Uniform ideals and strictness analysis

Christine Ernoult[1]
Alcatel Alsthom Recherche
Route de Nozay
91460 Marcoussis
France
E-mail: ernoult@aar.alcatel-alsthom.fr

Alan Mycroft
Computer Laboratory, Cambridge University
New Museums Site, Pembroke Street
Cambridge CB2 3QG
United Kingdom
E-mail: am@cl.cam.ac.uk

Abstract

We propose a notion of *uniform ideal* (certain Scott-closed sets) to characterise strictness properties. This enables us to explain why Hughes' and Wadler's H projection for lazy list strictness analysis is not in general expressible as an abstract interpretation property of the standard semantics. We give circumstances when it is so expressible. Doing so casts light on Burn's H_B projection and his question of its relationship to H.

Uniform ideals are a generalisation of the sets of values corresponding to types in (simple) polymorphic type systems. Wadler's doubly-lifted abstract domain constructor for lazy lists can be seen as a special case which only uses certain uniform ideals.

The confluence of strictness and type theory furthers Kuo and Mishra's notion of "strictness types".

Summary of results

- We characterise strictness properties as *uniform ideals*.
- This enables us to give abstract interpretation properties to show that a function on $list(t_1+t_2)$ is H-strict (Wadler and Hughes).
- We show that the property of a function on $list(int)$ of being H-strict cannot be expressed as an abstract interpretation of the standard interpretation under weak conditions including uniform treatment of integer constants and compositionality.[2]
- Burn's H_B-strictness is seen as the best possible analogue to H-strictness subject to these conditions, but it is a non-uniform property.

1 Introduction

Strictness analysis was originated by Mycroft [11] for the first-order case over flat domains, using a formalism based on abstraction and concretisation functions.

Temporarily, suppose that D is a flat cpo. Let 2 stand for the the set $\{0,1\}$ ordered by $0<1$. Recall that $f : D^n \to D$ is *strict* in its k-th argument if $(\forall \overline{x} \in D^n) f(x_1,\ldots,x_{k-1},\perp,x_{k+1},\ldots,x_n) = \perp$. Mycroft developed a strictness theory for first order functions on flat domains which gave a standard interpretation of a program user-defined function symbol (say `f`) as a function f as above and also non-standard interpretations $f^\sharp$ and $f^\flat : 2^n \to 2$. Such $f^\sharp$ satisfy a correctness property with respect to f along the lines of $f^\sharp(1,\ldots,1,0,1,\ldots,1) = 0 \Rightarrow f$ is strict in its k-th argument.

[1]This author's contribution was supported by an INRIA fellowship and subsequent funding.

[2]This condition excludes treatments like Hunt's involving abstractions based other than on binary relations between the standard domain model and abstract elements.

This property is respected by composition and fixpoint extraction and so lifts from base functions to user-defined functions.

Note the modality of $f^\sharp$: if $f^\sharp$ states (by yielding 0) that f returns $\perp$ then f MUST do so. On the other hand if $f^\sharp$ yields 1 then nothing is said about f (merely that it MAY return a (non-$\perp$) value. This safety criterion is the best we can do for computability reasons. $f^\flat$ had the opposite modality: if $f^\flat$ yields 0 then f MAY return $\perp$ (*i.e.* nothing is said) but if it yields 1 then f MUST return a non-$\perp$ value. The $f^\flat$ modality corresponds more to open set properties and we discuss it no more.

We can generalise the $f^\sharp$ notion by introducing ideals (here Scott-closed sets) by observing that f is strict in its k-th argument iff $f(D \times \ldots \times D \times \{\perp\} \times D \times \ldots \times D) \subseteq \{\perp\}$. Note that $D \times \ldots \times D \times \{\perp\} \times D \times \ldots \times D$ is an ideal of D^n and $\{\perp\}$ an ideal of D. Hence we refer to classical strictness analysis as *ideal-based* to distinguish it from Wadler and Hughes' *projection-based* analysis introduced below.

Now, any function between domains determines and is determined by its behaviour on *all* ideals. The point is that strictness properties are certain special ideals — *e.g.* the uniform ideals we introduce here. (Commonly in abstract interpretation we obtain computable properties only by treating all integers identically.) Now, the behaviour of a function on just the uniform ideals (expressed in terms of properties of the form $f(I) \subseteq J$) does not in general determine f.

It is worth viewing the argument of the last paragraph in terms of continuous functions. As the argument to such a function increases (in the Scott sense, $\sqsubseteq$) then so does the result. So we can see strictness properties of functions as a statement that when the input is at most defined as δ then the output is at most ϵ defined — *i.e.* $f(I) \subseteq J$ where I, J are ideals. This naturally translates to abstract strictness properties — a function satisfies a given abstract strictness property if it maps abstract strictness properties of arguments and results in the desired manner.

Following Burn, we define a *strictness property* to be an ideal (later uniform ideal) of a given domain. Note that the set of functions which map a family of ideals into another family of ideals is itself an ideal of the function space, *i.e.*

$$\{f \in D \to D \mid (\forall k)(f(I_k) \subseteq J_k)\}$$

is an ideal of $D \to D$.

Burn, Hankin and Abramsky [4] showed that the *Hoare* (or *relational*) power-domain[3] (also used in Nielson's thesis [12]) could be used to generate a theory of strictness analysis for the *simply typed* λ-calculus. Abramsky [1] showed that this framework could also encompass Hindley-Milner style polymorphism. Around the same time Hudak and Young [6] gave a definition of *strictness pairs* which enabled them to analyse the *untyped* λ-calculus. This work was less semantically based than Burn, Hankin and Abramsky's and in retrospect their use of power-sets of variable names in their 'strictness pair' domain was both unnecessary and over-syntactic.

