n \rangle) = \langle F(\phi_1, \ldots, \phi_n), F(\psi_1, \ldots, \psi_n) \rangle$. The same applies for $\mathbb{A}$ instead of $\mathbb{D}$.

The operations $\times$, $+$ on $\mathrm{Obj}(\mathbb{A})$ are extended straightforwardly to $\mathbb{A}$ by the usual pointwise definitions, thus becoming monotonic functors in $\mathbb{F}(\mathbb{A}^2, \mathbb{A})$. Since monotonic functors $F \in \mathbb{F}(\mathbb{A}^n, \mathbb{A})$ preserve embeddings (with $F(\phi_1, \ldots, \phi_n)^{\mathrm{L}} = F(\phi_1^{\mathrm{L}}, \ldots, \phi_n^{\mathrm{L}})$), we have $\times, + \in \mathbb{F}(\mathbb{A}_e^2, \mathbb{A}_e)$.

$\rightarrow$, already defined on $\mathrm{Obj}(\mathbb{A})$, is extended to $\mathrm{Ar}(\mathbb{A})$ by $(f \rightarrow g)\, x = g \cdot x \cdot f$. Thus $\rightarrow$ is a monotonic functor in $\mathbb{F}(\mathbb{A}^{\mathrm{op}} * \mathbb{A}, \mathbb{A})$. The contravariance in the first argument prevents straightforward extension to $\mathbb{A}_e$, therefore we define $\xrightarrow[e]{}$ on $\mathbb{A}_e^2$ by $(\phi \xrightarrow[e]{} \psi) = (\phi^{\mathrm{L}} \rightarrow \psi)$. Using $(\phi \xrightarrow[e]{} \psi)^{\mathrm{L}} = (\phi \rightarrow \psi^{\mathrm{L}})$, it follows that $(\phi \xrightarrow[e]{} \psi)$ is again an embedding and $\xrightarrow[e]{} \in \mathbb{F}(\mathbb{A}_e^2, \mathbb{A}_e)$. Observe that $\rightarrow$ and $\xrightarrow[e]{}$ behave the same on $\mathrm{Obj}(\mathbb{A}) = \mathrm{Obj}(\mathbb{A}_e)$.

2.4. *Chains, Limits, Fixpoints*

We now sketch the theory of fixpoints of functors on $\mathbb{D}$, based on limits of chains of domains. (See Smyth and Plotkin, 1982, for more information.) A *chain* is a sequence $\langle X_n, \phi_n \rangle_n$ of cpo's with embeddings $\phi_n \in [X_n \rightarrow X_{n+1}]$. The *limit* of the chain is defined by $\lim \langle X_n, \phi_n \rangle_n = \{ \langle x_n \rangle_n \mid \forall n (x_n \in X_n \wedge \phi_n^{\mathrm{L}}(x_{n+1}) = x_n) \}$, often abbreviated to $\lim_n X_n$; the ordering on $\lim_n X_n$ is defined pointwise. Then $\lim_n X_n$ is again a cpo; moreover, we have $c(\lim_n X_n) = \{ \langle x_n \rangle_n \in \lim_n x_n \mid \exists n (x_n \in$

$c(X_n) \wedge \forall m \geqslant n(x_{m+1} = \phi_m(x_m)))\}$. Being algebraic, bounded completeness, and having a top element are all preserved under taking limits; as a consequence, $\mathbb{A}_e$ is closed under lim.

Functors F are *continuous* if they commute with taking limits, i.e. (for unary F) if $F(\lim\langle X_n, \phi_n\rangle_n) = \lim_n\langle FX_n, F\phi_n\rangle_n$. $\mathbb{CF}(\mathbb{C}, \mathbb{C}')$ is the collection of all continuous functors from $\mathbb{C}$ to $\mathbb{C}'$. $\times$, $+$, and $\xrightarrow[e]{}$ are examples of continuous functors. ch(F), the *chain of functor* F is defined as $\langle F^n \perp, F^n\phi\rangle_n$, where ϕ is the unique embedding from $\perp$ to $F\perp$ (mapping $\perp$ to $\perp$). Fix(F), the *fixpoint of* F, is defined as lim(ch(F)) and we have $F(\mathrm{Fix}(F)) = \mathrm{Fix}(F)$ for continuous F.

Let $\langle X_n, \phi_n\rangle$, $\langle Y_n, \psi_n\rangle$ be chains with $\lim_n X_n = X$, $\lim_n Y_n = Y$. By the continuity of $\xrightarrow[e]{}$, $[X \to Y] = \lim_n[X_n \to Y_n]$, so elements f of $[X \to Y]$ are identified with sequences $\langle f_n\rangle_n \in \lim_n[X_n \to Y_n]$, i.e., satisfying (i) $f_n = (\phi_n \xrightarrow[e]{} \psi_n)^L f_{n+1} = \psi_n^I \cdot f_{n+1} \cdot \phi_n$ for all n; such sequences $\langle f_n\rangle_n$ are called *arrow chains*, and the corresponding $f \in [X \to Y]$ is called the limit $\lim_n f_n$ of $\langle f_n\rangle_n$. It is clear that $\mathbb{A}$ is closed under limits of arrow chains. Limits of arrow chains satisfy $(\lim_n f_n)(\langle x_n\rangle_n) = \langle \bigvee\{(\psi L_n^L \cdot \cdots \cdot \psi_{n+m-1}^L)(f_{n+m}(x_{n+m}))\,|\, m \in \omega\}\rangle_n$; also $\lim_n f_n \leqslant \lim_n g_n$ iff $f_n \leqslant g_n$ for all n, and $(\lim_n f_n)\cdot(\lim_n g_n) = \lim_n(f_n \cdot g_n)$.

If, moreover, (ii) $\psi_n \cdot f_n = f_{n+1} \cdot \phi_n$ for all n, then $\langle f_n\rangle_n$ is called a *strong arrow chain* (observe that (ii) implies (i); the notion of strong arrow chain is used in the construction of limits of abstractions, see 4.14(ii)).

For $F \in \mathbb{F}(\mathbb{C}_1 * \mathbb{C}_2, \mathbb{C}_3)$ and $f \in \mathbb{C}_1$ we define $F_f := \lambda g.F(f, g)$. If $f \in \mathrm{Obj}(\mathbb{C}_1)$, then $F_f \in \mathbb{F}(\mathbb{C}_2, \mathbb{C}_3)$; moreover, if F is continuous, then so is F_f.

Let F be an $(n+1)$-ary functor on $\mathbb{A}_e$ and $\phi = \langle \phi_1, ..., \phi_n\rangle \in \mathrm{Ar}(\mathbb{A}_e^n)$. Then $F_{\mathrm{dom}(\phi)}$ and $F_{\mathrm{cod}(\phi)}$ are unary functors on $\mathbb{A}_e$ and their chains have limits in Obj($\mathbb{A}_e$), say X and Y. Now $\mathrm{Fix}(F_\phi) \in [X \to Y]$ is an arrow in $\mathbb{A}_e$, called the *fixpoint of* F *in* ϕ. By abstracting from ϕ we get the fixpoint functor $\mathrm{Fix}(F) := \lambda\phi.\mathrm{Fix}(F_\phi) \in \mathbb{F}(\mathbb{A}_e^n, \mathbb{A}_e)$. If F is continuous, then so is Fix(F) (proved via permutation of limits of chains).

Some nonstandard notation used in the sequel:

$$^n\mathbb{C} \underset{\text{def}}{=} \mathbb{CF}(\mathbb{C}^n, \mathbb{C})$$

$$^*\mathbb{C} \underset{\text{def}}{=} \bigcup\{^n\mathbb{C}\,|\, n = 0, 1, ...,\}$$

$$_n\mathbb{C} \underset{\text{def}}{=} \mathbb{C}^n \to \bigcup \mathrm{Obj}(\mathbb{C})$$

$$_*\mathbb{C} \underset{\text{def}}{=} \bigcup\{_n\mathbb{C}\,|\, n = 0, 1, ...,\}.$$

Observe that $^0\mathbb{C}$ and $\mathbb{C}$ are isomorphic; they will be identified. The same holds for $_0\mathbb{C}$ and $\bigcup \mathrm{Obj}(\mathbb{C})$. The relation ε on $_*\mathbb{C} \times {}^*\mathbb{C}$ is defined as $\varepsilon = \bigcup\{\varepsilon_n\,|\, n \in \omega\}$, where

$$f\,\varepsilon_n F \quad \text{iff} \quad \forall X_1 \cdots X_n \in \mathrm{Obj}(\mathbb{C})(f(X_1 \cdots X_n) \in F(X_1 \cdots X_n)).$$

3. The Language and its Interpretation

3.1. *The Language* L

L is a typed lambda calculus, defined in the style of Barendregt (1992). It has expressions of three sorts: object, type, and kind. There are two sorts of statements:

$$\langle \text{type expression} \rangle : \langle \text{kind expression} \rangle,$$

$$\langle \text{object expression} \rangle : \langle \text{type expression} \rangle.$$

Object expressions are called (polymorphic) terms. Kind expressions have the form ${}^{n}\Theta$ ($n = 0, 1, \ldots$). ${}^{0}\Theta$, also written Θ, is the kind of type expressions of order zero; these type expressions are usually called types. ${}^{n}\Theta$ can be thought of as the collection $\Theta^{n} \to \Theta$ of n-ary type constructors.

