Strictness Properties of Lazy Algebraic Datatypes

P. N. Benton
University of Cambridge*

Abstract

A new construction of a finite set of strictness properties for any lazy algebraic datatype is presented. The construction is based on the categorical view of the solutions to the recursive domain equations associated with such types as initial algebras. We then show how the initial algebra induction principle can be used to reason about the entailment relation on the chosen collection of properties. We examine the lattice of properties given by our construction for the type *nlist* of lazy lists of natural numbers and give proof rules which extend the conjunctive strictness logic of [2] to a language including the type *nlist*.

1 Introduction

This paper concerns the problem of extending an ideal-based strictness analysis for a PCF-like language (e.g. the abstract interpretation of [4] or the strictness logic of [2, 6]) to a language which includes lazy algebraic datatypes, such as lazy lists or trees.

We start by giving a brief overview of ideal-based strictness analysis, using the language of strictness logic[1]. A more comprehensive account can be found in [1].

At each type σ of our language Λ_T we define a propositional theory $\mathcal{L}_\sigma = (L_\sigma, \leq_\sigma)$ where L_σ is a set of propositions and $\leq_\sigma \subseteq L_\sigma \times L_\sigma$ is the (finitely axiomatised) entailment relation. Each proposition ϕ^σ is interpreted as a non-empty Scott-closed subset (i.e. an *ideal*) $[\![\phi^\sigma]\!]$ of the domain D_σ which interprets the type σ in the standard denotational semantics for Λ_T. For $d \in D_\sigma$, $\phi^\sigma \in \mathcal{L}_\sigma$ write $d \models \phi^\sigma$ for $d \in [\![\phi^\sigma]\!]$. We require soundness of $\mathcal{L}_\sigma$ with respect to its intended interpretation, so $\phi \leq \psi$ implies that $[\![\phi]\!] \subseteq [\![\psi]\!]$, and we can sometimes get the converse (completeness) too. For decidability, we require that the Lindenbaum algebra $\mathcal{LA}_\sigma$ of the theory $\mathcal{L}_\sigma$ be finite. We shall assume that $\mathcal{L}_\sigma$ contains at least the propositions $\mathbf{t}^\sigma$ (with $[\![\mathbf{t}^\sigma]\!] = D_\sigma$)

*Author's address: University of Cambridge, Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK. Email: `Nick.Benton@cl.cam.ac.uk`. Research supported by the Cambridge Philosophical Society and a SERC Fellowship.

[1]Most of the present work can, however, be easily reinterpreted in terms of a more traditional abstract interpretation.

and $\mathbf{f}^\sigma$ (with $[\![\mathbf{f}^\sigma]\!] = \{\perp_{D_\sigma}\}$), and that the logic contains conjunction (interpeted as intersection). We can also define a lattice A_σ to be the set $\{[\![\phi^\sigma]\!] \mid \phi^\sigma \in \mathcal{L}_\sigma\}$ ordered by inclusion. The map $abs_\sigma : D_\sigma \to A_\sigma$ is given by $abs_\sigma(d) = \bigcap\{I \in A_\sigma \mid d \in I\}$. Note that if $\mathcal{L}_\sigma$ is complete, then $\mathcal{LA}_\sigma$ and A_σ will be isomorphic, but this will not always be the case.

We then give a program logic for deducing judgements of the form $\Gamma \vdash t : \phi^\tau$ where t is a language term of type τ and $\Gamma = \{x_i^{\sigma_i} : \psi_i^{\sigma_i}\}$ is a finite set of assumptions. The program logic should be sound in the sense that if $\Gamma \vdash t : \phi^\tau$ is derivable then for any environment ρ for which $\rho \models \Gamma$ (in the obvious pointwise sense) we have $[\![t]\!]\rho \models \phi^\tau$.

In extending such an analysis to a language with an algebraic datatype a, we first have to pick a set L_a of propositions ϕ^a representing ideals $[\![\phi^a]\!]$ of the domain D_a. We then need inference rules for the relation $\leq_a$ and program logic rules for the constructors and destructors associated with the type a.

