A (Calculational) Look at Optimization

J.N. Oliveira

Dept. Informática,
Universidade do Minho
Braga, Portugal

Mondrian Workshop#01
8-9 July 2010 (updated: August 2010
Aveiro, Portugal

Motivation

Questions:

- Why is **programming**, or **systems design** "difficult"?
- Is there a generic skill, or competence, that one such acquire to become a "good programmer"?

What **makes** programming difficult?

- **Technology** (mess) — don't fall in the trap: simply **abstract** from it!
- **Requirements** — again abstract from these as much as possible — too, write formal models or specs

Specifications:

- What is it that makes the specification of a problem hard to fulfill?

Problems = Easy + Hard

Superlatives in problem statements, eg.

- *"... the smallest such number"*
- *"... the longest such list"*
- *"... the best approximation"*

suggest two layers in specifications:

- the **easy** layer — **broad** class of solutions (eg. a *prefix* of a list)
- the **difficult** layer — requires one **particular** such solution regarded as **optimal** in some sense (eg. "shortest with maximal density").

Example

Requirements for **whole division** $x \div y$:

- Write a program which computes number z which, multiplied by y, approximates x.
- Check your program with the following test data:
  $x, y, z = 7, 2, 1$
  $x, y, z = 7, 2, 2$
- Ups! Forgot to tell that I want the **largest** such number (sorry!):
  $x, y, z = 7, 2, 3$

Deriving the algorithm... from what?

... *where is the formal specification of* $x \div y$*?*

Example

Requirements for **whole division** $x \div y$:

- Write a program which computes number z which, multiplied by y, approximates x.
- Check your program with the following test data:
  $x, y, z = 7, 2, 1$
  $x, y, z = 7, 2, 2$
- Ups! Forgot to tell that I want the **largest** such number (sorry!):
  $x, y, z = 7, 2, 3$

Deriving the algorithm... from what?

... *where is the formal specification of* $x \div y$*?*

Example — writing a spec

First version (literal):

$$x \div y = \langle \bigvee \; z \; :: \; z \times y \leq x \rangle \tag{1}$$

Second version (involved):

$$z = x \div y \;\Leftrightarrow\; \langle \exists\, r \;:\; 0 \leq r < y :\; x = z \times y + r \rangle \tag{2}$$

Third version (clever!):

$$z \times y \leq x \;\Leftrightarrow\; z \leq x \div y \qquad (y > 0) \tag{3}$$

— a Galois connection.

Why (3) is better than (1,2)

It captures the requirements:

- It is a solution: $x \div y$ multiplied by y approximates x

$$(x \div y) \times y \leq x$$

  (let $z := x \div y$ in (3) and simplify)

- It is the best solution because it provides the **largest** such number:

$$z \times y \leq x \;\Rightarrow\; z \leq x \div y \qquad (y > 0)$$

  (the $\Rightarrow$ part of $\Leftrightarrow$).

Main advantage:

Highly calculational!

Dissecting GCs

- Elsewhere, Silva and Oliveira (2008) follow the "GCs as specs" motto and show how to derive $x \div y$ from its defining GC.

- Today I would like to focus on a particular class of GCs in which the **easy+hard** split is particularly apparent.

- We will handle such GCs in the relational pointfree style, eventually leading to specs elegantly captured by a binary combinator of shape

$$E \upharpoonright H$$

where E (=easy) provides the broad class of solutions and H (=hard) provides the criterion for optimizing E so as to obtain the "*superlative* effect".

GCs as specs — examples

The *take* function on lists (longest prefix up-to specified length) is the upper adjoint of GC

$$\mathrm{len}\, y \leq n \wedge y \sqsubseteq x \;\Leftrightarrow\; y \sqsubseteq \mathit{take}(n, x)$$

(Oliveira, 2010). Another GC,

$$\langle \forall\, i \;:\; i \in \mathit{inds}\; y :\; p(y\; i) \rangle \wedge y \sqsubseteq x \;\Leftrightarrow\; y \sqsubseteq \mathit{takewhile}\; p\; x$$

specifies *takewhile p* (longest prefix meeting condition p).

Abstract pattern

Both GCs above (and many others!) share the abstract pattern

$$p\ y\ \wedge\ y \sqsubseteq x\ \Leftrightarrow\ y \sqsubseteq h\ x \tag{4}$$

meaning:

- a generic GC between all objects y which satisfy property p and all arbitrary such objects (x).
- lower-adjoint is an *embedding*
- upper-adjoint h is such that $h\ x$ yields the best **approximation** to x which satisfies p.
- (Typically, $\sqsubseteq$ will be a partial order.)

Aims

How much can we expect from (4)?