The collection $\mathbf{K} = \mathbf{K}_{\text{type}} \cup \mathbf{K}_{\text{object}}$ acts as a parameter of **L**. $\mathbf{K}_{\text{type}}$ contains statements of the form $K : {}^{n}\Theta$, where K is a type constant; $\mathbf{K}_{\text{object}}$ contains statements $k : \Phi$, where k is a term constant and Φ a closed and wellformed type expression.

Candidates for inclusion in **K** are

(i) $N : \Theta$ (the natural numbers), $B : \Theta$ (the Booleans), together with their usual unary and binary operations such as $+ : N \to N \to N$, not : $B \to B$, etc.;

(ii) $\times : {}^{2}\Theta$ and $+ : {}^{2}\Theta$, the type operators of product and sum, and the associated polymorphic operations such as pair : $\delta\alpha\beta.\alpha \to \beta \to (\alpha \times \beta)$, case : $\delta\alpha\beta\gamma \cdot (\alpha \to \gamma) \to (\beta \to \gamma) \to (\alpha \times \beta) \to \gamma$, etc.;

(iii) the conditional cond : $\delta\alpha.B \to \alpha \to \alpha \to \alpha$.

Besides the constants in **K** we have

${}^{n}\Theta$ ($n = 0, 1, \ldots$; the kinds of the language, see above)

$\to$ (binary type constructor, with the usual meaning)

δ (for type abstraction in the definition of type constructors)

μ (for the definition of the fixpoint of type constructors)

λ (for object abstraction defined on terms, as usual)

Λ (for type abstraction in the definition of polymorphic terms)

fix (the polymorphic fixpoint operator)

The expressions of the language are defined by giving the syntactical categories, followed by their metavariables and generating rules (if any):

KIND	(kinds)	κ	$\kappa ::= {}^0\Theta \mid {}^1\Theta \mid {}^2\Theta \mid \cdots$
TVAR	(type variables)	α, β, γ	
TYPE	(types)	ρ, σ, τ	$\tau ::= \alpha \mid \mu\Phi$
CONSTR	(constructors)	Φ	$\Phi ::= K \mid \to \mid \tau \mid \Phi\tau \mid \delta\alpha.\Phi$
OVAR	(object variables)	x, y, z	
TERM	(object terms)	r, s, t	$t ::= x \mid tt \mid \lambda x{:}\tau.t$
PTERM	(polymorphic terms)	p, q	$p ::= k \mid \mathrm{fix} \mid t \mid p\tau \mid \Lambda\alpha.p$

It is clear that we have TVAR $\subseteq$ TYPE $\subseteq$ CONSTR and OVAR $\subseteq$ TERM $\subseteq$ PTERM. We postulate that KIND, CONSTR, and PTERM are disjoint.

The notational conventions are as usual; e.g., rst means $(rs)t$, $s(t_1, ..., t_n)$ means $st_1 \cdots t_n$, $\lambda x_1{:}\tau_1 \cdots x_n{:}\tau_n.t$ means $\lambda x_1{:}\tau_1.(\cdots(\lambda x_n{:}\tau_n.t)\cdots)$; $\to\sigma\tau$ is written $\sigma \to \tau$, etc.

L has a deduction system for wellformedness. This system has the form of a sequent calculus with sequents of the form $\Gamma \vdash S$, where S is some statement and Γ is a finite set of statements of the form $x{:}\tau$. The intuitive meaning of $\Gamma \vdash E{:}E'$ is as follows: assuming the type assignment given by Γ, E is a wellformed expression and belongs to E'.

For $\varnothing \vdash S$ we write $\vdash S$. $\Gamma \vdash E_1{:}E_2{:}E_3$ abbreviates $\Gamma \vdash E_1{:}E_2$ and $\Gamma \vdash E_2{:}E_3$.

Axioms and rules for types:

$$\vdash \alpha : {}^0\Theta \qquad \vdash \to : {}^2\Theta \qquad \vdash S \text{ for } S \in \mathbf{K}_{\text{type}}$$

$$\frac{\vdash \Phi : {}^{n+1}\Theta \quad \vdash \tau : {}^0\Theta}{\vdash \Phi\tau : {}^n\Theta} \qquad \frac{\vdash \Phi : {}^n\Theta}{\vdash \delta\alpha.\Phi : {}^{n+1}\Theta} \qquad \frac{\vdash \Phi : {}^1\Theta}{\vdash \mu\Phi : {}^0\Theta}$$

Axioms and rules for terms:

$$\Gamma, x : \tau \vdash x : \tau \qquad \vdash \mathrm{fix} : \delta\alpha.(\alpha \to \alpha) \to \alpha \qquad \vdash S \text{ for } S \in \mathbf{K}_{\text{object}}$$

$$\frac{\Gamma \vdash s : \sigma \to \tau : {}^0\Theta \quad \Gamma \vdash t : \sigma : {}^0\Theta}{\Gamma \vdash st : \tau} \qquad \frac{\Gamma, x : \sigma \vdash t : \tau : {}^0\Theta \quad \vdash \sigma : {}^0\Theta}{\Gamma \vdash \lambda x : \sigma.t : \sigma \to \tau}$$

$$\frac{\Gamma \vdash p : \Phi : {}^{n+1}\Theta \quad \vdash \tau : {}^0\Theta}{\Gamma \vdash p\tau : \Phi\tau} \qquad \frac{\Gamma \vdash p : \Phi}{\Gamma \vdash \Lambda\alpha.p : \delta\alpha.\Phi} \ \alpha \text{ not free in } \Gamma$$

Finally there is a weakening rule:

$$\frac{\Gamma \vdash S}{\Gamma \cup \Delta \vdash S}$$

The fixpoint operator μ can be used for the explicit definition of recursive types. Two well-known examples: the list constructor L, recursively defined by $L(\tau) = 1 + (\tau \times L(\tau))$, has the explicit definition $L := \delta\alpha.\mu(\delta\beta.1 + (\alpha \times \beta))$; the equation $\tau = \sigma + (\tau \to \tau)$ of a model for the type-free lambda calculus over σ, which is solved by $\tau = \mu(\delta\alpha.\sigma + (\alpha \to \alpha))$.

3.2. *Interpretations*

An interpretation I of **L** is a mapping defined on the constants in **K**. Associated to an interpretation I is a Cartesian closed category $\mathbb{C}_I$ which is a subcategory of $\mathbb{D}$. I satisfies:

$$\text{if} \quad (K : {}^n\Theta) \in \mathbf{K}_{\text{type}} \text{ then } I(K) \in {}^n\mathbb{C}_I;$$

$$\text{if} \quad (k : \Phi) \in \mathbf{K}_{\text{object}} \text{ then } I(k)\, \varepsilon I[\![\Phi]\!].$$

(${}^n\mathbb{C}_I$ and ε are defined at the end of 2.4.) We extend I to a mapping defined on all wellformed expressions of **L**. This is done as usual with assignments (also called environments), which are mappings defined on variables of some syntactical category. If a is such an assignment, x a variable, and d an object, then $a' := a[x \to d]$ is the assigment defined by $a'(x) = d$, $a'(y) = a(y)$ if y is a variable different from x. Given I and $\mathbb{C}_I$, there are two collections of assignments:

$$\text{TENV}_I \underset{\text{def}}{=} \text{TVAR} \to \mathbb{C}_I$$

$$\text{OENV}_I \underset{\text{def}}{=} \text{OVAR} \to {}_0\mathbb{C}_I$$

Now we extend I to

$$I : \text{KIND} \to \{{}^n\mathbb{C}_I \mid n = 0, 1, \ldots\}$$

$$I : \text{CONSTR} \to \text{TENV}_I \to {}^*\mathbb{C}_I$$

$$I : \text{PTERM} \to \text{TENV}_I \to \text{OENV}_I \to {}_*\mathbb{C}_I$$

We adopt the usual convention of writing the first argument of I between open square brackets $[\![\ ]\!]$. The other arguments $a \in \text{TENV}_I$, $b \in \text{OENV}_I$ are written without parentheses. They will be dropped sometimes later on, e.g., if the value of $I[\![E]\!]$ does not depend on them (this is the case when E is closed).

