Strictness analysis via abstract interpretation
for recursively defined types

Gerard R. Renardel de Lavalette

Strictness analysis via abstract interpretation for recursively defined types

Gerard R. Renardel de Lavalette

University of Utrecht, Department of Philosophy
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands

Abstract

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

October 1990

1 INTRODUCTION.

A function $f : X \to Y$ (X, Y partial orderings with bottom element $\bot$) is called *strict* if $f\bot = \bot$. Strictness is the semantical counterpart of *neededness*, defined by: 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 [BKKS86] 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 & Cousot ([CC77], [CC79]), is a general theory for static analysis of programs. The idea behind is: 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, termination.

A frequently studied application of abstract interpretation is strictness analysis. The first work in this direction is performed by Mycroft ([M80], [M81]) 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 [BHA86]. [A85] contains a generalisation to polymorphic types ($\tau ::= c \mid \alpha \mid \tau \to \tau$), obtained by adding free type variables: via an operational semantics, it is shown that stricness 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 $\sigma ::= \tau \mid \forall \alpha.\sigma$), is shortly discussed in a remark at the end of [A85].

A more general treatment of abstract interpretations is found in the work of Nielson ([N84], [N86], [N88]). In [N88], a general theory on 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 with the language in [BHA86]. The strictness analysis of [BHA86] is treated in [N88] as an example. A difference between the denotational semantics used in [BHA86] and [N88] lies in the function spaces used for the interpretation of types $\sigma \to \tau$: in [BHA86] the functions are continuous, in [N88] 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 ([N88], 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 [N88], p. 76.

In this paper, we extend strictness analysis, based on abstract interpretation in the style of [BHA86] and [N88], to a language with polymorphic and recursively defined types ($\tau ::= \alpha \mid \mu\Phi$; type constructors $\Phi ::= K \mid \rightarrow \mid \tau \mid \Phi\tau \mid \delta\alpha.\Phi$). 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 ⊥-unique, preserves compactness and has a continuous right inverse.

We give a survey of what follows. In §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 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 3.2, with the intended interpretation C of L as an instance. In §4 another instance, the abstract interpretation A, is derived using the type interpretation abs (which is *almost* an interpretation in the sense of 3.2). §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. §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 [BHA86] is based on effective computations in finite complete lattices. However, as has been pointed out by several authors (see [A85], [H87]), 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 [H87].

2

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 [R88].

2.1. Complete partial orderings.

(See [S76], [S82] for more information.) $X = \langle X, \leq \rangle$ 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 $D(X) = \{ Y \subseteq X \mid Y \neq \emptyset \wedge \forall xy \in Y \exists z \in Y(x \leq z \wedge y \leq z) \}$ 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 $fix = fix_X \in [[X \to X] \to X]$ (X some cpo) by $fix(f) := \bigvee \{ f^n \bot \mid n \in \omega \}$, satisfying $f(fix(f)) = fix(f)$ and $\forall x \in X(fx = x \to fix(f) \leq x)$.

An element x of a cpo X is called *compact* (also called finite or isolated elsewhere) if $\forall Y \in D(X)(x \leq \bigvee Y \to \exists y \in Y(x \leq y))$. We put $c(X) = \{ x \in X \mid x \text{ compact} \}$, $\Downarrow_c(x) = c(X) \cap \{ y \mid y \leq x \}$. 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 $\exists x \in X \forall y \in Y(y \leq x)$ 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 $\mathbf{a} = a_1, \dots, a_n$ and $\mathbf{b} = b_1, \dots, b_n$ be finite sequences of elements of $c(X)$ resp. $c(Y)$. We define $Adm(\mathbf{a}, \mathbf{b}) := \forall i,j \leq n((a_i \leq a_j \to b_i \leq b_j) \wedge \forall x \in X(a_i, a_j \leq x \to \exists k \leq n(a_i, a_j \leq a_k \leq x)))$ and $f_{\mathbf{ab}}(x) := \bigvee \{ b_i \mid a_i \leq x \}$. Functions of the form $f_{\mathbf{ab}}$ are called finite, and admissible if also $Adm(\mathbf{a}, \mathbf{b})$ holds. (We shall use these notions later on, in the proof of (c) in lemma 4.4.(i).) 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 $\bigvee \{ x \in X \mid \forall z \in Z(x \leq z) \}$. If $f : X \to Y$ satisfies $f(\bigvee Z) = \bigvee(fZ)$ for all $Z \subseteq X$, then f is called *additive*.

3

Let X, Y be cpo's, $\phi \in [X \to Y]$. ϕ is called an *embedding* if (i) $\forall xy \in X(x \leq y \leftrightarrow \phi x \leq \phi y)$ and (ii) $\forall y \in Y(\{x \mid \phi x \leq y\} \in D(X))$. Embeddings ϕ have a continuous left inverse ϕ^L defined by $\phi^L(y) = \vee\{x \mid \phi x \leq y\}$ satisfying besides (iii) $\phi^L \cdot \phi = id_X$ also (iv) $\phi \cdot \phi^L \leq id_Y$. Conversely, if ϕ has a left inverse ϕ^L satisfying (iii) and (iv) then ϕ is an embedding. Pairs $<\phi,\phi^L>$ 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 $\vee\{\phi z \mid z \in Z\} \leq \phi(\vee Z) = \phi(\vee\{\phi^L(\phi z) \mid z \in Z\}) \leq (\phi \cdot \phi^L)(\vee\{\phi z \mid z \in Z\}) \leq \vee\{\phi z \mid z \in Z\}$).