Wadler and Hughes [15] suggested using projections to characterise strictness properties of first-order functions operating on lazy lists as solutions of equations like

$$\beta \circ f \circ \alpha = \beta \circ f$$

where α and β are projections. This is *projection-based* strictness analysis. However, there is some mismatch between such properties and intuitive notions of strictness. On one hand, considering the special case $f = f \circ (\lambda x.\perp)$, we observe that this is satified precisely if f is a constant function, in particular $f(x) = x = \perp \to \perp, 42$ is not a solution. This is unsatisfactory in that the ultimate justification of strictness analysis is to argue that some term *may as well* be evaluated early and this is unaffected by replacement of a function by a less defined function. On the other hand, such equations cannot express even Mycroft's simple notion of strictness on flat domains without introducing artificial elements by 'lifting'.

Recently, Hunt [7] has developed a new theory of abstract interpretation using *PER*s (partial (here: relaxing reflexivity) equivalence relations) which seems to be able both to characterise the H-strictness property and to express ideal-based properties without lifting. This theory appears very

[3] Already a view of (forward) strictness properties as ideals emerges.

promising but represents a radically new approach to abstract interpretation in that the abstraction relation is not between the standard and abstract values but between *pairs* of standard values and abstract values. An alternative view may be to regard his work as a conventional abstract interpretation between a non-standard semantics and the abstract semantics. The non-standard semantics would be like a PER model of the λ-calculus whose extra intensional detail somehow helps over the conventional domain model. It would seem to have analogies with the *collecting* semantics of Mycroft and Nielson or the *instrumented* semantics used by some authors whereby intensional detail is incorporated into the denotations for (*e.g.*) abstract complexity analysis. The main worry is that PERs are very general and it is not clear what are appropriate restrictions on them for strictness analysis. Certainly a notion like uniformity is liable to be relevant.

Burn [2] investigated the relationship between ideal- and projection-based strictness notions using smash projections which seem to represent the intersection of the ideal- and projection-based regimes. Our uniform ideals further this investigation.

Wadler [14] suggested use of a precursor domain to ours for abstracting lists *via* a four element domain

$$\perp \sqsubseteq \infty \sqsubseteq \perp\!\in \sqsubseteq \top\!\in$$

This work was attractive because it gave domain-theoretic characterisations of the sets of (concrete) values described by the abstract domain elements. Burn observed that treating these elements as describing nested ideals instead of mutually disjoint sets gave a more natural treatment of strictness. However, this domain was linearly ordered and could not characterise all the strictness properties we wish, in particular it could not characterise H (in fact no linear order can distinguish the projections H, T and $T \circ H$ which are themselves not linearly ordered). On the other hand, he did give an explanation of *list* as an abstract domain constructor which performed double lifting on its (abstract) argument domain. We exhibit Wadler's elements as (representations of) certain of our uniform ideals.

We argue that the strictness properties of the *list* data type are *uniform* properties of its member type, just as strongly typed languages insist that all members of a list are of the same type. In our framework, both of these ideas are reflected in the notion of "belong to the same ideal". Certainly it is possible to characterise strictness properties like "strict on the 1st element" or "strict on any element at a prime number position", but this seems to be unwieldy to abstract. (Compare one of the original justification of types for functional languages: that the user was forced to distinguish between lists and pairs — this took the pressure off implementers who were expected to treat lists of alternating tags and values efficiently). Pragmatically, it remains to be discovered whether functional programmers (who after all took a while to take to types!) are willing to write (say, using ML notation) $(x, [a, b, c])$ instead of $[x, a, b, c]$ when x, a, b, c are all the same type but the value is passed to a function in which x is treated rather differently to a, b, c. Certainly strictness properties do not have the software engineering persuasiveness of types, but we believe that movement in this direction is inevitable.

This work is structured in the following manner. Section 2 explains notation and the type system which constrains terms. Section 3 describes the problems in treating projection-based strictness as ideals and sections 4 and 5 defines the notion of uniform ideal and shows Wadler's abstract list elements are exemplars. Section 6 gives the main positive and negative results relating projections and uniform ideals. This is followed by section 7 on implementation. Finally section 8 makes some remarks about strictness and type models.

2 Notation and object language

Here we use the word *domain* to mean complete (pointed) partial order as usual. Let 2 stand for the domain $\{0, 1\}$ ordered by $0 < 1$. As is usual in abstract interpretation (one cause is abstraction of programs with conditional constructs whose test is irresolvable) many of our domains will actually be lattices. Following the types-as-ideals literature, we use the word *ideal*, I say, of a domain D to mean a non-empty Scott-closed subset of a D, *i.e.*

- $(\forall i \in I)(\forall d \in D) d \sqsubseteq i \Rightarrow d \in I$ (downwards closed),
- ($\forall$ directed sets $S \subseteq I$) $\bigsqcup S \in I$ (limit closed).

It is useful to define an *ideal closure* operator on subsets of a domain D: $ic(S)$ stands for the smallest ideal containing S. Recursive domain definitions are as usual and $+, \oplus, \times, \otimes$ will mean respectively separated sum, coalesced sum, cartesian product and smash product. One common domain construction we will use is the idea of (lazy) lists, defined by $list(D) = \{[]\} + D \times list(D)$. We will use :: to mean infix 'cons' and [] to represent 'nil' in the usual manner. For examples it is useful to have tail-strict lists defined by $elist(D) = \{[]\}_\perp \oplus D_\perp \otimes elist(D)$.

For defining ideals containing lists, it is convenient to use regular expression notation, thus (for example when discussing $list(int)$ and assuming d to range over $D = \mathbb{Z}_\perp$) $ic\{\perp_D\ d^*\ []\}$ represents the ideal of all lazy lists of integers which begin with $\perp_D$ together with $\perp_{list(D)}$. Similarly $ic\{d^*[]\}$ represents all lists and partial approximations thereto (including infinite lists *via* ideal closure) and $ic\{\perp_D^* \perp\}$ stands for $\{\perp,\ \perp :: \perp,\ \perp :: \perp :: \perp,\ \ldots\}$. Note that all non-$\perp$ elements of this ideal have a $\perp$ element even though limit values may no longer have a $\perp$ tail — however this effect is natural since functions like `length` behave in the same way for all elements of the ideal.

Given a function $f : D \to E$ it is convenient to use the word *kernel* of f to refer to $f^{-1}(\perp)$. The kernel of f is a closed set (and an ideal if $f\perp = \perp$).