Deciding which ideals of D_a we wish to reason about is a rather difficult problem. The collection which we pick should have the following characteristics:

1. It should be finite.
2. It should not be too big. This is because a large collection of properties will lead to an impractically slow analysis system.
3. It should contain as many 'useful' (in terms of optimisation-enabling) properties as possible. This plainly pulls against the previous requirement.
4. It should also contain sufficient points to enable us to deduce useful properties of real programs. Even if, for example, we were only interested in the optimisations which can be performed as a result of simple strictness, an analysis which only made use of the two-point lattice at every type would be extremely weak.
5. It should be closed under intersection, so that each domain element has a 'best' abstraction.
6. There should be some procedure by which the lattice of properties is derived from the type declaration.
7. The compiler should be able automatically to calculate a representation of the lattice of properties from the type declaration.
8. The ordering on the representation should be sound with respect to the inclusion ordering on the interpretations of representatives, and as complete as is practical.
9. The compiler should be able to synthesize proof rules or abstract semantic equations for the constructors and destructors of the type a.

For particular algebraic types, various *ad hoc* solutions to the problem have been suggested, the best known of which is probably Wadler's four point abstract domain for lazy lists of elements of a flat domain [9]. So far, however, there has been no general construction which meets all the above criteria.

2 Recursive Domain Equations

Before we can describe our approach to the problem of constructing lattices of strictness properties for algebraic datatypes, we need to recall how domains for such datatypes are constructed in the standard semantics as the solutions to recursive domain equations. Full details may be found in, for example, [8, 1].

Write $\mathcal{D}om$ for the category of pointed ω-cpo's with continuous maps as morphisms, $\mathcal{D}om_S$ for the subcategory of $\mathcal{D}om$ with only strict maps as morphisms, and $\mathcal{D}om_E$ for the subcategory of $\mathcal{D}om_S$ with only embeddings as morphisms. Although, as we are working with a non-strict language, we shall ultimately want to think of the domains which we construct as living in $\mathcal{D}om$, it will turn out that the category in which they have the properties which we shall use is $\mathcal{D}om_S$. Thus we shall treat the basic domain constructions as functors on $\mathcal{D}om_S$.

Each of the type constructors $+, \times, \rightarrow$ corresponds to a locally continuous functor T from $(\mathcal{D}om_S{}^{\mathrm{op}})^m \times \mathcal{D}om_S{}^n$ to $\mathcal{D}om_S$ (in particular, $+$ is interpreted using the *separated* sum functor). Such a functor gives rise to an ω-continuous functor $T^E: \mathcal{D}om_E{}^{m+n} \rightarrow \mathcal{D}om_E$. By composing these functors, an arbitrary (not necessarily algebraic) recursive type definition defines an ω-continuous functor $F: \mathcal{D}om_E \rightarrow \mathcal{D}om_E$. The domain which interprets the recusive type is then the colimit $\underrightarrow{\lim}\Delta$ of the ω-diagram Δ in $\mathcal{D}om_E$, where

$$\Delta = 1 \xrightarrow{!} F(1) \xrightarrow{F(!)} F^2(1) \xrightarrow{F^2(!)} \cdots$$

We will write Fix_F for $\underrightarrow{\lim}\Delta$, which comes with an isomorphism $\eta_F: F(\mathsf{Fix}_F) \rightarrow \mathsf{Fix}_F$.

Recall that if $F: \mathcal{C} \rightarrow \mathcal{C}$ is an endofunctor on a category $\mathcal{C}$, then an *F-algebra* consists of a pair (A, α) where A is an object of $\mathcal{C}$ and $\alpha: FA \rightarrow A$. The collection of F-algebras are the objects of a category $F - \mathcal{A}lg$ in which a morphism from (A, α) to (B, β) is a morphism $f: A \rightarrow B$ in $\mathcal{C}$ such that

$$\begin{tikzcd} FA \arrow[r, "Ff"] \arrow[d, "\alpha"'] & FB \arrow[d, "\beta"] \\ A \arrow[r, "f"'] & B \end{tikzcd}$$

commutes. Such an f is called an *F-homomorphism*.

Now it is the case that for an arbitrary ω-continuous functor $F: \mathcal{D}om_E \rightarrow \mathcal{D}om_E$, the pair (Fix_F, η_F) is the initial F-algebra. In the case of an algebraic datatype definition (one in which the name of the type being defined does not appear within the scope of a function space constructor), the functor F arises as T^E, where $T: \mathcal{D}om_S \rightarrow \mathcal{D}om_S$ is a locally continuous functor. In this case, the pair $(\mathsf{Fix}_{T^E}, \eta_{T^E})$ is the initial T-algebra. Since in the algebraic case T^E is just the restriction of T, we shall henceforth drop the distinction between T and T^E whenever it seems convenient.

Homomorphisms from the initial T-algebra capture a kind of primitive recursion over lazy algebraic datatypes. In the case of lazy lists, this gives the familiar reduce or fold function.