Some examples of complete lattices:

$\underline{\bot} := <\{\bot\}, \leq>$

$\underline{2} := <\{\bot,\top\}, \leq>$, with $\bot < \top$

$\underline{3} := <\{\bot,*,\top\}, \leq>$, with $\bot < * < \top$

$\underline{B} := <\{\bot, true, false, \top\}, \leq>$, with $x \leq y$ iff ($x=\bot$ or $x=y$ or $y=\top$)

$\underline{N} := <\{\bot,0,1,2,3,...,\top\}, \leq>$, with $x \leq y$ iff ($x=\bot$ or $x=y$ or $y=\top$)

Besides $\to$, 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. [AM75], [M71] for more information.) Categories $\mathbb{C}$, consisting of objects $Obj(\mathbb{C})$ and arrows $Ar(\mathbb{C})$, 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 $Obj(\mathbb{C}) \subseteq Ar(\mathbb{C}) = \mathbb{C}$. Given two categories $\mathbb{C}$ and $\mathbb{C}'$, the product category is denoted by $\mathbb{C}*\mathbb{C}'$, and for $\mathbb{C}*...*\mathbb{C}$ (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 $H : Obj(\mathbb{C}) \to Ar(\mathbb{C}')$ is a *natural transformation* between F, G $\in \mathbb{F}(\mathbb{C},\mathbb{C}')$ if $G(f) \cdot H(dom(f)) = H(cod(f)) \cdot F(f)$ for all $f \in Ar(\mathbb{C})$.

In the sequel, we shall 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 $\leq$; such a functor is also a functor from $\mathbb{D}_e^n$ to $\mathbb{D}_e$, with $F(<\phi_1,\psi_1>,..., <\phi_n,\psi_n>) = <F(\phi_1,...,\phi_n),F(\psi_1,...,\psi_n)>$. Idem for $\mathbb{A}$ instead of $\mathbb{D}$.

4

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 $F(\phi_1, ..., \phi_n)^L = F(\phi_1^L, ..., \phi_n^L)$), 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 $\mathbb{F}(\mathbb{A}^{op} * \mathbb{A}, \mathbb{A})$. The contravariance in the first argument prevents straightforward extension to $\mathbb{A}_e$, therefore we define $\rightarrow_e$ on $\mathbb{A}_e^2$ by $(\phi \rightarrow_e \psi) = (\phi^L \rightarrow \psi)$. Using $(\phi \rightarrow_e \psi)^L = (\phi \rightarrow \psi^L)$, it follows that $(\phi \rightarrow_e \psi)$ is again an embedding and $\rightarrow_e \in \mathbb{F}(\mathbb{A}_e^2, \mathbb{A}_e)$. Observe that $\rightarrow$ and $\rightarrow_e$ 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 [SP82] 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 $\lim \langle X_n, \phi_n \rangle_n = \{ \langle x_n \rangle_n \mid \forall n(x_n \in X_n \wedge \phi_n^L(x_{n+1}) = x_n) \}$, 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 $c(\lim_n X_n) = \{ \langle x_n \rangle_n \mid \exists n(x_n \in c(X_n) \wedge \forall m \geq n(x_{m+1} = \phi_m(x_m))) \}$. Being algebraic, boundedly completeness and having a top element are all preserved under taking limits; as a consequence, $\mathbb{A}_e$ is closed under lim.

Functors F on $\mathbb{A}_e$ are *continuous* if they commute with taking limits, i.e. (for unary F) if $F(\lim \langle X_n, \phi_n \rangle_n) = \lim_n \langle FX_n, F\phi_n \rangle_n$. $\times, +$ and $\rightarrow_e$ are examples of continuous functors. $\mathrm{ch}(F)$, the *chain of functor* F is defined as $\langle F^n \bot, F^n \phi \rangle_n$, where ϕ is the unique embedding from $\bot$ to $F\bot$ (mapping $\bot$ to $\bot$). $\mathrm{Fix}(F)$, the *fixpoint of* F, is defined as $\lim(\mathrm{ch}(F))$ 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 $\rightarrow_e$, $[X \rightarrow Y] = \lim_n [X_n \rightarrow Y_n]$, so elements f of $[X \rightarrow Y]$ are identified with sequences $\langle f_n \rangle_n \in \lim_n [X_n \rightarrow Y_n]$, i.e. satisfying (i) $f_n = (\phi_n \rightarrow_e \psi_n)^L f_{n+1} = \psi_n^L \cdot f_{n+1} \cdot \phi_n$ for all n; such sequences $\langle f_n \rangle_n$ are called *arrow chains*, and the corresponding $f \in [X \rightarrow Y]$ 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 $(\lim_n f_n)(\langle x_n \rangle_n) = \langle \bigvee \{ (\psi_n^L \cdot ... \cdot \psi_{n+m-1}^L)(f_{n+m}(x_{n+m})) \mid m \in \omega \} \rangle_n$; also $\lim_n f_n \leq \lim_n g_n$ iff $f_n \leq g_n$ 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.

5

Let F be an $(n+1)$-ary functor on $\mathbb{A}_e$ and $\phi = \langle\phi_1,\ldots,\phi_n\rangle \in \mathrm{Ar}(\mathbb{A}_e{}^n)$. Then $F_{\mathrm{dom}(\phi)}$ and $F_{\mathrm{cod}(\phi)}$ are unary functors on $\mathbb{A}_e$ and their chains have limits in $\mathrm{Obj}(\mathbb{A}_e)$, 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, the so is $\mathrm{Fix}(F)$ (proved via permutation of limits of chains).

Some nonstandard notation used in the sequel:

$$^n\mathbb{C} =_{\mathrm{def}} \mathbb{CF}(\mathbb{C}^n, \mathbb{C})$$
$$^*\mathbb{C} =_{\mathrm{def}} \bigcup\{^n\mathbb{C} \mid n=0,1,\ldots\}$$
$$_n\mathbb{C} =_{\mathrm{def}} \mathbb{C}^n \to \cup\mathrm{Obj}(\mathbb{C})$$
$$_*\mathbb{C} =_{\mathrm{def}} \bigcup\{_n\mathbb{C} \mid n=0,1,\ldots\}$$

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

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

3 THE LANGUAGE AND ITS INTERPRETATION

3.1. The language **L**.

L is a typed lambda calculus, defined in the style of [B90]. It has expressions of three sorts: object, type and kind. There are two sorts of statements:

<type expression> : <kind expression>,
<object expression> : <type expression>.