Typed language and interpretations

Our system T of types, ranged over by t, is given by:

$$t \in T ::= int \mid unit \mid bool \mid t + t' \mid t \times t' \mid t \to t' \mid t \oplus t' \mid t \otimes t' \mid list(t) \mid lift(t).$$

At the moment we do not allow recursive types *via* the rules $t ::= \tau \mid rec\ \tau.t$. For the purpose of illustrating uniform ideals on sum types, it is convenient to regard $bool$ as a shorthand for $unit + unit$ which is often written $\{false\} + \{true\}$. Similarly $t + t'$ acts as shorthand for $lift(t) \oplus lift(t')$.

We assume a language of terms which represent introduction and elimination for each of the above types, *e.g.* a λ-calculus with λ-abstractions to introduce $\to$ types and application to remove them.

An interpretation, A, specifies a domain D_t^A for basic type, a domain constructor for each type operator and values for each constant in the term language.

The standard interpretation given by treating types int and $unit$ as the usual domains $\mathbb{Z}_\perp$ and $\{\perp\}$, the $+$ and $list$ type constructors as the $+$ and $list$ domain constructors and the constants in the usual manner. A non-standard (here for strictness purposes) interpretation gives alternative domains and constructors together with interpretations of constants. Interpretations of constants must be related in the two interpretations to give a correctness property, but we do not concern ourselves with this here.

We will tend to write D_t for the standard interpretation of a type t and S_t as our non-standard (strictness) interpretation. In effect we give an alternative definition of the strictness interpretation of type t as the space of uniform ideals of D_t ordered by $\subseteq$, and refer to it as $\mathcal{U}[\![t]\!]$. We write $BHA[\![t]\!]$ for S_t when t is composed of int and $\to$ only.

Burn [3], following Wadler's abstract list constructor, suggests the strictness interpretation S_t given by treating types int as the usual domain 2, $unit$ as $\{\perp\}$, the $+$ and $list$ type constructors as the $\times$ and double-lifting domain constructors (note the abuse of notation where (*e.g.*) $\to$ stands for a type constructor in the first column but a domain constructor elsewhere.):

type, t	domain, D_t	strictness domain, S_t
int	$\mathbb{Z}_\perp$	2
$lift(t)$	$(D_t)_\perp$	$(S_t)_\perp$
$list(t)$	$list(D_t)$	$((S_t)_\perp)_\perp$
$t \to t'$	$D_t \to D_{t'}$	$S_t \to S_{t'}$
$t + t'$	$D_t + D_{t'}$	$(S_t \times S_{t'})_\perp$
$t \times t'$	$D_t \times D_{t'}$	$S_t \times S_{t'}$
$t \otimes t'$	$D_t \otimes D_{t'}$	$S_t \otimes S_{t'}$

This paper in effect proposes a different abstract $list(\cdot)$ domain constructor in section 7.

3 Projection based strictness

Wadler and Hughes [15] invented the idea of projection based strictness. A function $\alpha : D \to D$ is a *projection* if $\alpha \circ \alpha = \alpha$ and $\alpha \sqsubseteq 1_{D \to D}$. Their motivation was to characterise functions which evaluate their (list) arguments up to a given threshold.[4] In particular they say a function f is α-strict in a β-strict context if

$$\beta \circ f \circ \alpha = \beta \circ f.$$

They define several projections which are useful for lists, letting a range over elements:

$$\begin{array}{llcl} ID & x & = & x \\ T & l & = & (l \in ic\{a^*\perp\}) \to \perp,\ l \\ H & \perp & = & \perp \\ H & [] & = & [] \\ H & (a :: l) & = & (a = \perp) \to \perp,\ a :: H(l) \end{array}$$

Note that $T \circ H$ is also a projection. Intuitively, the property $f \circ \alpha = f$ represents that f is tail-strict (T), head-strict (H) and head- and tail-strict ($T \circ H$). Burn [2] observes that T and $T \circ H$ are smash-projections (*i.e.* $(\forall x)(\alpha(x) = x \vee \alpha(x) = \perp)$) and adds his own smash-projection:

$$\begin{array}{llcl} H_B & \perp & = & \perp \\ H_B & [] & = & [] \\ H_B & (a :: l) & = & (a = \perp) \to \perp,\ a :: l \end{array}$$

A function then such as `before0` $: list(int) \to list(int)$ defined by

$$\begin{array}{llcl} \texttt{before0} & \perp & = & \perp \\ \texttt{before0} & [] & = & [] \\ \texttt{before0} & (\texttt{a} :: \texttt{l}) & = & (\texttt{a} = \perp) \to \perp,\ (\texttt{a} = 0) \to [],\ \texttt{a} :: \texttt{before0}(\texttt{l}) \end{array}$$

illustrates the point that `before0` is ID-, H- and H_B-strict (but not T- nor $(T \circ H)$-strict) in an ID-strict context.

Burn [2] characterises ID-, H_B-, T- and $(T \circ H)$-strictness as ideal properties of lists using Wadler's ideals (see section 6 for details where ∞ is explained as $ic\{z^*\perp\}$) along the lines of:

$$\text{if } f(\infty) = \{\perp\} \text{ then } f \text{ is } T\text{-strict.}$$

We also note that the ideal $ic\{\perp z^*[]\}$, I_B say, has the property

$$\text{if } f(I_B) = \{\perp\} \text{ then } f \text{ is } H_B\text{-strict.}$$

[4] This use of the word threshold is informal and differs from ideal-oriented notions.