$$I[\![{}^n\Theta]\!] = {}^n\mathbb{C}_I$$

$$I[\![\alpha]\!]a = a(\alpha)$$

$$I[\![K]\!]a = I(K)$$

$$I[\![\to]\!]a = \underset{e}{\longrightarrow}$$

$$I[\![\Phi\tau]\!]a = I[\![\Phi]\!]a(I[\![\tau]\!]a)$$
$$I[\![\delta\alpha.\tau]\!]a = \lambda f \in \mathbb{C}_I.I[\![\tau]\!](a[\alpha \to f])$$
$$I[\![\mu\Phi]\!]a = \mathrm{Fix}(I[\![\Phi]\!]a)$$
$$I[\![x]\!]ab = b(x)$$
$$I[\![k]\!]ab = I(k)$$
$$I[\![\mathrm{fix}]\!]ab = \lambda f \in \mathbb{C}_I.\mathrm{fix}_{\mathrm{cod}(f)}$$
$$I[\![st]\!]ab = (I[\![s]\!]ab)(I[\![t]\!]ab)$$
$$I[\![\lambda x : \sigma.t]\!]ab = \lambda e \in I[\![\sigma]\!]a.I[\![t]\!]a(b[x \to e])$$
$$I[\![p\tau]\!]ab = I[\![p]\!]ab(I[\![\tau]\!]a)$$
$$I[\![\Lambda\alpha.t]\!]ab = \lambda f \in \mathbb{C}_I.I[\![t]\!](a[\alpha \to f])b$$

We also put

$(I, a, b) \models \Phi : K \underset{\text{def}}{=} I[\![\Phi]\!]a \in I[\![K]\!]$

$(I, a, b) \models p : \Phi \underset{\text{def}}{=} I[\![p]\!]ab\ \varepsilon I[\![\Phi]\!]a$

$(I, a, b) \models \Gamma \underset{\text{def}}{=} (I, a, b) \models S$ for all $S \in \Gamma$

$\Gamma \models_I S \underset{\text{def}}{=}$ for all $a \in \mathrm{TENV}_I$, $b \in \mathrm{OENV}_I$: if $(I, a, b) \models \Gamma$ then $(I, a, b) \models S$

$\Gamma \models S \underset{\text{def}}{=} \Gamma \models_I S$ for all interpretations I

3.2.1. LEMMA (Soundness). *If* $\Gamma \vdash S$ *then* $\Gamma \models S$.

Proof. With induction over the length of a derivation of $\Gamma \vdash S$, first for type statements, then for object statements. Some induction loading is required: for type statements $\Phi : {}^n\Theta$ with free type variables $\alpha_1, ..., \alpha_k$, the conclusion of the lemma has to be strengthened to

$$\Gamma \models \delta\alpha_1 \cdots \alpha_k.\Phi : {}^{k+n}\Theta;$$

for object statements $p : \Phi : {}^n\Theta$ with free type variables $\underline{\alpha} = \alpha_1, ..., \alpha_k$, free term variables $\underline{x} = x_1, ..., x_m$, and $\{\underline{x} : \underline{\tau}\} = \{x_1 : \tau_1, ..., x_m : \tau_m\} \subseteq \Gamma$, the required strengthening reads

$$\Gamma - \{\underline{x} : \underline{\tau}\} \models \Lambda\underline{\alpha}\underline{\beta}.\lambda\underline{x} : \underline{\tau}.p\underline{\beta} : \delta\underline{\alpha}\underline{\beta}.\Phi\underline{\beta} : {}^{k+n}\Theta,$$

where $\underline{\beta} = \beta_1, ..., \beta_n$ are fresh term variables. ∎

3.3. *The Concrete Interpretation C*

C is the straightforward interpretation of **L** into the category $\mathbb{A}_e$ of algebraic complete lattices with embeddings. Its effect on the constants is as expected: $C(N) = \underline{N}$, $C(B) = \underline{B}$, $C(B) = \underline{B}$, $C(\to) = \underset{e}{\longrightarrow}$. $C(\times) = \times$, $C(+) = +$, and likewise for other contants.

4. Abstraction

In order to perform strictness analysis, the idea is to simplify the concrete interpretation C to an abstract interpretation A. This is done with help of a type interpretation abs, presented as a mapping on type constants $K : {}^{n}\Theta$ with $\mathrm{abs}(K) : \mathbb{D}^n \to \mathbb{D}$. It will turn out that $\mathrm{abs}(\to)$ is *not* a functor, so abs is not exactly an interpretation in the sense of 3.2. Instead of the conditions given there, abs must satisfy (taking for simplicity $n = 1$ in $K : {}^{n}\Theta$)

$$\mathrm{dom}(\mathrm{abs}(K)f) = C(K)(\mathrm{dom}(f)), \tag{1}$$

$$\mathrm{cod}(f) = \mathrm{cod}(g) \Rightarrow \mathrm{cod}(\mathrm{abs}(K)f) = \mathrm{cod}(\mathrm{abs}(K)g). \tag{2}$$

Now the abstract interpretation A is defined on types by

$$A(K) \underset{\mathrm{def}}{=} \lambda f.\mathrm{cod}(\mathrm{abs}(K)1_{\mathrm{cod}(f)})$$

Then

$$\mathrm{cod}(\mathrm{abs}(K)f) = A(K)(\mathrm{cod}(f)).$$

On (polymorphic) object constants $k : \Phi : {}^{n}\Theta$, A must be defined in such a way that

$$A(k)(A[\![\tau_1]\!], \ldots, A[\![\tau_n]\!]) \geqslant \mathrm{abs}[\![\Phi\tau_1 \cdots \tau_n]\!](C[\![k\tau_1 \cdots \tau_n]\!]) \quad \text{for all wellformed types } \tau_1, \ldots, \tau_n \tag{3}$$

This reduces to $A(k) \geqslant \mathrm{abs}[\![\tau]\!](C(k))$ for $k : \tau : {}^{0}\Theta$, so here we can take $A(k) \underset{\mathrm{def}}{=} \mathrm{abs}[\![\tau]\!](C(k))$. Now A is an interpretation of **L** satisfying

$$\mathrm{dom}(\mathrm{abs}[\![\tau]\!]) = C[\![\tau]\!], \mathrm{cod}(\mathrm{abs}[\![\tau]\!]) = A[\![\tau]\!] \text{ for } \tau : {}^{0}\Theta \tag{4}$$

$$\mathrm{abs}[\![\tau]\!](C[\![t]\!]) \leqslant A[\![t]\!] \text{ for } t : \tau : {}^{0}\Theta. \tag{5}$$

The obvious example for abs and A is as follows: take $A(K) = \underline{2}$, $\mathrm{abs}(K) = \lambda x.(\text{if } x = \bot \text{ then } \bot \text{ else } \top)$ for $K : {}^{0}\Theta$, and $A(\times) = \mathrm{abs}(\times) = \times$, $A(+) = \mathrm{abs}(+) = +$, $A(\to) = \underset{e}{\to}$; as we see later on, $\mathrm{abs}(\to)$ cannot be equal to $\underset{e}{\to}$.

Under reasonable assumptions, (5) cannot be strengthened to equality. To see this, assume that abs_N (abs restricted to $\underline{N}$) satisfies $\mathrm{abs}_N(m) = \mathrm{abs}_N(n) \neq \mathrm{abs}_N(\bot)$ for some m, n with $m \neq n$ (in general, abs_N will identify *all* natural numbers). Define $t := \lambda x.\mathrm{cond}(\mathrm{eq}(m, x), \bot, n)$; then we have $\mathrm{abs}_N(C[\![tm]\!]) = \mathrm{abs}_N(\bot) \neq \mathrm{abs}_N(n) = \mathrm{abs}_N(C[\![tn]\!])$, but $A[\![tm]\!] = A[\![t]\!](\mathrm{abs}_N(m)) = A[\![t]\!](\mathrm{abs}_N(n)) = A[\![tn]\!]$. So $\mathrm{abs} \cdot C = A$ is impossible, unless abs_N is injective on $\{0, 1, 2, \ldots\}$ (hence hardly an abstraction) or

$\mathrm{abs}_N(n) = \mathrm{abs}_N(\bot)$ for some n (so abs_N is not $\bot$-unique). As a consequence, the strictness analysis based on A will not be exact, i.e., we do not have ($A[\![t]\!]$ strict $\Leftrightarrow C[\![t]\!]$ strict) for all terms t.