- A lot, as we will see
- **But**, we need to go *pointfree*

This means "shrinking" equivalence

$$p\ y \ \wedge\ y \sqsubseteq x \ \Leftrightarrow\ y \sqsubseteq h\ x$$

into (relational) equality

$$\Phi_p \cdot \sqsubseteq \ = \ \sqsubseteq \cdot h \tag{5}$$

where "$\cdot$ " means relational **composition** and Φ_p denotes the **partial identity** which captures property p. Details follow.

Relational composition and equality

Composition:

$$\begin{tikzcd} B & A \arrow[l, "R"'] & C \arrow[l, "S"'] \arrow[ll, bend left, "R \cdot S"] \end{tikzcd} \qquad (6)$$

$$b(R \cdot S)c \ \Leftrightarrow\ \langle \exists\ a\ ::\ b\ R\ a \wedge a\ S\ c \rangle \qquad (7)$$

In case S is a function (say h):

$$b(R \cdot h)c \quad \Leftrightarrow \quad b\ R\ (h\ a) \qquad (8)$$

Equality:

$$R = S \quad \Leftrightarrow \quad \langle \forall\ b, a\ ::\ b\ R\ a \Leftrightarrow b\ S\ a \rangle \qquad (9)$$

(as happens in GCs.)

Converses and partial identities (coreflexives)

Converses: every $B \xleftarrow{R} A$ has a **converse** $B \xrightarrow{R^\circ} A$ such that:

$$a(R^\circ)b \quad \Leftrightarrow \quad b\ R\ a \tag{10}$$

Coreflexives: binary relation encodings of unary predicates:

$$b\ \Phi_p\ a \quad \Leftrightarrow \quad b = a \wedge (p\ a) \tag{11}$$

Thus, given unary predicate $Bool \xleftarrow{p} A$, relation $A \xleftarrow{\Phi_p} A$ is the largest fragment of the identity $A \xleftarrow{id} A$ to involve objects satisfying p:

$$X \subseteq \Phi_p \quad \Leftrightarrow \quad X \subseteq id \ \wedge \ \langle \forall\ a\ :\ a\ X\ a :\ p\ a \rangle$$

Relational types

Note the arrow notation used for relations in the same way as for functions. This extends to writing arrows such as, for instance,

$$\Phi_p \xleftarrow{\sqsubseteq} \Phi_p$$

to mean the same as

$$\sqsubseteq \cdot \Phi_p \;\subseteq\; \Phi_p \cdot \sqsubseteq \tag{12}$$

In words: if an upper-bound satisfies p then the lower-bound does so as well. It can be checked that this means the same as the pointwise

$$x \sqsubseteq y \wedge (p\ y) \quad \Rightarrow \quad p\ x$$

In other words: property p is **downward closed**.

Calculating with generic GC

$$
\begin{array}{ll}
& \Phi_p \cdot \sqsubseteq \;=\; \sqsubseteq \cdot h \\
\Leftrightarrow & \qquad \{ \text{ anti-symmetry } \} \\
& \Phi_p \cdot \sqsubseteq \;\subseteq\; \sqsubseteq \cdot h \;\wedge\; \sqsubseteq \cdot h \subseteq \Phi_p \cdot \sqsubseteq \\
\Leftrightarrow & \qquad \{ \; h \subseteq \Phi_p \cdot \sqsubseteq \;\Leftrightarrow\; \sqsubseteq \cdot h \subseteq \Phi_p \cdot \sqsubseteq \text{ because } p \text{ is downward closed } \} \\
& \Phi_p \cdot \sqsubseteq \;\subseteq\; \sqsubseteq \cdot h \;\wedge\; h \subseteq \Phi_p \cdot \sqsubseteq \\
\Leftrightarrow & \qquad \{ \text{ converses ; swap conjuncts } \} \\
& h \subseteq \Phi_p \cdot \sqsubseteq \;\wedge\; (\Phi_p \cdot \sqsubseteq)^\circ \subseteq h^\circ \cdot \sqsupseteq \\
\Leftrightarrow & \qquad \{ \text{ shunting on } h^\circ \} \\
& \underbrace{h \subseteq \Phi_p \cdot \sqsubseteq}_{\text{“easy”}} \;\wedge\; \underbrace{h \cdot (\Phi_p \cdot \sqsubseteq)^\circ \subseteq \sqsupseteq}_{\text{“hard”}}
\end{array}
$$

Calculating with generic GC

Comments:

- Easy part: $h \subseteq \Phi_p \cdot \sqsubseteq$ — ensures h yielding approximations satisfying p
- Hard part: $h \cdot (\Phi_p \cdot \sqsubseteq)^\circ \subseteq \sqsupseteq$ — ensures h yielding **the best** such approximation.