However, until now there has been no explanation of H in terms of ideal properties. To do this the next section defines *uniform ideal properties*, but first let us consider $\texttt{before0}^{-1}(ic\{z^*\bot\})$ to see the problems involved. This is an ideal (Scott-closed by continuity, non-empty by observation) and can be written (abbreviating $\mathbb{Z}_\bot$ as D)[5]

$$\begin{aligned}&ic\{1^*\bot_D z^*[]\} \text{ or}\\ &ic\{\ \bot_{list(D)},\ 1::\bot_{list(D)},\ 1::1::\bot_{list(D)},\ \ldots,\\ &\quad\ \bot_D::[],\ 1::\bot_D::[],\ 1::\bot_D::0::[],\ \ldots\}.\end{aligned}$$

The problem is that the sub-ideal $ic\{1^*\bot_{list(D)}\}$ is non-uniform — it is a chain whereas strictness properties are seen as largely being (the downwards closure of) co-chains. In particular any representation of this ideal by an abstract value *via* an abstract interpretation which treats $0, 1, 2, \ldots$ uniformly will lead to the bounds of $ic\{\bot z^*[]\}$ (too little) or $ic\{z^*\bot z^*[]\}$ (too big).

Section 6 examines this in more detail.

4 Uniform ideal properties

Let t be a type, and D_t be the standard domain interpretation of t. We write $I \lhd t$ to mean that I is a *uniform ideal* of t. Let $\mathcal{I}_t$ be the set of all ideals of D_t. The archetype notion of uniform ideal is that of classical type (modelled as an ideal — see MacQueen Plotkin and Sethi [10]) in which we consider ideals which model types to include, using ML-style notation, the values $[1, 2]$ and $(1, \mathit{true})$, but not $[1, \mathit{true}]$.[6] Our intent is to use uniform ideals to model strictness properties. Let $\mathcal{U}[\![t]\!]$ be the set of uniform ideals for a type t. It is given by $\mathcal{U}[\![t]\!] = \{I \mid I \lhd t\}$, and is naturally ordered, when required, by $\subseteq$.

We first give definitions for which hold for any type and those for $\times$ and $\to$:

$$\frac{}{\{\bot\} \lhd t} \qquad \frac{}{D_t \lhd t} \qquad \text{trivial properties}$$

$$\frac{I_k \lhd t \quad I'_k \lhd t' \quad (k \in \mathcal{K})}{\{f \in D_{t \to t'} \mid (\forall k)(f(I_k) \subseteq I'_k)\} \ \lhd \ t \to t'} \qquad \dagger \text{ see below}$$

$$\frac{I \lhd t \quad I' \lhd t'}{I \times I' \lhd t \times t'}$$

Basically, this says that the uniform ideals of any type are the two trivial subdomains of its interpretation, but that composite types have additional uniform ideals. Note there is no rule

$$\frac{}{\{\} \lhd t}$$

as this destroys the following isomorphism between $\mathit{BHA}[\![t]\!]$ and $\mathcal{U}[\![t]\!]$.[7]

Proposition

Writing $\mathit{BHA} : \mathit{Type} \to \mathit{Domain}$ for the Burn-Hankin-Abramsky [4] strictness types given by

$$\begin{aligned}\mathit{BHA}[\![int]\!] &= 2\\ \mathit{BHA}[\![t \to t']\!] &= \mathit{BHA}[\![t]\!] \to \mathit{BHA}[\![t']\!],\end{aligned}$$

[5] For this to be correct it is necessary to consider $\texttt{before0} : \mathit{list}(\{0,1\}) \to \mathit{list}(\{0,1\})$ or to treat '1' as a representation of all non-zero integers.

[6] Technically, this only applies when we consider a universal domain since our typed λ-calculus eliminates such ill-typed terms syntactically, but the intuition is correct and in the universal domain our uniform ideals will be sub-ideals of the domains corresponding to types.

[7] One justification for having it would be that $\Box\{\}$ (see below) then represents the ideal $\{\bot, []\}$ but we feel this is better added as a rule for list types and not present in general. Empty sets can also cause trouble in the presence of recursive function types — see MacQueen Plotkin and Sethi [10].

and assuming $D_{int} \neq \{\bot\}$ we have the isomorphism

$$BHA[\![t]\!] \cong \mathcal{U}[\![t]\!].$$

Proof by simple induction with the induction step exploiting the fact that the inference rule † for $t \to t'$ allows us to specify any (monotonic) function from $\mathcal{U}[\![t]\!]$ to $\mathcal{U}[\![t']\!]$.

Similarly, the rule for $\times$ has the property that $\mathcal{U}[\![int \times \cdots \times int \to int]\!]$ is isomorphic to Mycroft's original flat analysis domain of $2^n \to 2$.

Note that in many cases the full rule for $\to$ is not required. Kuo and Mishra seem to use a reduced rule with $|\mathcal{K}| = 1$:

$$\frac{I \lhd t \quad I' \lhd t'}{\{f \in D_{t \to t'} \mid f(I) \subseteq I'\} \ \lhd \ t \to t'}$$

for function spaces appearing to the left of (contravariently to) a $\to$. This is analogous to the distinction between the ML type system λ_1, in which λ-bound variables are not polymorphic, and the 2nd-order polymorphic λ-calculus, λ_2, in which they are.

Vickers [13] observes that the rules for lists may be extended to include intersections and unions of our listed properties, *viz*

$$\frac{I \lhd t \quad I' \lhd t'}{I \ \cap \ I' \ \lhd \ t} \qquad \frac{I \lhd t \quad I' \lhd t'}{I \ \cup \ I' \ \lhd \ t}.$$

Such rules add all joins and meets to our lattice of uniform ideals of list types. We are grateful to John Hughes for noting that such intersections help to provide accurate abstract values for *cons*, *e.g.* (see below) $cons(\underline{in}_1(I), \Diamond\underline{in}_2(I')) = \Diamond\underline{in}_1(I) \cap \Diamond\underline{in}_2(I')$. However, we are wary of advocating such rules be available at all types in the presence of functions.[8] Certainly the $\cup$ rule is also useful for the conditional function at all types, but this rule does not affect the definition of uniform ideals of type $int \times \cdots \times int \to int$ in the first-order world. It is also worthy of note is that in the presence of the $\cap$ rule then our $\to$ rule (†) is no more powerful than Kuo and Mishra's rule with $|\mathcal{K}| = 1$. We avoid a general rule which simplifies the complexity at function types but which may add spurious elements at other types.