In order to preserve the more interesting implication of this last equivalence, i.e.,

$$A[\![t]\!] \text{ strict} \Rightarrow C[\![t]\!] \text{ strict}, \tag{6}$$

property (5) and

$$\mathrm{abs}[\![\tau]\!](xy) \leqslant \mathrm{abs}[\![\sigma \to \tau]\!]x\, \mathrm{abs}[\![\sigma]\!]y \tag{7}$$

are required ((7) is also used in the proof of (5), see Theorem 4.17). (6) makes strictness analysis a safe approximation: if it tells us that t is strict then t is strict indeed; on the other hand, there may (and will) be cases in which t is strict but the analysis does not tell us. It follows from (7) that the analysis will be a better approximation as $\mathrm{abs}[\![\sigma \to \tau]\!]$, defined in terms of $\mathrm{abs}[\![\sigma]\!]$ and $\mathrm{abs}[\![\tau]\!]$, is smaller. We claim that this is obtained by defining

$$\mathrm{abs}[\![\sigma \to \tau]\!] = \lambda zy.\ \bigvee \{\mathrm{abs}[\![\tau]\!](zx) \mid \mathrm{abs}[\![\sigma]\!]x \leqslant y\}; \tag{8}$$

to verify the claim, we observe:

(i) $\mathrm{abs}[\![\sigma \to \tau]\!]x\, \mathrm{abs}[\![\sigma]\!]y = \bigvee \{\mathrm{abs}[\![\tau]\!](xu) \mid \mathrm{abs}[\![\sigma]\!]x \leqslant \mathrm{abs}[\![\sigma]\!]y\} \geqslant \mathrm{abs}[\![\tau]\!](xy)$;

(ii) if f satisfies $\mathrm{abs}[\![\tau]\!](xy) \leqslant fx(\mathrm{abs}[\![\sigma]\!]y)$, then we have $z \geqslant \mathrm{abs}[\![\sigma]\!]y \Rightarrow fxz \geqslant \mathrm{abs}[\![\tau]\!](xy)$, so fxz is an upper bound of $\{\mathrm{abs}[\![\tau]\!](xu) \mid \mathrm{abs}[\![\sigma]\!]x \leqslant \mathrm{abs}[\![\sigma]\!]y\}$, hence $\mathrm{abs}[\![\sigma \to \tau]\!] \leqslant f$. We conclude that $\mathrm{abs}[\![\sigma \to \tau]\!]$ as defined by (8) is the smallest satisfying (7).

(6) is verified as follows:

$$\begin{aligned}
A[\![t]\!]\bot = \bot &\Rightarrow \mathrm{abs}[\![\sigma \to \tau]\!](C[\![t]\!])\bot = \bot && \text{(by (5))}\\
&\Rightarrow \mathrm{abs}[\![\sigma \to \tau]\!](C[\![t]\!])(\mathrm{abs}[\![\sigma]\!](\bot)) = \bot && \text{(abs is strict)}\\
&\Rightarrow \mathrm{abs}[\![\sigma \to \tau]\!](C[\![t]\!]\bot) = \bot && \text{(by (7))}\\
&\Rightarrow C[\![t]\!]\bot = \bot && \text{(abs preserves } \bot).
\end{aligned}$$

We now work out the definition of abs in more detail, starting with the definition of the category of abstractions in which abs resides. We return to the definition of A and abs after 4.15.

4.1. DEFINITION. (i) Let X, Y be algebraic complete lattices, $f: X \to Y$. f is called an *abstraction* if it is continuous, $\bot$-unique, preserves compactness, and has a continuous right inverse f^* (i.e., $f \cdot f^* = 1_Y$). $\langle f, f^* \rangle$ is called an *abstraction pair*.

(ii) $\mathbb{A}_a$ is the category of abstraction pairs, with composition defined by

$$\langle f, f^* \rangle \cdot \langle g, g^* \rangle = \langle f \cdot g, g^* \cdot f^* \rangle.$$

4.2. *Remarks.* (i) The reasons for the definition of abstractions are the following: strictness is used in the proof of 4.7 ($((f \gg f) \gg f)\,\mathrm{fix} \leqslant \mathrm{fix}$), 4.14(i), and (6); $\perp$-preservation is needed in the argument for (6); preservation of compactness is required in the proof that $(f \gg g)\,z$ is continuous whenever z is (4.4(ii).(c)).

(ii) $\mathbb{A}_a$ is indeed a category, i.e., identity functions are abstractions and the composition of two abstraction pairs is again an abstraction pair.

(iii) Examples of abstractions are abs_N and abs_B, the unique $\perp$-unique functions in $\underline{N} \to \underline{2}$ and $\underline{B} \to \underline{2}$, respectively.

4.3. DEFINITION. (i) $\times$ and $+$ are defined componentwise on abstraction pairs.

(ii) The binary operation $\gg$ (the intended interpretation of $\to$ under abs, on abstractions $f : X \to Y$, $g : X' \to Y'$ is defined by

$$(f \gg g)(z)\,y = \bigvee \{ g(zx) \mid fx \leqslant y \} \qquad (z \in [X \to X'],\ y \in Y).$$

So $f \gg g : [X \to X'] \to Y \to Y'$. $\gg$ is extended to abstraction pairs by

$$\langle f, f^* \rangle \gg \langle g, g^* \rangle = \langle f \gg g, f \to g^* \rangle.$$

4.4. LEMMA. (i) $\times,\ + \in \mathbb{F}(\mathbb{A}_a^2, \mathbb{A}_a)$;

(ii) $\gg : \mathbb{A}_a^2 \to \mathbb{A}_a$ ($\gg$ *is not a functor*: *see the remark after the proof*).

Proof. (i) Easy, using $c(X \times Y) = c(X) \times c(Y)$ and $c(X + Y) = c(X) + c(Y)$ to see that $f \times g$ and $f + g$ preserve compactness whenever f and g do.

(ii) This involves several steps. Let $\langle f, f^* \rangle$, $\langle g, g^* \rangle$ be abstraction pairs with $f : X \to Y$, $g : X' \to Y'$; furthermore let $z \in [X \to X']$, $y \in Y$. It suffices to show that

(a) $(f \gg g)(z)$ is continuous,

(b) $f \gg g$ is continuous,

(c) $f \gg g$ preserves compactness,

(d) $f \gg g$ is $\perp$-unique,

(e) $f \to g^*$ is a continuous right inverse of $f \gg g$.

(a) Let $Z \subseteq Y$ be directed. Then

$$(f \gg g)(z)\left(\bigvee Z\right) = \bigvee \left\{ g(zx) \mid fx \leqslant \bigvee Z \right\}$$

$$= \bigvee \left\{ g\left(z\left(\bigvee \Downarrow_c(x)\right)\right) \middle| fx \leqslant \bigvee Z \right\}$$

(X is algebraic)

$$= \bigvee \left\{ \bigvee \{ g(zx') \mid x' \in \Downarrow_c(x) \} \mid fx \leqslant \bigvee Z \right\}$$

($\Downarrow_c(x)$ directed; g, z continuous)

$$= \bigvee \left\{ g(zx') \mid x' \leqslant x,\ x' \text{ compact}, fx \leqslant \bigvee Z \right\}$$

$$= \bigvee \left\{ g(zx') \mid x' \text{ compact}, fx' \leqslant \bigvee Z \right\}$$

$$= \bigvee \{ g(zx') \mid x' \text{ compact}, \exists y \in Z\, fx' \leqslant y \}$$

(f preserves compactness)

$$= \bigvee \left\{ \bigvee \{ g(zx') \mid x' \text{ compact}, fx' \leqslant y \} \mid y \in Z \right\}$$

$$= \bigvee \left\{ \bigvee \{ g(zx') \mid x' \leqslant x,\ x' \text{ compact}, fx \leqslant y \} \mid y \in Z \right\}$$

$$= \bigvee \left\{ \bigvee \left\{ \bigvee \{ g(zx') \mid x' \in \Downarrow_c(x) \} \mid fx \leqslant y \right\} \middle| y \in Z \right\}$$

$$= \bigvee \left\{ \bigvee \left\{ g(zx) \mid fx \leqslant y \right\} \middle| y \in Z \right\}$$

$$= \bigvee \{ (f \gg g)(z)(y) \mid y \in Z \}.$$

(b) Let $Z \subseteq [X \to X']$ be directed. Then

$$(f \gg g)\left(\bigvee Z\right) = \lambda y. \bigvee \left\{ g\left(\left(\bigvee Z\right)(x)\right) \middle| fx \leqslant y \right\}$$

$$= \lambda y. \bigvee \left\{ g\left(\bigvee \{ zx \mid z \in Z \}\right) \middle| fx \leqslant y \right\}$$

$$= \lambda y. \bigvee \left\{ \bigvee \{ g(zx) \mid z \in Z \} \mid fx \leqslant y \right\} \qquad (g \text{ is continuous})$$

$$= \lambda y. \bigvee \left\{ \bigvee \{ g(zx) \mid fx \leqslant y \} \mid z \in Z \right\}$$

$$= \lambda y. \bigvee \{ (f \gg g)(z)(y) \mid z \in Z \}$$

$$= \bigvee \{ (f \gg g)(z) \mid z \in Z \}.$$

(c) Let $z \in [X \to X']$ be compact, so $z = z_{\mathbf{ab}} = \lambda y. \bigvee \{ b_i \mid a_i \leqslant y \}$ with Adm(**a**, **b**) (see 2.1). Then

$$(f \gg g)(z) = \lambda y. \bigvee \{ g(zx) \mid fx \leqslant y \}$$

$$= \lambda y. \bigvee \left\{ g\left(\bigvee \{ b_i \mid a_i \leqslant x \} \right) \middle| fx \leqslant y \right\}$$

$$= \lambda y. \bigvee \{ gb_i \mid a_i \leqslant x, fx \leqslant y \}$$

$$= \lambda y. \bigvee \{ gb_i \mid fa_i \leqslant y \}$$

$$= z_{f\mathbf{a},\, g\mathbf{b}},$$

so $(f \gg g)(z)$ is compact (see 2.1; $f\mathbf{a}$, $g\mathbf{b}$ are compact, since f and g preserve compactness) and we see that $f \gg g$ preserves compactness.