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 $\mathbf{K} = \mathbf{K}_{\mathrm{type}} \cup \mathbf{K}_{\mathrm{object}}$ acts as a parameter of **L**. $\mathbf{K}_{\mathrm{type}}$ contains statements of the form $K : {}^n\Theta$ where K is a type constant; $\mathbf{K}_{\mathrm{object}}$ 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 : Θ (the natural numbers), B : Θ (the Booleans), together with their usual unary and binary operations like $+ : N \to N \to N$, not $: B \to B$, etc.;

ii) $\times : {}^2\Theta$ and $+ : {}^2\Theta$, the type operators of product and sum, and the associated polymorphic operations like pair $: \delta\alpha\beta.\alpha\to\beta\to(\alpha\times\beta)$, case $: \delta\alpha\beta\gamma.(\alpha\to\gamma)\to(\beta\to\gamma)\to(\alpha\times\beta)\to\gamma$, 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,...; 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	$\kappa ::= {}^0\Theta \mid {}^1\Theta \mid {}^2\Theta \mid \ldots$
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	$p ::= k \mid fix \mid t \mid p\tau \mid \Lambda\alpha.p$

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, $s(t_1,\ldots,t_n)$ means $st_1\ldots t_n$, $\lambda x_1{:}\tau_1\ldots x_n{:}\tau_n.t$ means $\lambda x_1{:}\tau_1.(\ldots(\lambda x_n{:}\tau_n.t)\ldots)$; $\to\sigma\tau$ 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: 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 $\Gamma \vdash E_2 : E_3$.

Axoims and rules for types:

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

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

Axioms and rules for terms:

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

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

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

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 $L := \delta\alpha.\mu(\delta\beta.1{+}(\alpha{\times}\beta))$; secondly, 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. <u>Interpretations</u>.

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

$$\text{if } (K{:}^{n}\Theta) \in \mathbf{K}_{type} \text{ then } I(K) \in {}^{n}\mathbb{C}_I;$$
$$\text{if } (k{:}\Phi) \in \mathbf{K}_{object} \text{ then } I(k) \; \varepsilon \; I[\![\Phi]\!].$$

($^{n}\mathbb{C}_I$, and ε anr defined at the end of 2.4.) We extend I to a mapping defined on all wellformed expressions of $\mathbf{L}$. This is done as usual with assignments (also called

environments), which are mappings defined of variables of some syntactical category. If a is such an assignment, x a variable and d an object, then a':=a[x→d] is the assignment 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:

$$TENV_I =_{def} TVAR \rightarrow \mathbb{C}_I$$
$$OENV_I =_{def} OVAR \rightarrow {}_0\mathbb{C}_I$$

Now we extend I to

$$I : KIND \rightarrow \{ {}^n\mathbb{C}_I \mid n=0,1,... \}$$
$$I : CONSTR \rightarrow TENV_I \rightarrow {}^*\mathbb{C}_I$$
$$I : PTERM \rightarrow TENV_I \rightarrow OENV_I \rightarrow {}_*\mathbb{C}_I$$

We adopt the usual convention of writing the first argument of I between open square brackets [ ]. The other arguments $a \in TENV_I$, $b \in OENV_I$ 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).

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

$$I[\alpha]a = a(\alpha)$$
$$I[K]a = I(K)$$
$$I[\rightarrow]a = \rightarrow$$
$$I[\Phi\tau]a = I[\Phi]a(I[\tau]a)$$
$$I[\delta\alpha.\tau]a = \lambda f \in \mathbb{C}_I.I[\tau](a[\alpha \rightarrow f])$$
$$I[\mu\Phi]a = Fix(I[\Phi]a)$$

$$I[x]ab = b(x)$$
$$I[k]ab = I(k)$$
$$I[fix]ab = \lambda f \in \mathbb{C}_I.fix_{cod(f)}$$
$$I[st]ab = (I[s]ab)(I[t]ab)$$
$$I[\lambda x:\sigma.t]ab = \lambda e \in I[\sigma]a.I[t]a(b[x \rightarrow e])$$
$$I[p\tau]ab = I[p]ab(I[\tau]a)$$
$$I[\Lambda\alpha.t]ab = \lambda f \in \mathbb{C}_I.I[t](a[\alpha \rightarrow f])b$$

We also put

9

$(I,a,b) \vDash \Phi : \kappa =_{def} I[\![\Phi]\!]a \in I[\![\kappa]\!]$

$(I,a,b) \vDash p : \Phi =_{def} I[\![p]\!]ab \; \epsilon \; I[\![\Phi]\!]a$

$(I,a,b) \vDash \Gamma =_{def} (I,a,b) \vDash S$ for all $S \in \Gamma$

$\Gamma \vDash_I S =_{def}$ for all $a \in TENV_I$, $b \in OENV_I$: if $(I,a,b) \vDash \Gamma$ then $(I,a,b) \vDash S$

$\Gamma \vDash S =_{def} \Gamma \vDash_I S$ for all interpretations I

3.2.1. <u>Lemma</u> (soundness). If $\Gamma \vdash S$ then $\Gamma \vDash S$.

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

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

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

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

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

3.3 <u>The concrete interpretation C.</u>

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

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 $abs(K) : \mathbb{D}^n \to \mathbb{D}$. It will turn out that $abs(\to)$ 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 $K : {}^n\Theta$):

(1) $dom(abs(K)f) = C(K)(dom(f))$,

(2) $cod(f) = cod(g) \Rightarrow cod(abs(K)f) = cod(abs(K)g)$.