To simplify the definitions for uniform properties over lists it is convenient first to define modal-style operators. The idea here is that properties like "the third (or even first) element of a list is $\bot$" are non-uniform but "there is a $\bot$ element of a list" is uniform. Recalling the notation $ic(S)$ stand for the *ideal closure* of S, *i.e.* the smallest ideal containing S or in other words the completion of the downwards closure of S. So we define, assuming that I is a subset (soon uniform ideal) of D_t and abusing $\in$ to also mean element of a list,

$$\begin{array}{ll}
\Diamond I,\ \Box I,\ \Diamond_\bot I,\ \Box_\bot I \subseteq D_{list(t)} & \text{by} \\
\Diamond I = ic\{l \in D_{list(t)} \mid (\exists x \in l) x \in I\} & \\
\Box I = ic\{l \in D_{list(t)} \mid (\forall x \in l) x \in I\} & \\
\Diamond_\bot I = ic\{l \in D_{list(t)} \mid (\exists x \in l) x \in I \wedge l \text{ has } \bot \text{ tail}\} & \\
\Box_\bot I = ic\{l \in D_{list(t)} \mid (\forall x \in l) x \in I \wedge l \text{ has } \bot \text{ tail}\} &
\end{array}$$

Note that not all these are distinct, *e.g.* in $list(int)$, we have $\Diamond_\bot\{\bot\} = \Diamond_\bot\{\bot\} = \Box_\bot \mathbb{Z}_\bot$.

It is also convenient to define injection functions on ideals, so that $\underline{in}_i : \mathcal{I}_{t_i} \to \mathcal{I}_{t_1+t_2}$ is given by $\lambda I.ic\{in_i(x) \mid x \in I\}$, or equivalently by $\lambda I.in_i(I) \cup \{\bot\}$. Similarly $\underline{lift}(I) = ic\{lift(x) \mid x \in I\}$.

[8] The contravariance of $\to$ effectively swaps $\cap$ and $\cup$ in argument and function types.

Inference rules for defining the uniform ideals of other types are given by:

$$\frac{I \lhd t}{\mathit{lift}(I) \lhd \mathit{lift}(t)}$$

$$\frac{I \lhd t}{\underline{in}_1(I) \lhd t + t'} \qquad \frac{I' \lhd t'}{\underline{in}_2(I') \lhd t + t'} \qquad \frac{I \lhd t \quad I' \lhd t'}{\underline{in}_1(I) \cup \underline{in}_2(I') \lhd t + t'}$$

$$\frac{I \lhd t \quad I' \lhd t'}{\underline{in}_1(I) \cup \underline{in}_2(I') \lhd t \oplus t'}$$

$$\frac{I \lhd t}{\Box I \lhd \mathit{elist}(t)} \qquad \frac{I \lhd t}{\Diamond I \lhd \mathit{elist}(t)}$$

$$\frac{I \lhd t}{\Box I \lhd \mathit{list}(t)} \qquad \frac{I \lhd t}{\Diamond I \lhd \mathit{list}(t)}$$

$$\frac{I \lhd t}{\Box_\perp I \lhd \mathit{list}(t)} \qquad \frac{I \lhd t}{\Diamond_\perp I \lhd \mathit{list}(t)}$$

The multiple rules for $+$ arise from our desire not to treat $\{\}$ as an ideal — the third rule alone would suffice in its presence.

Note that we get the standard types-as-ideals interpretation by omitting the rule for $\{\perp\}$, the two asymmetric rules for $+$, all the rules for *list*/*elist* except $\Box$ and using the $|\mathcal{K}| = 1$ form of the $\to$ rule.

So, given our interpretation of D_{int} as $\mathbb{Z}_\perp$ and *bool* defined as $\{\mathit{false}\} + \{\mathit{true}\}$ we find that the uniform ideals $\mathcal{U}[\![\mathit{int}]\!] = \{\{\perp\}, \mathbb{Z}_\perp\}$ whereas $\mathcal{U}[\![\mathit{bool}]\!] = \{\{\perp\},\ \{\perp, \mathit{true}\},\ \{\perp, \mathit{false}\},\ \{\perp, \mathit{true}, \mathit{false}\}\}$, which will be the basis of our unconventional differentiation of structure of *int* and *bool* which forms the basis of the examples in this work.

5 Wadler's abstract list domain is uniform

Wadler [14] suggested using a four element domain to model strictness properties of lists of integers:

$$\perp \sqsubseteq \infty \sqsubseteq \perp\!\in \sqsubseteq \top\!\in$$

In fact he also generalised this by suggesting the abstract domain constructor whereby if S is the strictness property domain for type t then $(S_\perp)_\perp$ is a suitable one for type $\mathit{list}(t)$. Since 2 is usually taken as the domain of strictness properties of $\mathbb{Z}_\perp$ these two definitions coincide.

These elements (re-interpreted as downwards closures of Wadler's original by Burn) can now be seen as certain cases of our uniform ideals of $\mathit{list}(t)$. For the base case, Mycroft's strictness domain, 2, for *int* yields the correspondence

$$\begin{array}{lcl} 1 = \top & \leftrightarrow & \mathbb{Z}_\perp \\ 0 = \perp & \leftrightarrow & \{\perp\} \end{array}$$

Inductively, assume I is a uniform ideal of t corresponding to Wadler strictness property p of t then we have the correspondence on $\mathit{list}(t)$:

$$\begin{array}{lcl} \mathit{lift}(\mathit{lift}(p)) & \leftrightarrow & D_{\mathit{list}(t)} = \Box I \text{ if } p \text{ is maximal} \\ & \leftrightarrow & \Diamond I \text{ otherwise} \\ \mathit{lift}\perp & \leftrightarrow & \Diamond_\perp D_t = \Diamond_\perp\{\perp\} = \Box_\perp D_t \\ \perp & \leftrightarrow & \{\perp\} \end{array}$$

For example, on $\mathit{list}(\mathit{list}(\mathit{int}))$ the Walder ideals $\{\perp, \infty, \perp\!\in, \infty\!\in, \perp\!\in\in, \top\!\in\in\}$ correspond to our ideals $\{\{\perp\}, \Box_\perp\Box\mathbb{Z}_\perp, \Diamond\{\perp\}, \Diamond\Box_\perp\mathbb{Z}_\perp, \Diamond\Diamond\{\perp\}, \Box\Box\mathbb{Z}_\perp\}$.