3 The Construction

We were careful in the introduction to draw a distinction between A_σ (the lattice of properties) and $\mathcal{L}A_\sigma$ (the lattice of representatives for those properties). We shall start by giving a construction of A_a and then address the question of how this may be axiomatised in the theory $\mathcal{L}_a$.

A Scott-closed subset of a domain D is precisely the kernel $f^{-1}(\perp)$ of a continuous map $f: D \rightarrow \mathcal{O}$, where $\mathcal{O}$ is the two-point domain. An ideal is the kernel of a *strict* map into $\mathcal{O}$. We can therefore rephrase our problem as being that of finding a good set of strict maps from D_a into $\mathcal{O}$. Our construction will define such a collection of maps using homomorphisms from an initial algebra.

The first step is to generalise the idea of going from the domain of lists of natural numbers to the domain of lists of strictness properties of natural numbers. This latter domain is, of course, infinite and hence unsuited to our purposes (it contains points which represent properties such as 'has a $\perp$ in every prime-numbered position'). We shall then cut this down to a small finite set of properties.

For concreteness, let us assume that we are given the declaration of the type a in sum-of-products form:

$$a \;=\; \mathsf{C}_1 \text{ of } \delta_{1,0} \times \ldots \times \delta_{1,m_1-1} \mid \ldots \mid \mathsf{C}_n \text{ of } \delta_{n,0} \times \ldots \times \delta_{n,m_n-1};$$

where the type expressions δ are defined by the grammar

$$\delta ::= \kappa \mid a$$

$$\kappa ::= \iota \mid (\kappa \rightarrow \kappa) \mid (\kappa \times \kappa)$$

and the types of Λ_T are given by

$$\sigma ::= \iota \mid (\sigma \rightarrow \sigma) \mid (\sigma \times \sigma) \mid a$$

(Note that whilst m_i may be 0, n is always at least 1.)

The functor $T: \mathcal{D}om_S \rightarrow \mathcal{D}om_S$ associated with such a type declaration is then

$$T(X) = \sum_{i=1}^{n} \prod_{j=0}^{m_i-1} D_{i,j}(X)$$

where

$$D_{i,j}(X) = \begin{cases} D_\kappa & \text{if } \delta_{i,j} = \kappa \\ X & \text{if } \delta_{i,j} = a \end{cases}$$

Now let $\widehat{T}: \mathcal{D}om_S \rightarrow \mathcal{D}om_S$ be given by

$$\widehat{T}(X) = \sum_{i=1}^{n} \prod_{j=0}^{m_i-1} \widehat{D}_{i,j}(X)$$

where

$$\widehat{D}_{i,j}(X) = \begin{cases} A_\kappa & \text{if } \delta_{i,j} = \kappa \\ X & \text{if } \delta_{i,j} = a \end{cases}$$

Intuitively, $\mathsf{Fix}_{\widehat{T}}$ is then the appropriate generalisation of the idea of the domain of lists of strictness properties mentioned above. We now wish to give a formal definition of the obvious abstraction map from Fix_T to $\mathsf{Fix}_{\widehat{T}}$. Firstly, note that for any domain X, there is a strict map $\mu_X : T(X) \to \widehat{T}(X)$ given by

$$\mu_X = \sum_{i=1}^{n} \prod_{j=0}^{m_i - 1} f_{i,j}(X)$$

where $f_{i,j}(X) : D_{i,j}(X) \to \widehat{D}_{i,j}(X)$ is the strict map given by

$$f_{i,j}(X) = \begin{cases} abs_\kappa & \text{if } \delta_{i,j} = \kappa \\ id_X & \text{if } \delta_{i,j} = a \end{cases}$$

$\eta_{\widehat{T}} \circ \mu_{\mathsf{Fix}_{\widehat{T}}}$ makes $\mathsf{Fix}_{\widehat{T}}$ into a T-algebra, so by initiality there is a unique $h : \mathsf{Fix}_T \to \mathsf{Fix}_{\widehat{T}}$ such that $h \circ \eta_T = \eta_{\widehat{T}} \circ \mu_{\mathsf{Fix}_{\widehat{T}}} \circ T(h)$ and this is the map we want.