Now the abstract interpretation A is defined on types by

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

Then:

$$cod(abs(K)f) = A(K)(cod(f)).$$

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

(3) $A(k)(A[\tau_1],...,A[\tau_n]) \geq abs[\Phi\tau_1...\tau_n](C[k\tau_1...\tau_n])$ for all wellformed types $\tau_1,...,\tau_n$

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

(4) $dom(abs[\tau]) = C[\tau]$, $cod(abs[\tau]) = A[\tau]$ for $\tau : {}^0\Theta$
(5) $abs[\tau](C[t]) \leq A[t]$ for $t : \tau : {}^0\Theta$

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

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

(6) $A[t]$ strict $\Rightarrow$ $C[t]$ strict,

property (5) and

(7) $abs[\tau](xy) \leq abs[\sigma\rightarrow\tau]x\ abs[\sigma]y$

11

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 $abs[\sigma \to \tau]$, defined in terms of $abs[\sigma]$ and $abs[\tau]$, is smaller. We claim that this is obtained by defining

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

to verify the claim, we observe:

i) $abs[\sigma \to \tau]x\, abs[\sigma]y = \bigvee\{abs[\tau](xu) \mid abs[\sigma]x \leq abs[\sigma]y\} \geq abs[\tau](xy);$

ii) if f satisfies $abs[\tau](xy) \leq fx(abs[\sigma]y)$, then we have $z \geq abs[\sigma]y \Rightarrow fxz \geq abs[\tau](xy)$, so fxz is an upper bound of $\{abs[\tau](xu) \mid abs[\sigma]x \leq abs[\sigma]y\}$, hence $abs[\sigma \to \tau] \leq f$.

We conclude: $abs[\sigma \to \tau]$ as defined by (8) is the smallest satisfying (7).

(6) is verified as follows:

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

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 at the definition of A and abs after 4.15.

4.1. <u>Definition</u>.

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. $f \cdot f^* = 1_Y$). $<f,f^*>$ is called an *abstraction pair*.

ii) A_a is the category of abstraction pairs, with composition defined by

$$<f,f^*> \cdot <g,g^*> = <f \cdot g, g^* \cdot f^*>.$$

4.2. <u>Remarks</u>.

i) The reasons for the definition of abstractions are the following: strictness is used in the proof of 4.7 $(((f \gg f) \gg f)fix \leq fix)$, 4.14.(i) and (6); $\bot$-preservation is needed in the argument for (6);

preservation of compactness is required in the proof that $(f»g)z$ is continuous whenever z is (4.4.(i).(c)).

ii) 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 $\bot$-unique function in $N \to \underline{2}$ and $B \to \underline{2}$ respectively.

4.3. Definition.

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

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

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

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

$$<f,f^*> » <g,g^*> = <f»g, f\to g^*>.$$

4.4. Lemma.

i) $\times, + \in \mathbb{F}(A_a{}^2, A_a)$;

ii) $» : A_a{}^2 \to A_a$ ($»$ is not a functor: see the remark after the proof)

<u>Proof.</u> (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 $<f,f^*>$, $<g,g^*>$ 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:

(a) $(f»g)(z)$ is continuous,

(b) $f»g$ is continuous,

(c) $f»g$ preserves compactness,

(d) $f»g$ is $\bot$- unique,

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

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

$$(f»g)(z)(\bigvee Z) = \bigvee\{g(zx) \mid fx \le \bigvee Z\}$$
$$= \bigvee\{g(z(\bigvee\Downarrow_c(x))) \mid fx \le \bigvee Z\} \qquad (X \text{ is algebraic})$$
$$= \bigvee\{\bigvee\{g(zx') \mid x'\in \Downarrow_c(x)\} \mid fx \le \bigvee Z\} \qquad (\Downarrow_c(x) \text{ directed; } g,z \text{ continuous})$$
$$= \bigvee\{g(zx') \mid x'\le x, \ x' \text{ compact}, \ fx \le \bigvee Z\}$$

$$= \bigvee\{g(zx') \mid x' \text{ compact}, fx' \leq \bigvee Z\}$$
$$= \bigvee\{g(zx') \mid x' \text{ compact}, \exists y \in Z \ fx' \leq y\} \quad (f \text{ preserves compactness})$$
$$= \bigvee\{\bigvee\{g(zx') \mid x' \text{ compact}, fx' \leq y\} \mid y \in Z\}$$
$$= \bigvee\{\bigvee\{g(zx') \mid x' \leq x, x' \text{ compact}, fx \leq y\} \mid y \in Z\}$$
$$= \bigvee\{\bigvee\{\bigvee\{g(zx') \mid x' \in \Downarrow_c(x)\} \mid fx \leq y\} \mid y \in Z\}$$
$$= \bigvee\{\bigvee\{g(zx) \mid fx \leq y\} \mid y \in Z\}$$
$$= \bigvee\{(f \gg g)(z)(y) \mid y \in Z\}.$$

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

$$(f \gg g)(\bigvee Z) = \lambda y . \bigvee\{g((\bigvee Z)(x)) \mid fx \leq y\}$$
$$= \lambda y . \bigvee\{g(\bigvee\{zx \mid z \in Z\}) \mid fx \leq y\}$$
$$= \lambda y . \bigvee\{\bigvee\{g(zx) \mid z \in Z\} \mid fx \leq y\} \quad (g \text{ is continuous})$$
$$= \lambda y . \bigvee\{\bigvee\{g(zx) \mid fx \leq y\} \mid z \in Z\}$$
$$= \lambda y . \bigvee\{(f \gg g)(z)(y) \mid z \in Z\}$$
$$= \bigvee\{(f \gg g)(z) \mid z \in Z\}$$