Figures 1 and 2 give diagrams of the uniform ideals of $\mathit{list}(\mathit{int})$ and $\mathit{list}(\{\mathit{false}\} + \{\mathit{true}\})$ with the Wadler elements highlighted. Note that some of the uniform ideal strictness properties which do not appear in Wadler's abstract domain seem quite useful (*e.g.* $\Box\{\perp\}$).

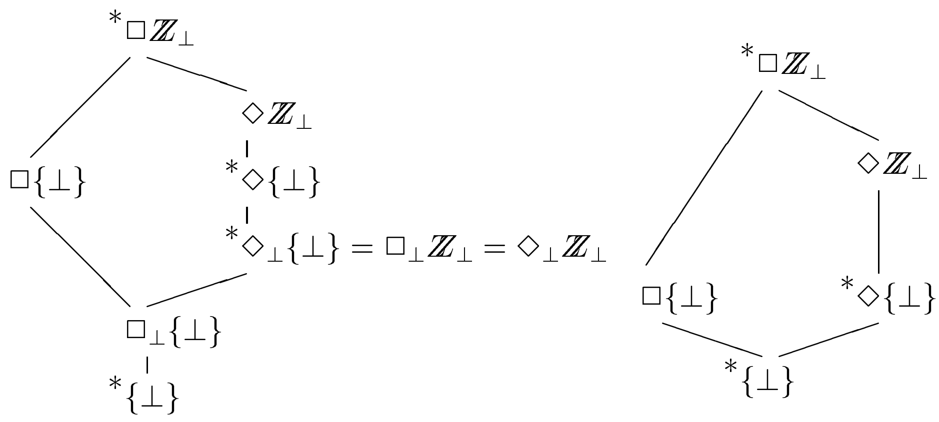

Figure 1: Uniform ideals of *list*(*int*) and of *elist*(*int*) ordered by $\subseteq$ (or $\sqsubseteq$)

$*$ denotes Wadler ideals.

Note that the only difference between $\Box\mathbb{Z}_\perp = D_{list(int)}$ and $\Diamond\mathbb{Z}_\perp$ is that the latter does not contain [], the empty list. Nevertheless this difference can be useful — $map(f, l)$ is strict in f if we can show l ranges over the latter, but not if it ranges over the former. We are grateful to John Hughes for this observation.

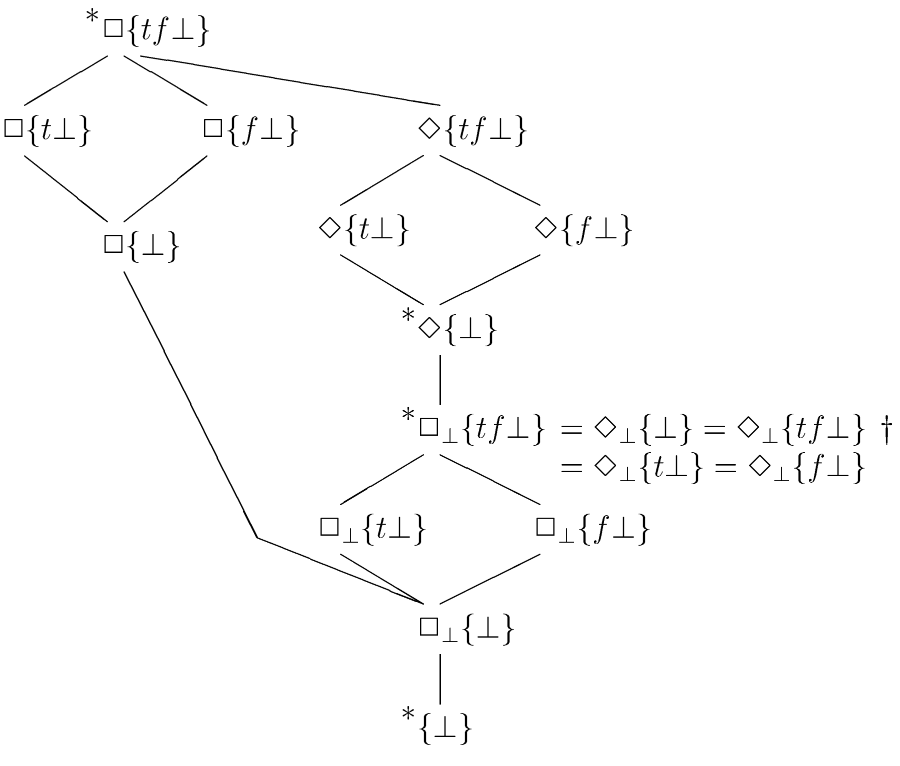

Figure 2: Uniform ideals of *list*($\{false\}+\{true\}$)

$*$ denotes Wadler ideals.

† It may seem odd that $\Diamond_\perp\{t\perp\} = \Diamond_\perp\{f\perp\}$ but (*e.g.*) $f :: \perp \in \Diamond_\perp\{t\perp\}$ since $f :: t :: \perp \in \Diamond_\perp\{t\perp\}$ and so $f :: \perp \in \Diamond_\perp\{f\perp\}$ by downwards closure.

6 Theorem

The main results of our work can be summarised as follows (1 and 2 follow from Burn [2], but their presence puts 3 and 4 in context):

Let t be a type, assumed in part 3 to be $t' + t''$ and in part 4 to be *int*. f has type $\mathit{list}(t) \to u$. Let I', I'' be uniform ideals of t', t''.