The second step in the construction uses the fact that $(\mathsf{Fix}_{\widehat{T}}, \eta_{\widehat{T}})$ is the initial $\widehat{T}$-algebra. This means that, given any strict map $g : \widehat{T}(\mathcal{O}) \to \mathcal{O}$, there is a unique strict map $g^* : \mathsf{Fix}_{\widehat{T}} \to \mathcal{O}$ such that

$$\begin{tikzcd} \widehat{T}(\mathsf{Fix}_{\widehat{T}}) \arrow[r, "\widehat{T}(g^*)"] \arrow[d, "\eta_{\widehat{T}}"'] & \widehat{T}(\mathcal{O}) \arrow[d, "g"] \\ \mathsf{Fix}_{\widehat{T}} \arrow[r, "g^*"'] & \mathcal{O} \end{tikzcd}$$

commutes. Because $\widehat{T}(\mathcal{O})$ is constructed from finite sums and products of finite domains, it will itself always be a finite domain. Thus there will only be a finite number of maps $g : \widehat{T}(\mathcal{O}) \to \mathcal{O}$. Each one of these gives rise to an ideal of Fix_T, viz.

$$K_g = (g^* \circ h)^{-1}(\bot)$$

and it is the collection of all these K_g which we propose as a good set of basic strictness properties of the type a. We shall also find it helpful to define $J_g \subseteq \mathsf{Fix}_{\widehat{T}}$ to be $(g^*)^{-1}(\bot)$.

Let us see how this works out in the familiar case of lists of natural numbers. A_ι is the two-point domain, which we will write as $\mathbf{2}$. This is, of course, isomorphic to $\mathcal{O}$, but this is just coincidental. There are twelve strict monotone maps g from $1 + \mathbf{2} \times \mathcal{O}$ into $\mathcal{O}$, each of which can be described by giving a pair, the first component of which is the image of the point $inl(*)$ and the second component of which is the image of the point $inr(x, y)$ for $x \in \mathbf{2}$, $y \in \mathcal{O}$. A little calculation then gives the informal interpretation of each K_g as shown in Figure 1 (we write K_n for K_{g_n}).

This collection of properties includes Wadler's four points: his $\bot$, ∞, $\bot\in$ and $\top\in$ are our K_{12}, K_6, K_8 and K_1 respectively. Burn's A^{hs} domain [3] consists of our K_{12}, K_4 and K_1. Ernoult and Mycroft's set of 'uniform ideals' [5] consists of our K_1, K_2, K_6,K_8,K_9,K_{10} and K_{12}. Note also that our twelve maps only give rise to ten distinct properties, as K_1, K_5 and K_7 are all the whole of the domain D_{nlist}.

n	$g_n(inl(*))$	$g_n(inr(x,y))$	K_n
1	$\bot$	$\bot$	All lists.
2	$\top$	$\bot$	All non-empty lists.
3	$\bot$	x	The empty list. Lists with $\bot$ as the first element.
4	$\top$	x	Non-empty lists with $\bot$ as the first element.
5	$\bot$	y	All lists.
6	$\top$	y	All infinite and partial lists.
7	$\bot$	$x \sqcap y$	All lists.
8	$\top$	$x \sqcap y$	Infinite and partial lists. Lists containing a $\bot$ elt.
9	$\bot$	$x \sqcup y$	The empty list.Lists whose elements are all $\bot$.
10	$\top$	$x \sqcup y$	Partial and infinite lists all of whose elt.s are $\bot$.
11	$\bot$	$\top$	The empty list.
12	$\top$	$\top$	Just the completely undefined list $\bot$.

Figure 1: Basic Strictness Properties of *nlist*.

These ten properties are all intuitively compelling: this seems to be the 'right' construction. These ten points alone, however, do not quite satisfy all the criteria we laid down earlier for a good collection of properties. This is because the collection is not closed under intersection. The intersection of K_8 and K_9 is the set of non-empty lists all of whose elements are $\bot$, and this is not one of our properties. This is not a serious problem—we just have to add intersections in explicitly[2].

Another of our criteria for a good collection of properties was that we should be able to calculate an entailment relation on a set of representations of the properties which was sound and fairly complete with respect to the real inclusion ordering on the properties. Defining L_a, a set of propositions which represent the ideals produced by our construction is slightly messy, but not difficult. We represent each K_g by giving a syntactic representation of g. This is done by giving the maximal elements of the kernel of g. More formally, we add the following to the formation rules given in [2]:

Define $P_{i,j}$ to be L_κ if $\delta_{i,j} = \kappa$ and $\mathcal{O}$ if $\delta_{i,j} = a$. Now define the set Q_i by

$$\frac{\phi_0 \in P_{i,0} \quad \ldots \quad \phi_{m_i-1} \in P_{i,m_i-1}}{\phi_0 \times \cdots \times \phi_{m_i-1} \in Q_i}$$

and then define the actual propositions in L_a by

$$\frac{S_1 \subseteq^{\text{fin}} Q_1 \quad \ldots \quad S_n \subseteq^{\text{fin}} Q_n}{S_1 + \cdots + S_n \in L_a}$$

This means that a typical element of L_a will look like

$$\phi^a = \bigwedge_{k=0}^{l-1} \sum_{i=1}^{n} S_i^k$$

[2] We might also want to add unions, if we were trying to extend a disjunctive strictness analysis [1] to algebraic types.