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

$$(f \gg g)(z) = \lambda y . \bigvee\{g(zx) \mid fx \leq y\}$$
$$= \lambda y . \bigvee\{g(\bigvee\{b_i \mid a_i \leq x\}) \mid fx \leq y\}$$
$$= \lambda y . \bigvee\{gb_i \mid a_i \leq x, fx \leq y\}$$
$$= \lambda y . \bigvee\{gb_i \mid fa_i \leq y\}$$
$$= z_{f\mathbf{a}, g\mathbf{b}}$$

so $(f \gg g)(z)$ is compact (by lemma 3.17.(ii), for $f\mathbf{a}, g\mathbf{b}$ 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 $(f \gg g)(z) = \bot \Leftrightarrow \forall y \ \bigvee\{g(zx) \mid fx \leq y\} = \bot \Leftrightarrow \forall x \ (g(zx) = \bot) \Leftrightarrow \forall x \ (zx = \bot) \Leftrightarrow z = \bot$, 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 $(f \gg g)((f \to g^*)z)y = \bigvee\{g(((f \gg g)^* z)x) \mid fx \leq y\} = \bigvee\{g((g^* \cdot z \cdot f)(x)) \mid fx \leq y\} = \bigvee\{z(fx) \mid fx \leq y\} = z(f(f^* y)) = zy$. So $(f \gg g)((f \to g^*)z) = z$, i.e. $(f \gg g) \cdot (f \to g^*) = \text{id}_{[Y \to Y']}$. $\square$

4.5. <u>Remark.</u> $\gg$ is *not* a functor on $\mathbb{A}_a$. For if so, then $\cdot$ and $\gg$ would have to 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.

14

4.6. <u>Counterexample</u>. There is an abstraction $f : X \to Y$ with $f \gg f \neq f \cdot (f \gg 1_X)$.
<u>Proof</u>. Define

$$X = \{\bot, 0, 1, \top\}, \; Y = \underline{3}, \; Z = [X \to Y],$$
$$f\bot = \bot, \; f0 = f1 = *, \; f\top = \top.$$

We observe $f \gg 1_X = \lambda z y. \bigvee \{zx \mid fx \leq y\}$. Now

$$(f \gg f)(1_Z)* = \bigvee \{fx \mid fx \leq *\} = \bigvee \{\bot, *\} = *,$$
$$(\mathrm{cod}(f) \gg f) \cdot (f \gg \mathrm{dom}(f))(1_Z)* = f((f \gg 1_X)1_Z)* = f(\bigvee \{x \mid fx \leq *\}) = f\top = \top. \qquad \square$$

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

4.7. <u>Lemma</u>. If $f : X \to Y$ is an abstraction, then $((f \gg f) \gg f)\mathrm{fix}_X = \mathrm{fix}_Y$.
<u>Proof</u>. Let $a \in [Y \to Y]$. Then

$$(((f \gg f) \gg f)\mathrm{fix})a = \bigvee \{f(\mathrm{fix}(z)) \mid (f \gg f)z \leq a\}$$
$$= \bigvee \{f(\bigvee \{z^n \bot \mid n \in \omega\}) \mid (f \gg f)z \leq a\}$$
$$= \bigvee \{\bigvee \{f(z^n \bot) \mid n \in \omega\} \mid (f \gg f)z \leq a\}$$
$$= \bigvee \{f(z^n \bot) \mid n \in \omega, (f \gg f)z \leq a\}$$
$$= \bigvee \{f(z^n \bot) \mid n \in \omega, b \in Z\},$$

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

$$((f \gg f) \gg f)(\mathrm{fix})a \geq \bigvee \{f(((f \to f^*)a)^n \bot) \mid n \in \omega\}$$
$$= \bigvee \{f((f^* \cdot a \cdot f)^n \bot) \mid n \in \omega\}$$
$$= \bigvee \{a^n(f \bot) \mid n \in \omega\}$$
$$\geq \bigvee \{a^n \bot \mid n \in \omega\} = \mathrm{fix}(a) \qquad \square$$

We go one level higher and introduce abstraction transformations.

4.8. <u>Definition</u>.
i) Let $<f,f^*>, <g,g^*>$ 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 $(\phi, \psi, <f,f^*>, <g,g^*>)$ an *abstraction diagram* if

$$\psi \cdot f = g \cdot \phi,$$
$$f^* = \phi^L \cdot g^* \cdot \psi.$$

ii) Let $F, G \in \mathbb{F}(A_e{}^n, A_e)$. 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) $\gg : \to_e \Rightarrow \to_e$.

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

(10) $\psi \cdot f = g \cdot \phi$,

(11) $f^* = \phi^L \cdot g^* \cdot \psi$,

(12) $\psi' \cdot f' = g' \cdot \phi'$,

(13) $f'^* = \phi'^L \cdot g'^* \cdot \psi'$,

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

(14) $(\psi \to_e \psi') \cdot (f \gg f') = (g \gg g') \cdot (\phi \to_e \phi')$,

(15) $f \to f^* = (\phi \to_e \phi')^L \cdot (g \to g'^*) \cdot (\psi \to_e \psi')^L$.

For (14) we argue as follows. We have

$$((\psi \to_e \psi') \cdot (f \gg f'))(x)y = \psi'(\bigvee \{ f'(xz) \mid fz \le \psi^L y \}),$$
$$((g \gg g') \cdot (\phi \to_e \phi'))(x)y = \bigvee \{ g'(\phi'(x(\phi^L z))) \mid gz \le y \} = \psi'(\bigvee \{ f'(x(\phi^L z)) \mid gz \le y \}),$$

16

where the last equality holds because of (12) and the additivity of ϕ' (see 2.2), so it suffices to show

(16) $\{z \mid fz \leq \psi^L y\} = \{\phi^L z \mid gz \leq y\}$,

which is done in two steps. First assume $z' \in \{z \mid fz \leq \psi^L y\}$, i.e. $fz' \leq \psi^L y$, then $\psi(fz') \leq \psi(\psi^L y) \leq y$, so (by (10)) $g(\phi z') \leq y$ and $z' = \phi^L(\phi z') \in \{\phi^L z \mid gz \leq y\}$. For the other way round, assume $u \in \{\phi^L z \mid gz \leq y\}$, then $u = \phi^L z$ for some z with $gz \leq y$; then $\psi^L(gz) \leq \psi^L y$ and also $f \cdot \psi \leq \psi^L \cdot g$ (a consequence of (10) and the properties of embeddings), so $f(\psi z) \leq \psi^L y$, so $u = \phi^L z \in \{z \mid fz \leq \psi^L y\}$. So (16) is proved, and we conclude (14).