Let us define a new combinator for this:

in general, given relation $B \xleftarrow{R} A$

and optimization criterion $B \xleftarrow{S} B$
on its outputs,

$$
\begin{tikzcd}
 & A \arrow[dl, "R \upharpoonright S"'] \arrow[d, "R"] \\
B & B \arrow[l, "S"]
\end{tikzcd}
$$

define $R \upharpoonright S$ satisfying universal property:

$$X \subseteq R \upharpoonright S \quad \Leftrightarrow \quad X \subseteq R \ \wedge \ X \cdot R^\circ \subseteq S \tag{13}$$

This is explained below with points (and words).

The "R optimized by S" combinator

- The $\Leftarrow$ part of the given property

$$X \subseteq R \upharpoonright S \;\Leftarrow\; X \subseteq R \;\wedge\; X \cdot R^\circ \subseteq S$$

$$\begin{tikzcd}
 & a \arrow[dl, "X"'] \arrow[d, "R"] \\
b' & b \arrow[l, "S"]
\end{tikzcd}$$

ensures $R \upharpoonright S$ as the largest sub-relation X of R such that, for all $b', b \in B$, if there exists $a \in A$ such that $b' X a \wedge b R a$, then $b' S b$ holds ("b' better than b").

- The same in a closed formula,

$$R \upharpoonright S \;=\; \underbrace{R}_{easy} \cap \underbrace{S / R^\circ}_{hard} \tag{14}$$

thanks to the GC of relational division (compare with **integer** division):

$$X \cdot R \subseteq S \;\Leftrightarrow\; X \subseteq S \,/\, R \tag{15}$$

Role of division (“hard” part)

With points:

$$c(S \,/\, P)a \;\Leftrightarrow\; \langle \forall\ b \ :\ a\ P\ b:\ c\ S\ b \rangle$$

$$\begin{tikzcd} & a \arrow[dl, "S/P"'] \\ c & b \arrow[u, "P"'] \arrow[l, "S"] \end{tikzcd}$$

Thus, $b'(R \upharpoonright S)a$ means

$$b'\ R\ a\ \wedge\ \langle \forall\ b\ :\ b\ R\ a:\ b'\ S\ b \rangle$$

Comments:

- Reasoning with quantifiers would mean “going one century back”.
- Instead, we resort on the algebra of relational division — see eg. next slide.

Role of division ("hard" part)

From GC $X \cdot R \subseteq S \;\Leftrightarrow\; X \subseteq S \,/\, R$ infer:

- (Right) cancellation:

$$(S/R) \cdot R \subseteq S \tag{16}$$

- Upper-adjoint distribution:

$$(S \cap P)/R = (S/R) \cap (P/R) \tag{17}$$

- Lower-adjoint distribution:

$$(X \cup Y) \cdot R = X \cdot R \cup Y \cdot R \tag{18}$$

etc

Algebra of $R \upharpoonright S$

- First intuitions arose when dealing with lists in Alloy in calculating the journaled refinement of a FLASH memory model, see (Ferreira and Oliveira, 2010) — no head/tail recursion in Alloy!
- Example of $R \upharpoonright S$ where R is a data-structure:

$$\left(\begin{array}{c|c} \mathit{Mark} & \mathit{Student} \\ \hline 10 & \mathit{John} \\ 11 & \mathit{Mary} \\ 12 & \mathit{John} \\ 15 & \mathit{Arthur} \end{array}\right) \upharpoonright \geq \quad = \quad \begin{array}{c|c} \mathit{Mark} & \mathit{Student} \\ \hline 11 & \mathit{Mary} \\ 12 & \mathit{John} \\ 15 & \mathit{Arthur} \end{array}$$

- Since then, I've been developing the algebra of $R \upharpoonright S$ on a "call by need" fashion.

Basic properties

Chaotic optimization:

$$R \upharpoonright \top \;=\; R \tag{19}$$

Impossible optimization:

$$R \upharpoonright \bot \;=\; \bot \tag{20}$$

Force determinism:

$$R \upharpoonright id \;=\; \text{largest deterministic fragment of } R \tag{21}$$

Pre-condition fusion:

$$(R \upharpoonright S) \cdot \Phi \;=\; (R \cdot \Phi) \upharpoonright S \tag{22}$$

Basic properties

Function fusion (where R_f abbreviates $f^\circ \cdot R \cdot f$):

$$(R \upharpoonright S) \cdot f \;=\; (R \cdot f) \upharpoonright S \tag{23}$$

$$(f \cdot S) \upharpoonright R \;=\; f \cdot (S \upharpoonright R_f) \tag{24}$$

Ensure simplicity (determinism):

$$R \upharpoonright S \text{ is simple} \;\Leftarrow\; S \text{ is anti-symmetric} \tag{25}$$

Deterministic (simple) = already optimized: for R simple,

$$R \upharpoonright S = R \quad\Leftrightarrow\quad \mathrm{img}\, R \subseteq S \tag{26}$$