For example, a proposition representing K_1 is $\{1\} + \{\mathbf{t}^{\iota} \times \top\}$, whilst one representing K_8 is $\{\} + \{\mathbf{f}^{\iota} \times \top, \mathbf{t}^{\iota} \times \bot\}$.

The problem of defining the entailment relation on this set of propositions is more interesting. We approach it by considering reasoning principles which allow us to deduce that one property is contained within another. Note that we already have one simple principle for deducing inclusions between properties, namely that if $g \sqsubseteq g'$ then $g^* \sqsubseteq g'^*$ and hence $K_{g'} \subseteq K_g$. This is not, however, sufficient to deduce all the inclusions which we should like. In the case of *nlist*, for example, we want to be able to deduce that $K_1 \subseteq K_5$, but it is not the case that $g_5 \sqsubseteq g_1$.

Fortunately, initiality offers a solution to this problem too. We can make use of a general induction principle for initial T-algebras which is due to Lehmann, Smyth and Plotkin [7].

Proposition 3.1 (Initial T-Algebra Induction Principle) *If $T : \mathcal{C} \to \mathcal{C}$ is a functor, (A, α) is the initial T-algebra and $m : B \to A$ is a mono in $\mathcal{C}$ which extends to a T-homomorphism $(B, \beta) \to (A, \alpha)$, then m is an isomorphism.* □

Corollary 3.2 *If $T : \mathcal{D}om_S \to \mathcal{D}om_S$ is a functor, (D, α) is the initial T-algebra and P is a subdomain of D with inclusion map $i : P \to D$ then the following induction principle is valid:*

$$\frac{\forall x \in TP.(\alpha \circ Ti)(x) \in P}{\forall y \in D. y \in P}$$

□

And using this, we can get a simple condition on the maps g which is sufficient for J_g to be the whole of $\mathsf{Fix}_{\widehat{T}}$ and hence for K_g to be the whole of Fix_T.

Remark 3.3 A more categorical definition of J_g would have been by the pullback square

$$\begin{tikzcd}
J_g \arrow[r, "j"] \arrow[d, "!"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathsf{Fix}_{\widehat{T}} \arrow[d, "g^*"] \\
1 \arrow[r, "!"'] & \mathcal{O}
\end{tikzcd}$$

where j is the inclusion map. Note that J_g is a subdomain of $\mathsf{Fix}_{\widehat{T}}$, and not just a subset.

Proposition 3.4 *If the following diagram commutes:*

$$\begin{tikzcd}
\widehat{T}(1) \arrow[r, "\widehat{T}(!)"] \arrow[d, "!"'] & \widehat{T}(\mathcal{O}) \arrow[d, "g"] \\
1 \arrow[r, "!"'] & \mathcal{O}
\end{tikzcd}$$

then J_g is the whole of $\mathsf{Fix}_{\widehat{T}}$.

Proof. Consider the following:

$$\begin{tikzcd}
\widehat{T}(J_g) \arrow[r, "\widehat{T}(j)"] \arrow[d, "\widehat{T}(!)"'] & \widehat{T}(\mathsf{Fix}_{\widehat{T}}) \arrow[r, "\eta_{\widehat{T}}"] \arrow[d, "\widehat{T}(g^*)"] & \mathsf{Fix}_{\widehat{T}} \arrow[d, "g^*"] \\
\widehat{T}(1) \arrow[r, "\widehat{T}(!)"] \arrow[dr, "!"'] & \widehat{T}(\mathcal{O}) \arrow[r, "g"] & \mathcal{O} \\
& 1 \arrow[ur, "!"'] &
\end{tikzcd}$$