1. if $f(\Box_\perp D_t) = \{\perp\}$ then $f \circ T = f$.
2. if $f(\Diamond\{\perp\}) = \{\perp\}$ then $f \circ (T \circ H) = f$.
3. if $f(\Box_\perp \underline{in}_1(I')) = \{\perp\}$ or $f(\Box_\perp \underline{in}_2(I'')) = \{\perp\}$ then $f \circ H = f$.
4. The property of $f : \mathit{list}(\mathit{int}) \to u$ being H-strict is not in general derivable as a uniform property. Indeed the strongest uniform ideal statement (*i.e.* the largest argument ideal) one may make which implies $f \circ H = f$ is $f(\Diamond\{\perp\}) = \{\perp\}$ which reduces to 2 above.
5. Burn's H_B projection does admit of an ideal characterisation: if $f(ic\{\perp z^*[]\}) = \{\perp\}$ then $f \circ H_B = f$. However, this is not a *uniform ideal* property since it requires the first element of a list to be $\perp$.
6. Further these results are the best possible in any abstract interpretation framework based on a binary abstraction relation between concrete and abstract values which is compositional and which abstracts integers identically (uniformly). Here compositional means that the abstract semantics is given by the usual compositional form and all integers are treated the same:

$$\begin{aligned} &[\![f(e)]\!]\rho = \mathit{app}([\![f]\!]\rho)([\![e]\!]\rho) \\ &[\![n]\!]\rho = [\![0]\!]\rho = [\![1]\!]\rho. \end{aligned}$$

Proof

1. Observe $T(x) = (x \in \Box_\perp D_t) \to \perp, x$. Burn [2] calls such functions *smash projections*. So $(f \circ T)(x) = \perp = f(x)$ if $x \in \Box_\perp D_t$ and $(f \circ T)(x) = f(x)$ if $x \notin \Box_\perp D_t$ (as $T(x) = x$).
2. Similarly, with $T \circ H$, $\Diamond\{\perp\}$ replacing T, $\Box_\perp D_t$.
3. Assume $f(\Box_\perp \underline{in}_1(I')) = \{\perp\}$. Note H is not a smash projection, however, $(f \circ H)(x) = \perp = f(x)$ if $x \in \Box_\perp \underline{in}_1(I')$. But, assuming $x \notin \Box_\perp \underline{in}_1(I')$, we have that $H(x) = x$ as the cases $H\perp = \perp$ and $H(\perp :: l) = \ldots$ cannot occur. So $f \circ H = f$. Similarly if $f(\Box_\perp \underline{in}_2(I'')) = \{\perp\}$.
4. Consider the functions $\mathtt{xbefnon0}, \mathtt{xbefnon1} : \mathit{list}(\mathit{int}) \to \{\top\}_\perp$ given by

$$\begin{array}{lcl} \mathtt{xbefnon0} \;\; \perp & = & \perp \\ \mathtt{xbefnon0} \;\; [] & = & \top \\ \mathtt{xbefnon0} \;\; (\mathtt{a} :: \mathtt{l}) & = & (\mathtt{a}{=}\perp) \to \perp,\ (\mathtt{a}{\neq}\mathtt{0}) \to \top,\ \mathtt{xbefnon0}(\mathtt{l}) \\ \mathtt{xbefnon1} \;\; \perp & = & \perp \\ \mathtt{xbefnon1} \;\; [] & = & \top \\ \mathtt{xbefnon1} \;\; (\mathtt{a} :: \mathtt{l}) & = & (\mathtt{a}{=}\perp) \to \perp,\ (\mathtt{a}{\neq}\mathtt{1}) \to \top,\ \mathtt{xbefnon1}(\mathtt{l}) \end{array}$$

   Any uniform (here just abstracting 0 and 1 identically) compositional abstract interpretation which is based on the notion of "the abstraction of a value in the standard interpretation"[9] will ascribe identical abstract meanings to `xbefnon0` and `xbefnon1`.

   Now, the intersection of $\mathtt{xbefnon0}^{-1}\{\perp\}$ and $\mathtt{xbefnon1}^{-1}\{\perp\}$ is precisely $ic\{\perp z^*[]\}$ (it may help to consider the diagram of $\mathtt{before0}^{-1}(ic\{z^*\perp\})$ from section 3). Interestingly this is the

[9] Hunt's does not appear to be.

kernel I_B of Burn's H_B projection. In some sense then, although it is non-uniform, H_B is the nearest projection to H which is fully modellable in an abstract interpretation based on abstracting *values*.

5. Done above.

6. Done above.

Note that 1 and 2 are reversible implications if we further add the additional assumption that f is strict [2]. The implication in 3 is not reversible since we may imagine a function which scans a list, halting before [] only if it reaches a *false* in an even position or a *true* in an odd position. It may be desirable to handle such functions, and a more general notion of uniform property on lists which includes regular language properties may help. Similarly a notion of *sequential* ideal property on lists (which would include all values in the ideal closure of (*e.g.*) $\{true^* \bot b^* []\}$ (where b ranges over $false + true$) is also worth investigating further. Hunt's use of PERs rather than ideals appears to be more satisfactory at characterising these forms of H-strictness.

To show the result 3. is non-vacuous, one may observe that `xbeftrue` : $list(false + true) \to \{\top\}_\perp$ given by

$$\begin{array}{lcll}
\texttt{xbeftrue} & \perp & = & \perp \\
\texttt{xbeftrue} & [] & = & \top \\
\texttt{xbeftrue} & (\perp \ :: \ \texttt{l}) & = & \perp \\
\texttt{xbeftrue} & (in_2(true) \ :: \ \texttt{l}) & = & \top \\
\texttt{xbeftrue} & (in_1(false) \ :: \ \texttt{l}) & = & \texttt{xbeftrue(l)}
\end{array}$$

satisfies the ideal property

$$f(\Box_\perp\{false\}) = \{\perp\}.$$

and is hence H-strict.