(d) For $z \in [X \to X']$ we have $(f \gg g)(z) = \bot \Leftrightarrow \forall y \bigvee \{ g(zx) \mid fx \leqslant y \} = \bot \Leftrightarrow \forall x\, (g(zx) = \bot) \Leftrightarrow \forall x\, (zx = \bot) \Leftrightarrow z = \bot$, using for the third equivalence the fact that g is $\bot$-unique.

(e) Continuity of $(f \to g^*)$ is straightforward. Now let $z \in [Y \to Y']$, $y \in Y$, then $(f \gg g)((f \to g^*)\, z)\, y = \bigvee \{ g(((f \gg g)^* z)x) \mid fx \leqslant y \} = \bigvee \{ g((g^* \cdot z \cdot f)(x)) \mid fx \leqslant y \} = \bigvee \{ z(fx) \mid fx \leqslant y \} = z(f(f^* y)) = zy$. So $(f \gg g)((f \to g^*)\, z) = z$, i.e., $(f \gg g) \cdot (f \to g^*) = \mathrm{id}_{[Y \to Y']}$. ∎

4.5. *Remark.* $\gg$ is *not* a functor on $\mathbb{A}_a$. For if so, then $\cdot$ and $\gg$ would have no commute, i.e., $(1_Y \cdot f) \gg (f \cdot 1_X) = (1_Y \gg f) \cdot (f \gg 1_X)$, i.e., $f \gg f = f \cdot (f \gg 1_X)$, which is refuted by the following counterexample.

4.6. COUNTEREXAMPLE. *There is an abstraction $f: X \to Y$ with $f \gg f \neq f \cdot (f \gg 1_X)$.*

Proof. Define

$$X = \{ \bot, 0, 1, \top \}, \qquad Y = \underline{3}, \qquad Z = [X \to Y],$$

$$f\bot = \bot, \qquad f0 = f1 = *, \qquad f\top = \top.$$

We observe that $f \gg 1_X = \lambda zy. \bigvee \{zx \mid fx \leqslant y\}$. Now

$$(f \gg f)(1_Z)* = \bigvee \{fx \mid fx \leqslant *\} = \bigvee \{\bot, *\} = *,$$

$$(\mathrm{cod}(f) \gg f) \cdot (f \gg \mathrm{dom}(f))(1_Z)* = f((f \gg 1_X)\, 1_Z)*$$

$$= f\left(\bigvee \{x \mid fx \leqslant *\}\right) = f\top = \top. \quad \blacksquare$$

We have a look at the behaviour of fix under abstractions.

4.7. LEMMA. *If $f: X \to Y$ is an abstraction, then $((f \gg f) \gg f)\,\mathrm{fix}_X = \mathrm{fix}_Y$.*

Proof. Let $a \in [Y \to Y]$. Then

$$(((f \gg f) \gg f)\,\mathrm{fix})a = \bigvee \{f(\mathrm{fix}(z)) \mid (f \gg f)z \leqslant a\}$$

$$= \bigvee \left\{ f\left(\bigvee \{z^n \bot \mid n \in \omega\}\right) \,\middle|\, (f \gg f)z \leqslant a \right\}$$

$$= \bigvee \left\{ \bigvee \{f(z^n \bot) \mid n \in \omega\} \mid (f \gg f)z \leqslant a \right\}$$

$$= \bigvee \{f(z^n \bot) \mid n \in \omega, (f \gg f)z \leqslant a\}$$

$$= \bigvee \{f(z^n \bot) \mid n \in \omega, b \in Z\},$$

where $Z = \{z \mid (f \gg f)z \leqslant a\} = \{z \mid \forall x \in X \forall y \in Y(fx \leqslant y \to f(zx) \leqslant ay)\}$; now, if $z \in Z$, then $f(z^n \bot) \leqslant a^n \bot$ (proved with induction over n, using strictness of f), so we have $((f \gg f) \gg f)(\mathrm{fix})a \leqslant \bigvee \{a^n \bot \mid n \in \omega\} = \mathrm{fix}(a)$. But $(f \gg f)((f \to f^*)a) = ((f \gg f) \cdot (f \gg f)^*)a = a$, so $(f \to f^*)a \in Z$ and

$$((f \gg f) \gg f)(\mathrm{fix})a \geqslant \bigvee \{f((f \to f^*)a)^n \bot) \mid n \in \omega\}$$

$$= \bigvee \{f((f^* \cdot a \cdot f)^n \bot) \mid n \in \omega\}$$

$$= \bigvee \{a^n(f \bot) \mid n \in \omega\}$$

$$\geqslant \bigvee \{a^n \bot \mid n \in \omega\} = \mathrm{fix}(a). \quad \blacksquare$$

We go one level higher and introduce abstraction transformations.

4.8. DEFINITION. (i) Let $\langle f, f^* \rangle, \langle g, g^* \rangle$ be abstraction pairs with $f: X \to Y$, $g: X' \to Y'$ and let $\phi: X \to X'$, $\psi: Y \to Y'$ be embeddings. We call $(\phi, \psi, \langle f, f^* \rangle, \langle g, g^* \rangle)$ an *abstraction diagram* if

$$\psi \cdot f = g \cdot \phi,$$

$$f^* = \phi^{\mathrm{L}} \cdot g^* \cdot \psi.$$

(ii) Let $F, G \in \mathbb{F}(\mathbb{A}_e^n, \mathbb{A}_e)$. H is called an *abstraction transformation between F and G* (notation: $H: F \Rightarrow G$) if the following holds:

> if $(\phi_1, \psi_1, p_1, q_1), \ldots, (\phi_n, \psi_n, p_n, q_n)$ are abstraction diagrams, then so is $(F(\phi_1, \ldots, \phi_n), G(\psi_1, \ldots, \psi_n), H(p_1, \ldots, p_n), H(q_1, \ldots, q_n))$.

4.9. *Remark.* If $H: F \Rightarrow G$ and there is an H_1 with $(H\langle f, f^* \rangle)_1 = H_1(f)$, then H_1 is a natural transformation from F to G. To see this, observe that $(\phi, \phi, \langle 1_X, 1_X \rangle, \langle 1_Y, 1_Y \rangle)$ is an abstraction diagram, so $F(\phi) \cdot H_1(X) = H_1(Y) \cdot G(\phi)$.

4.10. LEMMA. (i) $\times : \times \Rightarrow \times$.

(ii) $+ : + \Rightarrow +$.

(iii) $\gg : \xrightarrow{e} \Rightarrow \xrightarrow{e}$.

(iv) *Abstraction transformations are closed under composition.*

Proof. (i), (ii), (iv): straightforward.