The square on the left commutes because it is $\widehat{T}$ applied to the pullback square in the remark above. The square on the right commutes because it is the definition of g^*, and the large triangle on the bottom commutes by assumption. Hence the outside path commutes, and since the composite $! \circ \widehat{T}(!) : \widehat{T}(J_g) \to 1$ on the left must be equal to $! : \widehat{T}(J_g) \to 1$, we have that

$$\forall x \in \widehat{T}(J_g).\, (\eta_{\widehat{T}} \circ \widehat{T}j)(x) \in J_g$$

so that by Corollary 3.2 we are done. □

Checking the condition of Proposition 3.4 is a simple, finite computation which could be incorporated into a mechanizable formal system for reasoning about strictness properties. Let us see how it works out in the case of *nlist*. In this case, the condition on g is that the following pair of equations hold:

$$g(inl(*)) = \bot$$

$$\forall x \in \mathbf{2}.\, g(inr(x, \bot)) = \bot$$

and by monotonicity, the second of these reduces to checking $g(inr(\top, \bot)) = \bot$. These two conditions pick out g_1, g_5 and g_7, which are precisely those g_n for which K_n is the whole of the domain.

But we need more than a rule for determining when some K_g is the whole of Fix_T. We also want a rule for deducing more general inclusions between intersections of the K_g. In the case of *nlist*, for example, we should like some way of deducing that $K_{11} \cap K_{10} \subseteq K_{12}$ and that $K_8 \cap K_3 \subseteq K_4$. The initial algebra induction principle can give us this too.

If A and B are ideals of a domain D then the set

$$A \Rightarrow B \stackrel{\text{def}}{=} \{d \in D \mid d \in A \Rightarrow d \in B\}$$

is a subdomain of D, since it clearly contains $\bot_D$ and if $\langle d_n \rangle$ is a chain in $A \Rightarrow B$ then if $\bigsqcup d_n \in A$ we must have $d_n \in A$ for all n by down-closure of A. Hence $d_n \in B$ for all n and so $\bigsqcup d_n \in B$ as B is closed under sups of chains. Obviously, if $A \Rightarrow B$ is the whole of D then $A \subseteq B$.

Since the intersection of a set of ideals is an ideal, this means by Corollary 3.2 that to show that $J_{m_1} \cap \cdots \cap J_{m_k} \subseteq J_n$ (and hence that $K_{m_1} \cap \cdots \cap K_{m_k} \subseteq K_n$), it suffices to show

$$\forall l \in \widehat{T}(J_{m_1} \cap \cdots \cap J_{m_k} \Rightarrow J_n).\ (\eta_{\widehat{T}} \circ \widehat{T}i)(l) \in J_{m_1} \cap \cdots \cap J_{m_k} \Rightarrow J_n$$

where $i : (J_{m_1} \cap \cdots \cap J_{m_k} \Rightarrow J_n) \to \mathsf{Fix}_{\widehat{T}}$ is the inclusion map. I do not yet have a re-statement of this condition in such a pleasant form as the condition of Proposition 3.4, but it is relatively easy to unwind in the special case of *nlist* to obtain

Proposition 3.5 *If the following two conditions hold*

1. *If* $\forall i.\, g_{m_i}(inl(*)) = \bot$ *then* $g_n(inl(*)) = \bot$
2. *For all* $b_1, \ldots, b_k, b \in \mathcal{O}$ *such that if* $\forall i.\, b_i = \bot$ *then* $b = \bot$ *we have that* $\forall x \in \mathbf{2}$ *if* $\forall i.\, g_{m_i}(inr(x, b_i)) = \bot$ *then* $g_n(inr(x, b)) = \bot$

then $K_{m_1} \cap \cdots \cap K_{m_k} \subseteq K_n$. □

Interestingly, this does not subsume Proposition 3.4, since Proposition 3.5 cannot be used to deduce $K_1 \subseteq K_7$. The conditions of Proposition 3.5 are finitely checkable, and it can be seen that this will remain true of the corresponding conditions for any algebraic type.

Using Propositions 3.4 and 3.5, we discover that only two more points need to be added the properties shown in Figure 1 to obtain a set which is closed under intersection. These are $K_4 \cap K_6$, with the informal interpretation of 'partial and infinite lists with $\bot$ as the first element', and $K_4 \cap K_9$ which corresponds to 'non-empty lists all of whose elements are $\bot$'. For *nlist*, we thus get the lattice of properties shown in Figure 2. This figure was deduced with the aid of a short computer program to check the conditions of Proposition 3.5. Burn's A^{rp} abstract domain for lists [3] contains our K_1, K_4, K_6, $K_4 \cap K_6$, K_8 and K_{12}.