(15) is proved as follows: $(f \to f'^*)(z) = f^* \cdot z \cdot f = \phi'^L \cdot g'^* \cdot \psi' \cdot z \cdot \psi^L \cdot g \cdot \phi = ((\phi \to_e \phi')^L \cdot (g \to g'^*) \cdot (\psi \to_e \psi')^L)(z)$, using (13) and (10) for the second equality. $\square$

4.11. <u>Remark</u>. 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. <u>Counterexample</u>. There are diagrams (ϕ, ψ, f, g) and (ϕ', ψ', f', g') satisfying $f = \psi^L \cdot g \cdot \phi$, $f = \psi'^L \cdot g' \cdot \phi'$, but $(f \gg f') \neq (\psi \to_e \psi')^L \cdot (g \gg g') \cdot (\phi \to_e \phi')$.
<u>Proof</u>. We have

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

(b) $((\psi \to_e \psi')^L \cdot (g \gg g') \cdot (\phi \to_e \phi'))xy = \psi'^L(\vee \{g'(\phi'(x(\phi^L z)))) \mid gz \leq \psi y\})$

Now take

$f : \underline{3} \to \underline{2}$ with $f\bot = \bot$, $f* = fT = T$

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

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

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

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

(a) $= \vee \{u \mid fu \leq T\} = T \neq * = \vee \{\bot, *\} = \vee \{z \mid z \leq \psi T\} = $ (b). $\square$

17

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 $F,G \in \mathbb{F}(A_e, A_e)$ and $H : F \Rightarrow G$. The arrow chain of H is defined by $\mathrm{arch}(H) = \langle H^n \perp \rangle_n$, the fixpoint of H by $\mathrm{Fix}(H) = \lim(\mathrm{arch}(H))$.

ii) Now let $F,G \in \mathbb{F}(A_e{}^m * A_e, A_e)$, $H : F \Rightarrow G$, $f \in A_e{}^m$. Then $\mathrm{arch}(H_f) = \langle H_f{}^n \perp \rangle_n$ is the arrow chain of H in f, and $\mathrm{Fix}(H) = \lambda f.\mathrm{Fix}(H_f)$ 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) $\mathrm{Fix}(H)$ is an abstraction pair.

<u>Proof</u>. 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 with 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 $f_n = (H^n \perp)_1$, $f^*_n = (H^n \perp)_2$, $f = (\mathrm{Fix}(H))_1$, $f^* = (\mathrm{Fix}(H))_2$, $\phi_n = F^n(\phi)$, $\psi_n = F^n(\psi)$; 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: (a) f is $\perp$-unique, (b) f preserves compactness and (c) $f \cdot f^* = 1_{\mathrm{Fix}(G)}$.

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

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

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

so $y_k \in C(Y_k)$ and $\forall m \geq k(y_{m+1} = \psi_m(y_m))$, i.e. $<y_n>_n$ 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: $Fix(H) = <f, f^*>$ is an abstraction pair. □

4.15. <u>Lemma</u>.

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

ii) If $H : F \Rightarrow G$, then $Fix(H) : Fix(F) \Rightarrow Fix(G)$.

<u>Proof</u>. i) Easy, using that $(1_{dom(p)}, 1_{cod(p)}, p, p)$ 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 $p = <f, f^*> \in \mathbb{A}_a{}^m$, $q = <g, g^*> \in \mathbb{A}_a{}^m$, $\phi, \psi \in \mathbb{A}_e{}^m$. By (i) and lemma 4.14.(ii) we see that $Fix(H)p$ and $Fix(H)q$ are abstractions. So we only must show $(Fix(G)\psi) \cdot (Fix(H)p)_1 = (Fix(H)q)_1 \cdot (Fix(F)\phi)$ and $(Fix(H)p)_2 = (Fix(G)\psi)^L \cdot (Fix(H)q)_2 \cdot (Fix(F)\phi)$, i.e.

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

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

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

for all n. This is proved with induction: $n=0$ this is trivial, the induction step requires

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

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

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

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

furthermore we put

$abs[\rightarrow] = »$.

19

4.16. <u>Lemma</u>. If $\vdash \Phi : {}^n\Theta$ then $abs[\Phi]a : C[\Phi]dom(a) \Rightarrow A[\Phi]cod(a)$ for all $a \in$ TENV$_{abs}$.

<u>Proof</u>. Induction over the length of a derivation of $\vdash \Phi : {}^n\Theta$, strengthening the conclusion like in the proof of lemma 3.2.1 and using 4.10, 4.14, 4.15. $\square$

We now derive abs·C ≤ A, the main result on C, A and abs.

4.17. <u>Theorem</u>. If $\Gamma \vdash p : \Phi$ and $a \in$ TENV$_{abs}$, $b \in$ OENV$_C$, $b' \in$ OENV$_A$ satisfy

(Γ)   for all $(x : \sigma) \in \Gamma$, $abs[\sigma]a(bx) \leq b'x$,

then

(p,Φ) $abs[\Phi]a \cdot (C[p](dom(a))b) \leq A[p](cod(a))b'$.

<u>Proof</u>. Induction over the length of a derivation of $\Gamma \vdash p : \Phi$. We write d for dom(a), c for cod(a) and inspect the various cases. Observe that (t,τ) reads $abs[\tau]a(C[t]db) \leq A[t]cb'$.

$p = x$, $\Phi = \tau$: 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 with the induction hypothesis and (7).