(iii): Let $(\phi, \psi, \langle f, f^* \rangle, \langle g, g^* \rangle)$ and $(\phi', \psi', \langle f', f'^* \rangle, \langle g', g'^* \rangle)$ be two abstraction diagrams, so

$$\psi \cdot f = g \cdot \phi, \tag{9}$$

$$f^* = \phi^L \cdot g^* \cdot \psi, \tag{10}$$

$$\psi' \cdot f' = g' \cdot \phi', \tag{11}$$

$$f'^* = \phi'^L \cdot g'^* \cdot \psi'. \tag{12}$$

We must show that $(\phi \xrightarrow{e} \phi', \psi \xrightarrow{e} \psi', \langle f \gg f', f \to f'^* \rangle, \langle g \gg g', g \to g'^* \rangle)$ is an abstraction diagram, i.e.,

$$(\psi \xrightarrow{e} \psi') \cdot (f \gg f') = (g \gg g') \cdot (\phi \xrightarrow{e} \phi'), \tag{13}$$

$$f \to f'^* = (\phi \xrightarrow{e} \phi')^L \cdot (g \to g'^*) \cdot (\psi \xrightarrow{e} \psi'). \tag{14}$$

For (13) we argue as follows. We have

$$((\psi \xrightarrow{e} \psi') \cdot (f \gg f'))(x)\, y = \psi' \left(\bigvee \{ f'(xz) \mid fz \leqslant \psi^L y \} \right),$$

$$((g \gg g') \cdot (\phi \xrightarrow{e} \phi'))(x)\, y = \bigvee \{ g'(\phi'(x(\phi^L z))) \mid gz \leqslant y \}$$

$$= \psi' \left(\bigvee \{ f'(x(\phi^L z)) \mid gz \leqslant y \} \right),$$

where the last equality holds because of (11) and the additivity of ψ' (see 2.2), so it suffices to show that

$$\{ z \mid fz \leqslant \psi^L y \} = \{ \phi^L z \mid gz \leqslant y \}, \tag{15}$$

which is done in two steps. First assume $z' \in \{z \mid fz \leqslant \psi^{\mathrm{L}} y\}$, i.e., $fz' \leqslant \psi^{\mathrm{L}} y$, then $\psi(fz') \leqslant \psi(\psi^{\mathrm{L}} y) \leqslant y$, so (by (9)) $g(\phi z') \leqslant y$ and $z' = \phi^{\mathrm{L}}(\phi z') \in \{\phi^{\mathrm{L}} z \mid gz \leqslant y\}$. For the other way around, assume $u \in \{\phi^{\mathrm{L}} z \mid gz \leqslant y\}$, then $u = \phi^{\mathrm{L}} z$ for some z with $gz \leqslant y$; then $\psi^{\mathrm{L}}(gz) \leqslant \psi^{\mathrm{L}} y$ and also $f \cdot \psi \leqslant \psi^{\mathrm{L}} \cdot g$ (a consequence of (9) and the properties of embeddings), so $f(\psi z) \leqslant \psi^{\mathrm{L}} y$, so $u = \phi^{\mathrm{L}} z \in \{z \mid fz \leqslant \psi^{\mathrm{L}} y\}$. So (15) is proved, and we conclude (13).

(14) is proved as follows: $(f \to f'^{*})(z) = f'^{*} \cdot z \cdot f = \phi'^{\mathrm{L}} \cdot g'^{*} \cdot \psi' \cdot z \cdot \psi^{\mathrm{L}} \cdot g \cdot \phi = ((\phi \xrightarrow{}_{\mathrm{e}} \phi')^{\mathrm{L}} \cdot (g \to g'^{*}) \cdot (\psi \xrightarrow{}_{\mathrm{e}} \psi'))(z)$, using (12) and (9) for the second equality. ∎

4.11. *Remark.* The condition $\psi \cdot f = g \cdot \phi$ in the definition of abstraction diagram is needed for the definition of fixpoints of abstraction transformations (see 4.14(ii)). The next counterexample shows that it is also required in the proof of Lemma 4.10.

4.12. COUNTEREXAMPLE. *There are diagrams* (ϕ, ψ, f, g) *and* (ϕ', ψ', f', g') *satisfying* $f = \psi^{\mathrm{L}} \cdot g \cdot \phi$, $f' = \psi'^{\mathrm{L}} \cdot g' \cdot \phi'$, *but* $(f \gg f') \neq (\psi \xrightarrow{}_{\mathrm{e}} \psi')^{\mathrm{L}} \cdot (g \gg g') \cdot (\phi \xrightarrow{}_{\mathrm{e}} \phi')$.

Proof. We have

(a) $(f \gg f')\, xy = \bigvee \{f'(xu) \mid fu \leqslant y\}$,

(b) $((\psi \xrightarrow{}_{\mathrm{e}} \psi')^{\mathrm{L}} \cdot (g \gg g') \cdot (\phi \xrightarrow{}_{\mathrm{e}} \phi'))\, xy = \psi'^{\mathrm{L}}(\bigvee \{g'(\phi'(x(\phi^{\mathrm{L}} z))) \mid gz \leqslant \psi y\})$.

Now take

$$f: \underline{3} \to \underline{2} \text{ with } f\bot = \bot, f* = f\top = \top$$

$$\psi: \underline{2} \to \underline{3} \text{ with } \psi\bot = \bot, \psi\top = * \text{ (so } \psi^{\mathrm{L}} = f)$$

$$g = \phi = f' = g' = \phi' = \psi' = 1_{\underline{3}}$$

$$x = 1_{\underline{3}}, y = \top \in \underline{2};$$

then f, g, f', g' are abstractions (with $f^{*} = \psi$, $g^{*} = f'^{*} = g'^{*} = 1_{\underline{3}}$), ϕ, ψ, ϕ', ψ' are embeddings (with $\psi^{\mathrm{L}} = f$, $\phi^{\mathrm{L}} = \phi'^{\mathrm{L}} = \psi'^{\mathrm{L}} = 1_{\underline{3}}$) satisfying the conditions, and moreover

$$(\mathrm{a}) = \bigvee \{u \mid fu \leqslant \top\} = \top \neq * = \bigvee \{\bot, *\} = \bigvee \{z \mid z \leqslant \psi\top\} = (\mathrm{b}). \quad ∎$$

Finally we consider fixpoints of abstraction transformations, beginning with the simple case without parameters. We extend the definition of fixpoints of functors (see 2.4) to abstraction transformations.

4.13. DEFINITION. (i) Let $F, G \in \mathbb{F}(\mathbb{A}_{\mathrm{e}}, \mathbb{A}_{\mathrm{e}})$ and $H: F \Rightarrow G$. The arrow chain of H is defined by $\mathrm{arch}(H) = \langle H^{n} \bot \rangle_{n}$, the fixpoint of H by $\mathrm{Fix}(H) = \lim(\mathrm{arch}(H))$.

(ii) Now let $F, G \in \mathbb{F}(\mathbb{A}_e^m * \mathbb{A}_e, \mathbb{A}_e)$, $H: F \Rightarrow G$, $f \in \mathbb{A}_e^m$. Then $\operatorname{arch}(H_f) = \langle H_f^n \perp \rangle_n$ is the arrow chain of H in f, and $\operatorname{Fix}(H) = \lambda f.\operatorname{Fix}(H_f)$ is called the fixpoint of H.

4.14. LEMMA. *Let* $H: F \Rightarrow G$.

(i) $\langle (H^n \perp)_1 \rangle_n$ *is a strong arrow chain,* $\langle (H^n \perp)_2 \rangle_n$ *is an arrow chain.*

(ii) $\operatorname{Fix}(H)$ *is an abstraction pair.*

Proof. (i) Let $\phi: \perp \to F(\perp)$, $\psi: \perp \to F(\perp)$ be defined by $\phi\perp = \perp$, $\psi\perp = \perp$. We have to show, for all n,

$$G^n(\psi) \cdot (H^n \perp)_1 = (H^{n+1} \perp)_1 \cdot F^n(\phi),$$

$$(H^n \perp)_2 = F^n(\phi)^L \cdot (H^{n+1} \perp)_2 \cdot G^n(\psi),$$

i.e., $(F^n(\phi), G^n(\psi), H^n \perp, H^{n+1} \perp)$ is an abstraction diagram. This follows by induction over n. The base step is $\psi \cdot 1_\perp = (H \perp)_1 \cdot \phi$, $1_\perp = \phi^L \cdot (H \perp)_2 \cdot \psi$; the second equation is trivial, the first follows from the fact that H preserves abstractions, so $(H \perp)_1$ is strict. For the induction step we use $H: F \Rightarrow G$.

(ii) Let $f_n = (H^n \perp)_1$, $f_n^* = (H^n \perp)_2$, $f = (\operatorname{Fix}(H))_1, f^* = (\operatorname{Fix}(H))_2$, $\phi_n = F^n(\phi)$, $\psi_n = F^n(\psi)$; then $f = \lim_n f_n$, $f^* = \lim_n f_n^*$. We observe that, by (i), the $H^n \perp$ are abstraction pairs, so all f_n are $\perp$-unique, preserve compactness, and satisfy $f_n \cdot f_n^* = 1$. We shall show that (a) f is $\perp$-unique, (b) f preserves compactness, and (c) $f \cdot f^* = 1_{\operatorname{Fix}(G)}$.

(a) $f\perp = \langle \bigvee \{(\psi_n^L \cdot \cdots \cdot \psi_{n+m-1}^L)(f_{n+m}(\perp)) \mid m \in \omega\} \rangle_n = \langle \bigvee \{(\psi_n^L \cdot \cdots \cdot \psi_{n+m-1}^L)(\perp) \mid m \in \omega\} \rangle_n = \langle \bigvee \{\perp \mid m \in \omega\} \rangle_n = \perp$. On the other hand, if $f(\langle x_n \rangle_n) = \perp$, then $\bigvee \{(\psi_n^L \cdot \cdots \cdot \psi_{n+m-1}^L)(f_{n+m}(x_{n+m})) \mid m \in \omega\} = \perp$ for all n, so $(m := 0)\ f_n(x_n) = \perp$ for all n, hence $\langle x_n \rangle_n = \perp$.