Although the initial algebra induction principle is undoubtedly the correct way to reason about inclusions between properties, there is still a problem. We need to be able to express this reasoning as a set of syntactic proof rules for deducing entailments between propositions. At present I do not have a satisfactory way of doing this. The difficulty is that a generalised form of Proposition 3.5 is inherently second-order as it involves quantification over properties (the '$\forall x \in \mathbf{2}$' in the second clause). Thus a naive proof rule would involve quantification over propositions. This is unpleasant for a couple of reasons. Firstly, the theories associated with non-algebraic types are all first-order. Secondly, there will in general be an infinite number of propositions to quantify over. Although this problem is in principle avoidable (as the Lindenbaum algebras are all finite), it leads to a very complicated rule and means that one has to calculate explicitly the whole of $\mathcal{LA}_\sigma$ to be able to reason about entailments at any algebraic type which involves σ. This is completely impractical for any non-trivial σ. I believe that it is possible to reduce the quantification to a manageable finite set of propositions, but have not yet succeeded in doing so. Proof rules which capture the content of Proposition 3.4 and the naive principle $g \sqsubseteq g' \Rightarrow K_{g'} \subseteq K_g$ can, however, be given fairly easily in terms of our representation.

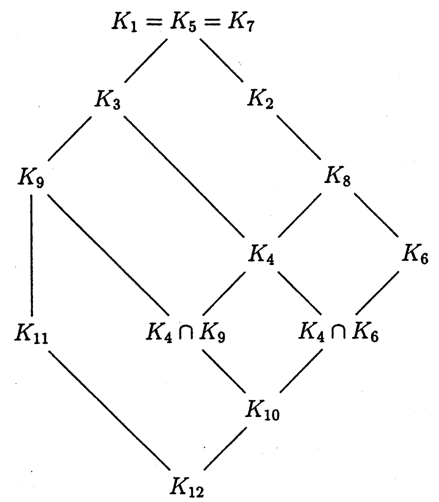

Figure 2: The Lattice of Strictness Properties of *nlist*

The final requirement of a collection of properties was that the compiler be able to synthesize program logic rules for the constructors and destructors of the algebraic type. This is fairly straightforward in terms of syntax of propositions which was given earlier, though the generality of the rules makes them look somewhat intimidating at first sight. For $p = \psi_0 \times \cdots \times \psi_{m_i-1} \in Q_i$ and $\phi \in L_a$, define $p[\phi/\bot] \in L_{\delta_{i,0}} \times \cdots \times L_{\delta_{i,m_i-1}}$ to be $(\psi'_0, \ldots, \psi'_{m_i-1})$ where

$$\psi'_j = \begin{cases} \psi_j & \text{if } \delta_{i,j} = \kappa \\ \phi & \text{if } \delta_{i,j} = a \text{ and } \psi_j = \bot \\ \mathbf{t}^a & \text{otherwise} \end{cases}$$

The program logic proof rule for constructors is then

$$\frac{\exists p \in S_i.\forall j.\quad \Gamma \vdash t_j : \pi_j(p[\sum_{i=1}^{n} S_i/\bot])}{\Gamma \vdash \mathsf{C}_i(t_0, \ldots, t_{m_i-1}) : \sum_{i=1}^{n} S_i}$$

The rule for destructors is

$$\frac{\Gamma \vdash t : \bigwedge_{k=0}^{l-1} \sum_{i=1}^{n} S_i^k \qquad \begin{array}{l} \forall i.\forall \langle p_0, \ldots, p_{l-1}\rangle \in \prod_{k=0}^{l-1} S_i^k. \\ \Gamma, x_{i,0} : \bigwedge_{k=0}^{l-1} \phi_0^k, \ldots, x_{i,m_i-1} : \bigwedge_{k=0}^{l-1} \phi_{m_i-1}^k \vdash u_i : \psi \end{array}}{\begin{array}{l} \Gamma \vdash \mathsf{case}\ t\ \mathsf{of}\ \ \mathsf{C}_1(x_{1,0}, \ldots, x_{1,m_1-1}) \Rightarrow u_1 \quad : \psi \\ \qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad \mathsf{C}_n(x_{n,0}, \ldots, x_{n,m_n-1}) \Rightarrow u_n \end{array}}$$

$$\Gamma \vdash \mathsf{Nil} : K_{11} \qquad \frac{\Gamma \vdash u : \mathbf{f} \qquad g(\bot, \top) = \bot}{\Gamma \vdash \mathsf{Cons}(u, l) : K_g}\,[\text{Cons1}]$$