7 Abstract domain constructors and mechanisation

Although the operators like $\Box, \Diamond, \underline{in}_i, \ldots$ have been defined as operators on ideals, they can be used as syntactic objects which name abstract domain points corresponding to ideals. Moreover, the abstract lattice so obtained for a (non-recursive) type t is finite. Indeed it appears to match S_t for all non-*list* types. Hence the uniform ideal properties are finitely computable. Although the syntactic representations of *list* uniform ideals are non-canonical in that (*e.g.*) $\Box_\perp \mathbb{Z}_\perp = \Diamond_\perp\{\perp\}$, Cousot observes that the correctness (of abstract interpretation) theory can still be carried through. A suitable definition for $S_{list(t)}$ might be $2 \times S_t + S_t$ where the '+' distinguishes $\Box, \Diamond$ and the '$\times$' distinguishes $\Box, \Box_\perp$. Since the $\Diamond_\perp I$ all collapse to a (the same) ideal of the form $\Box_\perp I'$, it is not necessary to distinguish $\Diamond, \Diamond_\perp$.

8 Connection of strictness properties and types

Recall that in the "types as ideals" approach, the semantics of the (lazy) list type $\forall\alpha.list(\alpha)$ would be the ideal $\Box\{\perp_D\}$ containing $ic\{\perp_D^*[]\}$, probably injected into a universal domain. This sub-ideal of all of the ideals corresponding to the types $list(t)$ (polymorphism is treated as intersection) can be seen as a useful strictness property too, since it can be used to characterise the strictness behaviour of functions like *nth*:

$$\begin{array}{lllcl}
\texttt{nth} & \texttt{n} & [] & = & \perp \\
\texttt{nth} & \texttt{n} & (\texttt{a} :: \texttt{l}) & = & \texttt{n}{=}0 \to \texttt{a},\ \texttt{nth}\ (\texttt{n}-1)\ \texttt{l}
\end{array}$$

whose strictness properties include $nth(\mathbb{Z}_\perp)(\Box\{\perp\}) \subseteq \{\perp\}$ and $nth\{\perp\}(\Box\mathbb{Z}_\perp) \subseteq \{\perp\}$. One (ground type instance) *type* property of `nth` is represented by $nth(\mathbb{Z}_\perp)(\Box\mathbb{Z}_\perp) \subseteq \mathbb{Z}_\perp$.

So there is a fundamental sense in which strictness properties behave like types, which was noted by Kuo and Mishra [9] and could be used to underpin Wright [16]. Moreover, treating types and strictness properties both as ideals has the benefit of giving a uniform treatment of strictness properties as types and to a unified treatment of polymorphism in types and strictness analysis since (as Kuo and Mishra observe) Abramsky's [1] polymorphic invariance seems a little removed from polymorphism in types.

Further work

It would be nice to extend this work to cover lists defined by their λ-calculus coding to formalise [5].

The notion of uniform properties developed here has parallels with Nielson's *expected forms* analysis.

More work should be done on deciding exactly what uniform properties on lists are 'useful' for strictness properties. Similarly, it is desirable that we develop a more general theory of which properties of a general polynomial recursive data type (*e.g.* the ML/Haskell *datatype* construct) are uniform instead of our consideration of the special case of lists.

Similarly, it would be desirable to consider whether certain finite lattices could represent strictness properties for the untyped λ-calculus modelled as the recursive type $u = \mathit{int} + (u \to u)$, instead as the infinite chains present in Hudak and Young.

Acknowledgments

Comments made at a seminar at Imperial College greatly aided the presentation of this work, in particular we are indebted to Samson Abramsky, Geoffrey Burn, Sebastian Hunt and Steve Vickers. We are indebted to the referees for their careful reading and apposite comments and also to John Hughes for his remarks on abstract '*cons*' operations. The authors thank the Computer Laboratory and Robinson College, Cambridge for supporting visits during this research.

References

[1] Abramsky, S. Strictness analysis and polymorphic invariance. In [8].

[2] Burn, G. A relationship between abstract interpretation and projection analysis (extended abstract). Proc. 17th ACM symp. on Principles of Programming Languages, 1990.

[3] Burn, G. Lazy functional languages: interpretation and compilation. Unpublished manuscript, forthcoming book, 1990.

[4] Burn, G., Hankin, C. and Abramsky, S. The theory and practice of strictness analysis for higher order functions. In [8].

[5] Ernoult, C. Higher-order strictness analysis over non-flat domains. INRIA (Rocquencourt) rapport 1190, 1990.

[6] Hudak, P. and Young, J. Higher order strictness analysis in untyped lambda calculus. Proc. 13th ACM symp. on Principles of Programming Languages, 1986.

[7] Hunt, L.S. PERs generalise projections for strictness analysis. Departmental report DOC 90/14, Dept. of Computing, Imperial College, London.

[8] Jones, N.D. and Ganzinger, H. (eds.) Programs as Data Objects. Lecture Notes in Computer Science: Proc. of a Workshop, Copenhagen, vol. 215, Springer-Verlag, 1985.

[9] Kuo, T.-M. and Mishra, P. Strictness analysis: a new perspective based on type inference. ACM-IFIP, Proc. of the functional programming and computer architecture conference, 1989.

[10] MacQueen, D., Plotkin, G.D. and Sethi, R. An ideal model for recursive polymorphic types. Proc. 11th ACM symp. on Principles of Programming Languages, 1984.

[11] Mycroft, A. Abstract interpretation and optimising transformations of applicative programs. Ph.D. thesis, Edinburgh University, 1981. Available as computer science report CST-15-81.

[12] Nielson, F. Abstract interpretation using domain theory. Ph.D. thesis, Edinburgh University, 1984. Available as computer science report CST-31-84.

[13] Vickers, S. Topology via logic. Cambridge Tracts in Theoretical Computer Science 5. Cambridge University Press 1989.

[14] Wadler, P.L. Strictness analysis on non-flat domains (by abstract interpretation over finite domains). In Abramsky, S. and Hankin, C. (eds.) Abstract interpretation of declarative languages. Ellis Horwood, Chichester, UK, 1987.

[15] Wadler, P.L. and Hughes R.J.M. Projections for strictness analysis. In G. Kahn (ed.) Lecture Notes in Computer Science: Proc. of the functional programming and computer architecture conference, vol. 274, Springer-Verlag, 1987.

[16] Wright, D.A. Strictness analysis via (type) inference. Report R89-3, Electrical Engineering and Computer Science Dept., Tasmania University, 1989.