Algebra of Logical Relations

Roland Backhouse

19th December 2001

Outline

- Theorems For Free
- Abstract Interpretation
- Point-Free Laws
- Factorisation Lemma
- Example

Theorems-For-Free (Reynolds, Wadler)

Example: the function *length* on lists has type

$$List.a \to \mathbb{N}$$

for all instances of the type parameter a.

From this type, a "theorem-for-free" is that *length* is a natural transformation: for all functions f and all lists as,

$$length.as \;=\; length.(List.f.as) \quad .$$

(Note: *List* is used where functional programmers would traditionally use `map`.)

Example: the function *fold* has type

$$List.a \times a \times (a \times a \to a) \to a$$

for all instances of the type parameter a.

From this type, a "theorem-for-free" is that, for all lists of integers ms and all integers m,

$$even.fold(ms\,,m\,,(\times)) \;\equiv\; fold(List.even.ms\,,even.m\,,(\vee)) \quad .$$

All uses of parametricity to date have been "general": they say something about possible implementations of the polymorphic lambda calculus ... The main contribution of this paper is to suggest that parametricity also has "specific" applications: it says interesting things about particular functions with particlar types.

Wadler, "Theorems for free", International Symposium on Functional Programming Languages and Computer Architecture, London, September, 1989.

Logical Relations

Notation I will use lower case letters f, g, etc. for total functions and upper case letters R, S etc. for binary relations.

Definition Let R and S be binary relations. Define the binary relation $R \rightarrow S$ on functions by

$$(f,g) \in R \rightarrow S \quad \equiv \quad \langle \forall u,v :: \ (u,v) \in R \ \Rightarrow \ (f.u \, , \, g.v) \in S \rangle$$

In words, f and g map R-related values to S-related values.

Theorems-For-Free

Let t be a type expression parameterised by type variables a, b, $\ldots$, c and suppose f is a parametrically polymorphic function of type t for all instances of a, b, $\ldots$, c. Then for arbitrary relations R, S, $\ldots$, T,

$$(f,f) \ \in \ t\,[a,b,\ldots,c := R,S,\ldots,T]$$

Example

Suppose $\sqsubseteq$ and $\preceq$ are partial orderings. Then, the statement

$$(f, f) \in \sqsubseteq \to \leq$$

means that f is a monotonic function. That is,

$$\langle \forall u, v :: \; u \sqsubseteq v \;\Rightarrow\; f.u \leq f.v \rangle \;\; .$$

Function composition has type

$$(b \to c) \times (a \to b) \;\to\; (a \to c)$$

for all instances of the type parameters a, b and c. From this type, a "theorem-for-free" is that function composition preserves monotonicity of functions: for all partial orderings $\sqsubseteq$, $\leq$ and $\preceq$ and all functions f and g

$$f \in \sqsubseteq \to \leq \;\wedge\; g \in \preceq \to \sqsubseteq \;\Rightarrow\; f \bullet g \in \preceq \to \leq$$

(where $f \in \sqsubseteq \to \leq$ is an abbreviation for $(f, f) \in \sqsubseteq \to \leq$).

Abstract Interpretation

Abstract interpretation is a technique for approximating the execution behaviour of a computer program.

For example, abstract interpretation aims to answer questions like

is $231 \times 15 \times 44 \times 26 \times 179 \times 182$ even? and

is $231 \times 15 \times 44 \times 26 \times 179 \times 182$ divisible by six?

without performing the multiplications.

Application of results outlined here is to show that safety of abstract interpretation of higher-order functions is a “free” theorem.

See Kevin Backhouse and Roland Backhouse, *Abstraction Interpretations for Free*.

The Problem — Illustrated

Let k be a fixed positive integer. Consider the relation DK between integers m and booleans b defined by

$$(m,b) \in DK \quad \equiv \quad k \backslash m \Leftarrow b \ .$$

(Read $k \backslash m \Leftarrow b$ as "k divides m if b".)

DK relates all integers to *false* and the integers divisible by k to *true*.

Assume the function *fold* has type

$$List.a \times a \times (a \times a \to a) \to a$$

for all instances of type parameter a. Then, theorems-for-free says that

$$(fold, fold) \ \in \ List.DK \times DK \times (DK \times DK \to DK) \to DK \ .$$

But what does this mean?

According to the definition ...

$$(fold, fold) \in List.DK \times DK \times (DK \times DK \to DK) \to DK$$

means

$$\begin{aligned}
&\langle \forall\, ms, bs, m, b, \oplus, \otimes \\
&\qquad : \quad (ms, bs) \in List.DK \\
&\qquad\quad \wedge (m, b) \in DK \\
&\qquad\quad \wedge \langle \forall\, n, c, p, d \\
&\qquad\qquad : \quad (n, c) \in DK \wedge (p, d) \in DK \\
&\qquad\qquad : \quad (n \oplus p\,,\, c \otimes d) \in DK \\
&\qquad\qquad \rangle \\
&\qquad : \quad (fold(ms\,, m\,, \oplus)\;,\; fold(bs\,, b\,, \otimes)) \in DK \\
&\qquad \rangle
\end{aligned}$$

But what does this mean?