$$\frac{\Gamma, x : \mathbf{t}, xs : K_8 \vdash v : \phi \qquad \Gamma, x : \mathbf{f}, xs : K_1 \vdash v : \phi \qquad \Gamma \vdash l : K_8}{\Gamma \vdash \mathsf{nlistcase}\ l\ \mathsf{of\ Nil} \Rightarrow u \mid \mathsf{Cons}(x, xs) \Rightarrow v\ : \phi}\,[\text{nlistcase8}]$$

$$\frac{\Gamma, x : \mathbf{f}, xs : K_6 \vdash v : \phi \qquad \Gamma \vdash l : K_4 \cap K_6}{\Gamma \vdash \mathsf{nlistcase}\ l\ \mathsf{of\ Nil} \Rightarrow u \mid \mathsf{Cons}(x, xs) \Rightarrow v\ : \phi}\,[\text{nlistcase46}]$$

Figure 3: Sample Proof Rules for *nlist*

where ϕ_j^k is an abbreviation for

$$\pi_j(p_k[\sum_{i=1}^{n} S_i^k/\bot])$$

Although these rules may look complicated, in the case of *nlist* they specialise to give a small set of very simple and natural proof rules. Some examples of these are shown in Figure 3, in which an attempt to improve readability has been made by replacing syntactic propositions with the names of the properties which they represent.

4 Conclusions and Further Work

We have presented a construction of a finite lattice of strictness properties (ideals) of any lazy algebraic datatype. This improves on previous work in that it is a general mathematical construction, rather than an *ad hoc* collection of useful-looking points for one particular type. For the particular case of lazy lists of elements of a flat domain our construction gives a set which is remarkably close to being the union of all the sets of properties which have been suggested in the literature. In this case, one's intuition should be that we are choosing as basic properties the kernels of all the evaluators (in the sense of Burn [3]) which can be written in terms of the reduce function. We also defined a uniform syntactic representation for these properties.

We then showed how the initial algebra induction principle could be used to reason about the inclusion ordering on (intersections of) our properties, though we do not yet have satisfactory proof rules which capture this reasoning.

We then gave general program logic proof rules which allow properties to be assigned to terms, and showed how these specialised to give derived rules for the case of lazy lists.

There is considerable scope for further work on this construction. The most immediate problem is to formulate good proof rules for the theory $\mathcal{L}_a$. Until this has been done, we have not completely succeeded in extending strictness logic to arbitrary algebraic types, although for any fixed algebraic type we do now at least have a framework for constructing the appropriate theories by hand calculation.

A natural extension of the current work is to investigate a disjunctive logic of strictness properties for algebraic types. Whilst this would simplify some aspects of the logic (for example, the sets S_i of propositions could be replaced by a single proposition), disjunctive strictness analysis introduces a number of further complications. See [1] for further information. We should also look at how well the construction works for more complicated algebraic types. Finally, we have not addressed the question of implementations – it remains to be seen whether a practical analysis system can be based on these ideas.

References

[1] P. N. Benton. *Strictness Analysis of Lazy Functional Programs.* PhD thesis, Computer Laboratory, University of Cambridge, December 1992.

[2] P. N. Benton. Strictness logic and polymorphic invariance. In A. Nerode and M. Taitslin, editors, *Proceedings of the Second International Symposium on Logical Foundations of Computer Science, Tver, Russia*, volume 620 of *Lecture Notes in Computer Science*, pages 33–44. Springer-Verlag, July 1992.

[3] G. L. Burn. *Lazy Functional Languages: Abstract Interpretation and Compilation.* Research Monographs in Parallel and Distributed Computing. MIT Press, Cambridge, Mass., 1991.

[4] G. L. Burn, C. L. Hankin, and S. Abramsky. The theory and practice of strictness analysis for higher-order functions. *Science of Computer Programming*, 7:249–278, 1986.

[5] C. Ernoult and A. Mycroft. Uniform ideals and strictness analysis. In *Proceedings of ICALP 91*. Springer-Verlag, 1991.

[6] T. P. Jensen. Strictness analysis in logical form. In *Proceedings of the 1991 Conference on Functional Programming Languages and Computer Architecture*, 1991.

[7] D. Lehmann and M. Smyth. Algebraic specification of data types: A synthetic approach. *Math. Systems Theory*, 14, 1981.

[8] M. B. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11, 1982.

[9] P. Wadler. Strictness analysis on non-flat domains (by abstract interpretation over finite domains). In S. Abramsky and C. L. Hankin, editors, *Abstract Interpretation of Declarative Languages*, chapter 12, pages 266–275. Ellis Horwood Ltd., 1987.