(b) Let $\langle x_n \rangle_n \in \operatorname{Fix}(F)$ be compact; then $x_k \in c(X_k) \wedge \forall m \geqslant k (x_{m+1} = \phi_m(x_m))$ for some k. Now let $\langle y_n \rangle_n = f\langle x_n \rangle_n$, then for $n \geqslant k$

$$\begin{aligned} y_n &= \bigvee \{(\psi_n^L \cdot \cdots \cdot \psi_{n+m-1}^L \cdot f_{n+m})\, x_{n+m} \mid m \in \omega\} \\ &= \bigvee \{(\psi_n^L \cdot \cdots \cdot \psi_{n+m-1}^L \cdot f_{n+m} \cdot \phi_{n+m-1} \cdot \cdots \cdot \phi_k)\, x_k \mid m \geqslant n-k\} \\ &= (\psi_{n-1} \cdot \cdots \cdot \psi_k)(f_k x_k) \qquad (\psi_i \cdot f_i = f_{i+1} \cdot \phi_i), \end{aligned}$$

so $y_k \in c(Y_k)$ and $\forall m \geqslant k\ (y_{m+1} = \psi_m(y_m))$, i.e., $\langle y_n \rangle_n$ is compact.

(c) Easy, for $f \cdot f^* = (\lim_n f_n) \cdot (\lim_n f_n^*) = \lim_n (f_n \cdot f_n^*) = \lim_n 1 = 1$.

Conclusion: $\operatorname{Fix}(H) = \langle f, f^* \rangle$ is an abstraction pair. ∎

4.15. LEMMA. (i) *If* $H: F \Rightarrow G$ *and* p *is an abstraction pair, then* $H_p: F_{\mathrm{dom}(p)} \Rightarrow G_{\mathrm{cod}(p)}$.

(ii) *If* $H: F \Rightarrow G$, *then* $\mathrm{Fix}(H) : \mathrm{Fix}(F) \Rightarrow \mathrm{Fix}(G)$.

Proof. (i) Easy, using that $(1_{\mathrm{dom}(p)}, 1_{\mathrm{cod}(p)}, p, p)$ is an abstraction diagram.

(ii) Assume $H: F \Rightarrow G$ and let $(\phi_i, \psi_i, p_i, q_i)$ $(i = 1, ..., m)$ be abstraction diagrams with $p = \langle f, f^* \rangle \in \mathbb{A}_a^m$, $q = \langle g, g^* \rangle \in \mathbb{A}_a^m$, $\phi, \psi \in \mathbb{A}_e^m$. By (i) and Lemma 4.14(ii) we see that $\mathrm{Fix}(H)p$ and $\mathrm{Fix}(H)q$ are abstractions. So we need only show that $(\mathrm{Fix}(G)\,\psi) \cdot (\mathrm{Fix}(H)\,p)_1 = (\mathrm{Fix}(H)\,q)_1 \cdot (\mathrm{Fix}(F)\,\phi)$ and $(\mathrm{Fix}(H)\,p)_2 = (\mathrm{Fix}(G)\,\psi)^{\mathrm{L}} \cdot (\mathrm{Fix}(H)\,q)_2 \cdot (\mathrm{Fix}(F)\,\phi)$, i.e.,

$$\lim_n G_\psi^n \perp \cdot \lim_n (H_p^n \perp)_1 = \lim_n (H_q^n \perp)_1 \cdot \lim_n F_\phi^n \perp,$$
$$\lim_n (H_p^n \perp)_2 = (\lim_n G_\psi^n \perp)^{\mathrm{L}} \cdot \lim_n (H_q^n \perp)_2 \cdot \lim_n F_\phi^n \perp.$$

By commuting $\cdot$ and lim, this comes down to

$$G_\psi^n \perp \cdot (H_p^n \perp)_1 = (H_q^n \perp)_1 \cdot F_\phi^n \perp,$$
$$(H_p^n \perp)_2 = (G_\psi^n \perp)^{\mathrm{L}} \cdot (H_q^n \perp)_2 \cdot F_\phi^n \perp$$

for all n. This is proved by induction: $n = 0$ is trivial; the induction step requires

$$G(\psi, G_\psi^n \perp) \cdot (H(p, H_p^n \perp))_1 = (H(q, H_q^n \perp))_1 \cdot F(\phi, F_\phi^n \perp),$$
$$(H(p, H_p^n \perp))_2 = G(\psi, G_\psi^n \perp)^{\mathrm{L}} \cdot (H(q, H_q^n \perp))_2 \cdot F(\phi, F_\phi^n \perp),$$

and this follows from the induction hypothesis, $\psi \cdot f = g \cdot \phi$, $f = \psi^{\mathrm{L}} \cdot g \cdot \phi$, and $H: F \Rightarrow G$. ∎

We return to the abstraction interpretation abs. The associated category is $\mathbb{A}_a$, but we loosen the requirement $\mathrm{abs}(K) \in \mathbb{CF}(\mathbb{A}_a^n, \mathbb{A}_a)$ to

$$\text{if} \quad (K : {}^n\Theta) \in \mathbf{K}, \qquad \text{then} \quad \mathrm{abs}(K): C(K) \Rightarrow A(K);$$

furthermore we put

$$\mathrm{abs}[\![\rightarrow]\!] = \gg.$$

4.16. LEMMA. *If* $\vdash \Phi : {}^n\Theta$ *then* $\mathrm{abs}[\![\Phi]\!]\,a: C[\![\Phi]\!]\,\mathrm{dom}(a) \Rightarrow A[\![\Phi]\!]\,\mathrm{cod}(a)$ *for all* $a \in \mathrm{TENV}_{\mathrm{abs}}$.

Proof. By induction over the length of a derivation of $\vdash \Phi : {}^n\Theta$, strengthening the conclusion as in the proof of Lemma 3.2.1, and using 4.10, 4.14, 4.15. ∎

We now derive $\mathrm{abs} \cdot C \leqslant A$, the main result on C, A, and abs.

4.17. THEOREM. *If* $\Gamma \vdash p : \Phi$ *and* $a \in \mathrm{TENV}_{\mathrm{abs}}$, $b \in \mathrm{OENV}_C$, $b' \in \mathrm{OENV}_A$ *satisfy*

$$\text{for all } (x : \sigma) \in \Gamma, \qquad \mathrm{abs}[\![\sigma]\!]\, a(bx) \leqslant b'x, \tag{Γ}$$

then

$$\mathrm{abs}[\![\Phi]\!]\, a \cdot (C[\![p]\!](\mathrm{dom}(a))b) \leqslant A[\![p]\!](\mathrm{cod}(a))b'. \tag{p, Φ}$$

Proof. Induction over the length of a derivation of $\Gamma \vdash p : \Phi$. We write d for $\mathrm{dom}(a)$, c for $\mathrm{cod}(a)$, and inspect the various cases. Observe that (t, τ) reads $\mathrm{abs}[\![\tau]\!]\, a(C[\![t]\!]\, db) \leqslant A[\![t]\!]\, cb'$.
$p = x$, $\Phi = \tau$: now $(x : \tau) \in \Gamma$ and (x, τ) is part of (Γ).
p is a constant: then (p, Φ) follows from (3).
$p = st$, $\Phi = \tau$: (st, τ) follows from the induction hypothesis and (7).
$p = \lambda y : \sigma.t$, $\Phi = \sigma \to \tau$: then $\Gamma, y : \sigma \vdash t : \tau$ is the premiss of the conclusion $\Gamma \vdash \lambda y : \sigma.t : \tau$. Assuming Γ, we must show $(\lambda y : \sigma.t, \sigma \to \tau)$, i.e. (after some rewriting, using $\bigvee X \leqslant z \Leftrightarrow \forall x \in X\, x \leqslant z$),

$$\forall e' \in A[\![\sigma]\!]c\ (\mathrm{abs}[\![\sigma]\!]a(by) \leqslant e' \Rightarrow \mathrm{abs}[\![\tau]\!]a(C[\![t]\!]db) \leqslant A[\![t]\!]c(b'[y \to e'])),$$

and this is equivalent to $(y : \sigma) \Rightarrow (t, \tau)$, which follows from Γ by the induction hypothesis. The other cases are straightforward. ∎

This completes the development of abs and A: now

$$A[\![t]\!]\ \mathrm{strict} \Rightarrow C[\![t]\!]\ \mathrm{strict} \tag{6}$$

holds, for we have (5) (Theorem 4.17) and (7) (i.e., $\mathrm{abs}[\![\tau]\!](xy) \leqslant \mathrm{abs}[\![\sigma \to \tau]\!]x\, \mathrm{abs}[\![\sigma]\!]y$), so A provides a safe strictness analysis.