The Solution — Point-Free

Definition

$$(f,g) \in R \to S \quad \equiv \quad R \subseteq f^{\cup} \bullet S \bullet g \ .$$

Distributivity Properties

$$(R \to S)^{\cup} \quad = \quad R^{\cup} \to S^{\cup}$$

$$(R \to S) \bullet (T \to U) \ \subseteq \ (R \bullet T) \to (S \bullet U)$$

$$(R \bullet f^{\cup}) \to (S \bullet g) \quad = \quad (R \to S) \bullet (f^{\cup} \to g)$$

Monotonicity

$$R \to S \ \subseteq \ T \to U \quad \Leftarrow \quad R \supseteq T \ \wedge \ S \subseteq U$$

Shunting

$$f \bullet R \subseteq S \quad \equiv \quad R \subseteq f^{\cup} \bullet S$$

Type Expressions

We consider the following (extended BNF) grammar of type expressions

$$\mathit{TypeExp} \quad ::= \quad (\mathit{Relator}\ \mathit{TypeExp}^*) \mid (\mathit{TypeExp} \to \mathit{TypeExp}) \mid \mathit{Variable}$$

where *Variable* and *Relator* are (disjoint) finite sets.

An example of a type expression is

$$\mathit{Production} \to b \times \mathit{List}.a \to a \times \mathit{List}.b \ .$$

I will use lower case letters a, b, etc. for variables.

Variable Assignments

A *variable assignment* is a function with domain *Variable*. A variable assignment will be denoted by an assignment statement as in, for example,

$$a,b := \mathit{integer},\mathit{boolean} \ .$$

Given a variable assignment V, its application to type variable a is denoted by V_a.

Extend variable assignments to type expressions in two ways. First, for all type expressions u, v

$$V_{u\to v} = V_u \to V_v$$

and, for all type expressions u, $\ldots$, v,

$$V_{F.(u\ldots v)} = F.(V_u \ldots V_v) \ .$$

Positive and Negative Variables

For all variables a,

$$[V,W]_a \;=\; V_a$$

For all type expressions u and v,

$$[V,W]_{u\to v} \;=\; [W,V]_u \to [V,W]_v$$

For all type expressions u, ..., v,

$$[V,W]_{F.(u\ldots v)} \;=\; F.([V,W]_u \ldots [V,W]_v)$$

For example,

$$\left[(a := dk)\,,(a := kd^{\cup})\right]_{List.a\times a\times(a\times a\to a)\to a}$$

$$=\; List\,.\,kd^{\cup} \times kd^{\cup} \times (dk\times dk\to kd^{\cup}) \to dk \;\;.$$

Factorisation of functions

Suppose R, S, f and g are type assignments such that, for all type variables a, f_a and g_a are total functions.

Then, for all type expressions t,

$$[R,S]_t \bullet [f^{\cup}, g]_t \subseteq [R \bullet f^{\cup}, S \bullet g]_t$$

and

$$[R,S]_t \bullet [f, g^{\cup}]_t \supseteq [R \bullet f, S \bullet g^{\cup}]_t \ .$$

Proof (extract only)

$$
\begin{array}{ll}
 & [R,S]_{u\to v} \bullet \left[f\,, g^{\cup}\right]_{u\to v} \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & ([S,R]_u \to [R,S]_v) \bullet (\left[g^{\cup}\,, f\right]_u \to \left[f\,, g^{\cup}\right]_v) \\
= & \quad \{ \quad \text{distributivity} \quad \} \\
 & [S,R]_v \bullet \left[g^{\cup}\,, f\right]_v \to [R,S]_u \bullet \left[f\,, g^{\cup}\right]_u \\
\supseteq & \quad \{ \quad \text{monotonicity, induction hypotheses} \quad \} \\
 & \left[S \bullet g^{\cup}\,,\, R \bullet f\right]_u \to \left[R \bullet f\,,\, S \bullet g^{\cup}\right]_v \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \left[R \bullet f\,,\, S \bullet g^{\cup}\right]_{u\to v} \quad .
\end{array}
$$

Application

For brevity, let t denote $List.a \times a \times (a \times a \to a)$

$$
\begin{array}{ll}
 & List.even \times even \times (even \times even \to even) \\
= & \quad \{ \quad \text{introduce the identity relation } I \quad \} \\
 & (List.even \bullet I^{\cup}) \times (even \bullet I^{\cup}) \times ((I \bullet even) \times (I \bullet even) \to even \bullet I^{\cup}) \\
\supseteq & \quad \{ \quad \text{factorization of functions, definition of } t \quad \} \\
 & [even, I]_t \bullet [I, even]_t \quad .
\end{array}
$$

Application (Continued)

Hence,

$$
\begin{array}{ll}
 & (fold, fold) \;\in\; List.even \times even \times (even \times even \to even) \to even \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & fold \bullet List.even \times even \times (even \times even \to even) \;\subseteq\; even \bullet fold \\
\Rightarrow & \quad \{ \quad \text{above, monotonicity} \quad \} \\
 & fold \bullet [even, I]_t \bullet [I, even]_t \;\subseteq\; even \bullet fold \\
= & \quad \{ \quad [I, even]_t \;=\; \left[I\,, even^{\cup}\right]_t^{\cup}\text{, shunting} \quad \} \\
 & fold \bullet [even, I]_t \;\subseteq\; even \bullet fold \bullet \left[I\,, even^{\cup}\right]_t \\
= & \quad \{ \quad \text{inclusion of total functions is equality} \quad \} \\
 & fold \bullet [even, I]_t \;=\; even \bullet fold \bullet \left[I\,, even^{\cup}\right]_t \;\; .
\end{array}
$$

This is the fusion property of *fold* mentioned earlier.

It shows that *fold* is a *dinatural* transformation and illustrates the algebraic calculation used to derive the general property.