$p = \lambda y{:}\sigma.t$, $\Phi = \sigma{\rightarrow}\tau$: 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 $(\lambda y : \sigma.t, \sigma{\rightarrow}\tau)$, i.e. (after some rewriting, using $\bigvee X \leq z \Leftrightarrow \forall x \in X\ x{\leq}z$)

$\forall e' \in A[\sigma]c\ (\ abs[\sigma]a(by) \leq e' \rightarrow abs[\tau]a(C[t]db) \leq A[t]c(b'[y{\rightarrow}e'])))$

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

This completes the development of abs and A: now

(6) $A[t]$ strict $\Rightarrow C[t]$ strict

holds, for we have (5) (theorem 4.17) and (7) (i.e. $abs[\tau](xy) \leq abs[\sigma{\rightarrow}\tau]x\ abs[\sigma]y$), 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:

$$\text{cond, pcond} : \delta\alpha.B \times \alpha \times \alpha \rightarrow \alpha$$
$$C[\text{cond}] = \text{cond}, C[\text{pcond}] = \text{pcond}$$
$$A[\text{cond}] = \text{abscond}, AC[\text{pcond}] = \text{abspcond}$$

where cond, pcond, abscond, abspcond are defined by

$$\text{cond}(T,x,y) = x \vee y$$
$$\text{cond}(\text{true},x,y) = x$$
$$\text{cond}(\text{false},x,y) = y$$
$$\text{cond}(\perp,x,y) = \perp$$

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

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

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 $((f\text{»}f)\text{»}f)\text{fix} = \text{fix}$. This enabled us to put $A(\text{fix}) = \lambda X \in \text{Obj}(A).\text{fix}_X$, 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 $((f\text{»}f)\text{»}f)\text{fix} \leq \text{fix}$, and this is enough to admit the definition $A(\text{fix}) = \lambda X \in \text{Obj}(A).\text{fix}_X$ while retaining the property $\text{abs} \cdot C \leq A$.

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 A of algebraic complete lattices. But now it is no longer possible to find a right

21

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

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

5.2. <u>Perspectives for future research</u>.

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 [B90]). The obvious question is: given some concrete interpretation, find an abstract interpretation and an abstraction mapping between the two.

ii) Generalisation (e.g. using adjunctions in the style of [N88]) 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.

REFERENCES.

[A85]   S. Abramsky, *Strictness analysis and polymorphic invariance*, in: H. Ganzinger, N.D. Jones (eds.) *Programs as Data Objects*, (Lecture Noter on Computer Science Vol. 217), Springer-Verlag, Berlin etc., 1985, 1 - 23

[AM75]   M.A. Arbib, E.G. Manes, *Arrows, Structures and Functors: the Categorical Imperative*, Academic Press, 1975

[B90]   H.P. Barendregt, *Lambda calculus with types*, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum, *Handbook of Logic in Computer Science*, Oxford University Press, to appear

[BKKS86]   H.P. Barendregt, J.R. Kennaway, J.W. Klop, M.R. Sleep, *Needed reduction and spine strategies for the lambda calculus*, Report CS-R8621, Centre for Mathematics and Computer Science, 1986

[BHA86]   G.L. Burn, C.L. Hankin, S. Abramsky, *Strictness analysis for higher order functions*, Science of Computer Programming 7 (1986) 249 - 278

[CC77]   P. Cousot, R. Cousot, *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* (1977) 238 - 252

[CC77]   P. Cousot, R. Cousot, *Systematic design of program analysis frameworks*, in: *Conference Record of the 6th ACM Symposium on Principles of Programming Languages* (1977) 238 - 252

[H87]   J. Hughes, *Backwards analysis of functional programs*, in: XXX

[M71]   S. MacLane, *Categories for the Working Mathematician*, Springer-Verlag, Berlin etc., 1971

[My80]   A. Mycroft, *The theory and practice of transforming call-by-need into call-by-value*, in: Proceedings International Symposium on Programming (Lecture Noter on Computer Science Vol. 83), Springer-Verlag, Berlin etc., 1980, 269 - 281

[My81]   A. Mycroft, *Abstract interpretation and optimising transformations for applicative programs*, Ph.D. Thesis, Edinburgh University, 1981

[My84]   A. Mycroft, *Polymorphic type schemes and recursive definition*, in: M. Paul and B. Robinet (eds.), *International Symposium on Programming*, LNCS 167, Springer-Verlag, 1984

[N84]   F. Nielson, *Abstract interpretation usnig domain theory*, Ph.D. Thesis, Edinburgh University, 1984

[N86]   F. Nielson, *Abstract interpretations of denotational definitions*, in: *Proceedings STACS 1986*, Springer-Verlag (Lecture Notes in Computer Science, Vol. 210), 1986

[N88]   F. Nielson, *Strictness analysis and denotational abstract interpretation*, Information and Computation 76 (1988) 29 - 92

[R88]   G.R. Renardel de Lavalette, *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, 1988

[S76]   D.S. Scott, *Data types as lattices*, SIAM Journal of Computing 5 (1976) 522 - 587

[S82]   D.S. Scott, *Domanis for denotational semantics*, in M. Nielson , E. Schmidt (eds.), *9th International Colloquium on Automata, Languages and Programming*, Springer-Verlag (Lecture Notes in Computer Science, Vol. 140), 1982, 577 - 613

[SP82]   M. Smyth, G. Plotkin, *The category-theoretic solution of recursive domain equations*, SIAM Journal of Computing 11 (1982) 761-783

Logic Group Preprint Series
Department of Philosophy
University of Utrecht
Heidelberglaan 2
3584 CS Utrecht
The Netherlands