4.18. We end this section with some examples of (polymorphic) constants and their abstractions, using the notation $x \vee y = \bigvee \{x, y\}$, $x \wedge y = \bigwedge \{x, y\}$.

The usual operations associated with products and sums of types (pairing, projection, case distinction, etc.) remain unchanged under abstraction, due to the fact that $\times$ and $+$ remain unchanged. The usual unary and binary operations on N and B (plus, times, and, or, not, etc.) become $\lambda x.x$ and $\lambda xy.x \wedge y$, respectively, expressing that they are strict in all their arguments. More interesting are the conditional and the parallel (or Plotkin) conditional:

$$\textit{cond}, \textit{pcond} : \delta\alpha.B \times \alpha \times \alpha \to \alpha$$
$$C[\![\textit{cond}]\!] = \mathrm{cond},\ C[\![\textit{pcond}]\!] = \mathrm{pcond}$$
$$A[\![\textit{cond}]\!] = \mathrm{abscond},\ A[\![\textit{pcond}]\!] = \mathrm{abspcond}$$

where cond, pcond, abscond and abspcond are defined by

$$\text{cond}(\top, x, y) = x \vee y$$

$$\text{cond}(\text{true}, x, y) = x$$

$$\text{cond}(\text{false}, x, y) = y$$

$$\text{cond}(\bot, x, y) = \bot$$

$$\text{pcond}(b, x, y) = \text{cond}(b, x, y) \text{ if } b \neq \bot$$

$$\text{pcond}(\bot, x, y) = x \wedge y$$

$$\text{abscond}(b, x, y) = b \wedge (x \vee y)$$

$$\text{abspcond}(b, x, y) = (b \wedge x) \vee (b \wedge y) \vee (x \wedge y)$$

5. Concluding Remarks

5.1. *A Simplification: No Right Inverses to Abstractions*

The only reason for adding the requirement of having a right inverse in the definition of abstraction lies in the proof of $((f \gg f) \gg f)\,\text{fix} = \text{fix}$. This enabled us to put $A(\text{fix}) = \lambda X \in \text{Obj}(\mathbb{A}).\text{fix}_X$, i.e., to interpret the (polymorphic) fixpoint operator of **L** by the "real" fixpoint operator in the abstract interpretation. However, without right inverse to abstractions we are still able to prove $((f \gg f) \gg f)\,\text{fix} \leqslant \text{fix}$, and this is enough for us to admit the definition $A(\text{fix}) = \lambda X \in \text{Obj}(\mathbb{A}).\text{fix}_X$ while retaining the property $\text{abs} \cdot C \leqslant A$.

We indicate briefly the consequences of this simplification.

1. The interpretation C can be defined in the category $\mathbb{D}$ of domains, which is in some sense more natural. Then $+$ can be interpreted by the usual separated sum of two domains (i.e., not adding a top element as in the definition given in 2.2). The abstract interpretation A remains in the category $\mathbb{A}$ of algebraic complete lattices. But now it is no longer possible to find a right inverse for $f + g$: the new top element of $\text{cod}(f) + \text{cod}(g)$ has no counterpart in $\text{dom}(f) + \text{dom}(g)$.

2. The category $\mathbb{A}_a$ has to be replaced by the collection of abstractions, a subclass of $\mathbb{D}$ which is not a category (if f is an abstraction and $\text{dom}(f)$ is not a lattice, then $1_{\text{dom}(f)}$ is not an abstraction).

3. The definition of abstraction diagram can be simplified to tuples (ϕ, ψ, f, g) with $\psi \cdot f = g \cdot \phi$; similarly for abstraction transformations.

Inspection of the proofs shows that right inverses f^* of abstractions f never occur in arguments involving abstractions (with the exception of the proof of $((f \gg f) \gg f)\,\text{fix} = \text{fix}$, see 4.7).

Another option is to work in $\mathbb{D}$ with the $+$ of $\mathbb{A}$, i.e., with $+ \in \mathbb{F}(\mathbb{D}^2, \mathbb{A})$. This makes the interpretation C somewhat more natural, but we chose not do so here in order to make the argument more smoothly.

5.2. *Perspectives for Future Research*

The following items, not covered in this paper, seem of interest for future research.

(i) Extension of the method to stronger languages, e.g., second-order lambda calculus or other versions of typed lambda calculus (see Barendregt, 1990). The obvious question is, given some concrete interpretation, to find an abstract interpretation and an abstraction mapping between the two.

(ii) Generalisation, in the style of Nielson (1988), using adjunctions, of the induction of the abstract interpretation A from the concrete interpretation C.

(iii) Refinement of the method, e.g., by considering finite approximations of limits of chains, in order to make computation of demonstrable strictness feasible.

ACKNOWLEDGMENTS

This work was supported by the Dutch Parallel Reduction Machine Project, sponsored by the Dutch Ministry of Science and Education (Science Council). The author thanks Samson Abramsky, Chris Hankin, and Piet Rodenburg for useful discussions, and Henk Barendregt for many stimulating meetings concerning the subject of this paper and other matter. Comments, provided by two referees on an earlier version, led to substantial improvement of form and content of this paper.

RECEIVED October 4, 1988; FINAL MANUSCRIPT RECEIVED November 2, 1990

REFERENCES

ABRAMSKY, S. (1985), Strictness analysis and polymorphic invariance, *in* "Programs as Data Objects" (H. Ganzinger and N. D. Jones, Eds.), pp. 1–23, Lecture Notes in Computer Science, Vol. 217, Springer-Verlag, Berlin/New York.

ARBIB, M. A., AND MANES, E. G. (1975), "Arrows, Structures and Functors: The Categorical Imperative," Academic Press, New York.

BARENDREGT, H. P. (1992), Lambda calculus with types, *in* "Handbook of Logic in Computer Science" (S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, Eds.), Oxford Univ. Press, to appear.

BARENDREGT, H. P., KENNAWAY, J. R., KLOP, J. W., AND SLEEP, M. R. (1987), Needed reduction and spine strategies for the lambda calculus, *Inform. and Comput.* **75**, 191–231.

BURN, G. L., HANKIN, C. L., AND ABRAMSKY, S. (1986), Strictness analysis for higher order functions, *Sci. Comput. Programming* **7**, 249–278.

COUSOT, P. AND COUSOT, R. (1977), Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, *in* "Conference Record of the 4th ACM Symposium on Principles of Programming Languages," pp. 238–252.

COUSOT, P., AND COUSOT, R. (1979), Systematic design of program analysis frameworks, *in* "Conference Record of the 6th ACM Symposium on Principles of Programming Languages," pp. 238–252.

HUGHES, J. (1987), Backwards analysis of functional programs, manuscript.

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

MYCROFT, A. (1980), The theory and practice of transforming call-by-need into call-by-value, *in* "Proceedings, International Symposium on Programming," pp. 269–281, Lecture Notes in Computer Science, Vol. 83, Springer-Verlag, Berlin/New York.

MYCROFT, A. (1981), "Abstract Interpretation and Optimising Transformations for Applicative Programs," Ph.D. Thesis, Edinburgh University.

MYCROFT, A. (1984), Polymorphic type schemes and recursive definition, *in* "International Symposium on Programming" (M. Paul and B. Robinet, eds.), Lecture Notes in Computer Science, Vol. 167, Springer-Verlag, Berlin/New York.

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

NIELSON, F. (1986), Abstract interpretations of denotational definitions, *in* "Proceedings, STACS 1986," Lecture Notes in Computer Science, Vol. 210, Springer-Verlag.

NIELSON, F. (1988), Strictness analysis and denotational abstract interpretation, *Inform. and Comput.* **76**, 29–92.

RENARDEL DE LAVALETTE, G. R. (1988), "Strictness Analysis with POLYREC, A Language with Polymorphic and Recursive Types," Logic Group Preprint Series No. 33, Department of Philosophy, University of Utrecht, The Netherlands.

SCOTT, D. S. (1976), Data types as lattices, *SIAM J. Comput.* **5**, 522–587.

SCOTT, D. S. (1982), Domains for denotational semantics, *in* "9th International Colloquium on Automata, Languages and Programming" (M. Nielson and E. Schmidt, Eds.), pp. 577–613, Lecture Notes in Computer Science, Vol. 140, Springer-Verlag, Berlin/New York.

SMYTH, M., AND PLOTKIN, G. (1982), The category-theoretic solution of recursive domain equations, *SIAM J. Comput.* **11**, 761–783.