1   C.P.J. Koymans, J.L.M. Vrancken, *Extending Process Algebra with the empty process*,
    September 1985
2   J.A. Bergstra, *A process creation mechanism in Process Algebra*, September 1985
3   J.A. Bergstra, *Put and get, primitives for synchronous unreliable message passing*,
    October 1985
4   A. Visser, *Evaluation, provably deductive equivalence in Heyting's arithmetic of
    substitution instances of propositional formulas*, November 1985
5   G.R. Renardel de Lavalette, *Interpolation in a fragment of intuitionistic propositional logic*,
    January 1986
6   C.P.J. Koymans, J.C. Mulder, *A modular approach to protocol verification using Process
    Algebra*, April 1986
7   D. van Dalen, F.J. de Vries, *Intuitionistic free abelian groups*, April 1986
8   F. Voorbraak, *A simplification of the completeness proofs for Guaspari and Solovay's R*,
    May 1986
9   H.B.M. Jonkers, C.P.J. Koymans & G.R. Renardel de Lavalette, *A semantic framework
    for the COLD-family of languages*, May 1986
10 G.R. Renardel de Lavalette, *Strictheidsanalyse*, May 1986
11 A. Visser, *Kunnen wij elke machine verslaan? Beschouwingen rondom Lucas' argument*,
    July 1986
12 E.C.W. Krabbe, *Naess's dichotomy of tenability and relevance*, June 1986
13 H. van Ditmarsch, *Abstractie in wiskunde, expertsystemen en argumentatie*, Augustus
    1986
14 A. Visser, *Peano's Smart Children, a provability logical study of systems with built-in
    consistency* , October 1986
15 G.R. Renardel de Lavalette, *Interpolation in natural fragments of intuitionistic
    propositional logic*, October 1986
16 J.A. Bergstra, *Module Algebra for relational specifications*, November 1986
17 F.P.J.M. Voorbraak, *Tensed Intuitionistic Logic*, January 1987
18 J.A. Bergstra, J. Tiuryn, *Process Algebra semantics for queues*, January 1987
19 F.J. de Vries, *A functional program for the fast Fourier transform*, March 1987
20 A. Visser, *A course in bimodal provability logic*, May 1987
21 F.P.J.M. Voorbraak, *The logic of actual obligation, an alternative approach to deontic
    logic*, May 1987
22 E.C.W. Krabbe, *Creative reasoning in formal discussion*, June 1987
23 F.J. de Vries, *A functional program for Gaussian elimination*, September 1987
24 G.R. Renardel de Lavalette, *Interpolation in fragments of intuitionistic propositional logic*,
    October 1987 (revised version of no. 15)
25 F.J. de Vries, *Applications of constructive logic to sheaf constructions in toposes*, October
    1987
26 F.P.J.M. Voorbraak, *Redeneren met onzekerheid in expertsystemen*, November 1987
27 P.H. Rodenburg, D.J. Hoekzema, *Specification of the fast Fourier transform algorithm as
    a term rewriting system*, December 1987

28 D. van Dalen, *The war of the frogs and the mice, or the crisis of the Mathematische Annalen,* December 1987

29 A. Visser, *Preliminary Notes on Interpretability Logic,* January 1988

30 D.J. Hoekzema, P.H. Rodenburg, *Gauß elimination as a term rewriting system,* January 1988

31 C. Smoryński, *Hilbert's Programme,* January 1988

32 G.R. Renardel de Lavalette, *Modularisation, Parameterisation, Interpolation,* January 1988

33 G.R. Renardel de Lavalette, *Strictness analysis for POLYREC, a language with polymorphic and recursive types,* March 1988

34 A. Visser, *A Descending Hierarchy of Reflection Principles,* April 1988

35 F.P.J.M. Voorbraak, *A computationally efficient approximation of Dempster-Shafer theory,* April 1988

36 C. Smoryński, *Arithmetic Analogues of McAloon's Unique Rosser Sentences,* April 1988

37 P.H. Rodenburg, F.J. van der Linden, *Manufacturing a cartesian closed category with exactly two objects,* May 1988

38 P.H. Rodenburg, J.L.M.Vrancken, *Parallel object-oriented term rewriting : The Booleans,* July 1988

39 D. de Jongh, L. Hendriks, G.R. Renardel de Lavalette, *Computations in fragments of intuitionistic propositional logic,* July 1988

40 A. Visser, *Interpretability Logic,* September 1988

41 M. Doorman, *The existence property in the presence of function symbols,* October 1988

42 F. Voorbraak, *On the justification of Dempster's rule of combination,* December 1988

43 A. Visser, *An inside view of EXP, or: The closed fragment of the provability logic of $I\Delta_0+\Omega_1$,* February 1989

44 D.H.J. de Jongh & A. Visser, *Explicit Fixed Points in Interpretability Logic,* March 1989

45 S. van Denneheuvel & G.R. Renardel de Lavalette, *Normalisation of database expressions involving calculations,* March 1989

46 M.F.J. Drossaers, *A Perceptron Network Theorem Prover for the Propositional Calculus,* July 1989

47 A. Visser, *The Formalization of Interpretability,* August 1989

48 J.L.M. Vrancken, *Parallel Object Oriented Term Rewriting : a first implementation in Pool2,* September 1989

49 G.R. Renardel de Lavalette, *Choice in applicative theories,* September 1989

50 C.P.J. Koymans & G.R. Renardel de Lavalette, *Inductive definitions in COLD-K,* September 1989

51 F. Voorbraak, *Conditionals, probability, and belief revision (preliminary version),* October 1989

52 A. Visser, *On the Σ_1^0-Conservativity of Σ_1^0-Completeness,* October 1989

53 G.R. Renardel de Lavalette, *Counterexamples in applicative theories with choice,* January 1990

54 D. van Dalen, *L.E.J. Brouwer. Wiskundige en Mysticus,* June 1990

55 F. Voorbraak, *The logic of objective knowledge and rational belief,* September 1990

56 J.L.M. Vrancken, *Reflections on Parallel and Functional Languages,* September 1990

57 A. Visser, *An inside view of EXP, or: The closed fragment of the provability logic of $I\Delta_0+\Omega_1$,* revised version with new appendices, October 1990

58 S. van Denneheuvel, K. Kwast, G.R. Renardel de Lavalette, E. Spaan, *Query optimization using rewrite rules,* October 1990

59 G.R. Renardel de Lavalette, *Strictness analysis via abstract interpretation for recursively defined types,* October 1990