Thus (functions)

$$f \upharpoonright S = f \quad\Leftarrow\quad S \text{ is reflexive} \tag{27}$$

Basic properties

Union:

$$(R \cup S) \upharpoonright Q = (R \upharpoonright Q) \cap Q/S^\circ \cup (S \upharpoonright Q) \cap Q/R^\circ \tag{28}$$

This has a number of corollaries, namely conditionals:

$$(P \rightarrow R \ , \ T) \upharpoonright S \quad = \quad P \rightarrow (R \upharpoonright S) \ , \ (T \upharpoonright S) \tag{29}$$

Disjoint union:

$$[R, S] \upharpoonright U \quad = \quad [R \upharpoonright U, S \upharpoonright U] \tag{30}$$

where the *junc* operator

$$[R, S] \quad \triangleq \quad R \cdot i_1^\circ \ \cup \ S \cdot i_2^\circ \tag{31}$$

is associated to relational coproducts.

The "function competition" rule

A corollary of the union rule,

$$(f \cup g) \upharpoonright S \;=\; (f \cap S \cdot g) \cup (g \cap S \cdot f) \qquad (32)$$

since $S / g^{\circ} = S \cdot g$. Comments:

- For S anti-symmetric, $(f \cup g) \upharpoonright S$ is always simple at the cost of not being entire.
- If furthermore one function (say g) "always wins" over the other with respect S — $(g\ x) S (f\ x)$ for all x — then $(f \cup g) \upharpoonright S = g$.

Details in the next slide.

The "function competition" rule

From (32) we easily infer a side condition for g to win over f:

$$(f \cup g) \upharpoonright S = g \quad \Leftarrow \quad g \subseteq S \cdot f \wedge f \subseteq (S \cdot g \Rightarrow g) \tag{33}$$

Note that:

- Condition $f \subseteq (S \cdot g \Rightarrow g)$ — which ensures that the outcome is a function — can be dropped for anti-symmetric S.
- This is so because $f \subseteq S^\circ \cdot g$ (the same as the first conjunct, taking converses) eventually makes $f \subseteq (S \cdot g \Rightarrow g)$ equivalent to $f \cap (S \cap S^\circ) \cdot g \subseteq g$.
- Note, however, that S is usually a preorder, therefore not anti-symmetric.

Optimizing inductive relations

Quite often, the orderings involved in optimization are **inductive** relations.

- Inductive orderings lead to recursive programs
- "Greedy algorithms" and "dynamic programming" studied in this way in the *Algebra of Programming* book (Bird and de Moor, 1997).
- Complexity of the approach puts many readers off (need for a tabular, power allegory; always transposing relations to powerset functions; ...)
- $R \upharpoonright S$ algebra **greatly simplifies** and generalizes the calculation of programs from such specifications.

Inductive relations

Example — inductive definition of the **prefix** relation:

$$x \sqsubseteq nil \;\Leftrightarrow\; x = nil$$
$$x \sqsubseteq cons(h,t) \;\Leftrightarrow\; x = nil \;\vee\; \langle \exists\, x' \;:\; x = cons(h,x') :\; x' \sqsubseteq t \rangle$$

The same in the pointfree style — unique solution of equation

$$\sqsubseteq \cdot [nil, cons] = [nil, nil \cup cons] \cdot (id + id \times \sqsubseteq) \tag{34}$$

Notation "folklore":

$$\sqsubseteq \;=\; (\!|[nil, nil \cup cons]|\!)$$

where $(\!|\cdots|\!)$ is termed the $\kappa\alpha\tau\alpha$ combinator.

$K\alpha\tau\alpha$s in general

In general, for F a polynomial functor (relator) and initial

$\mu\mathsf{F} \xleftarrow{in} \mathsf{F}(\mu\mathsf{F})$,

$$
\begin{tikzcd}
\mu\mathsf{F} \arrow[rr, bend left, "in^\circ"] \arrow[rr, phantom, "\cong"] \arrow[d, "(|R|)"'] & & \mathsf{F}(\mu\mathsf{F}) \arrow[ll, bend left, "in"] \arrow[d, "\mathsf{F}(|R|)"] \\
A & & \mathsf{F}\,A \arrow[ll, "R"]
\end{tikzcd}
$$

there is a unique solution to equation $X = R \cdot \mathsf{F}\,X \cdot in^\circ$ — thus universal property:

$$X = (|R|) \quad \Leftrightarrow \quad X = R \cdot \mathsf{F}\,X \cdot in^\circ \qquad (35)$$

(Read $(|R|)$ as "$\kappa\alpha\tau\alpha$ R".)

Introducing the $\kappa\alpha\tau\alpha$ combinator

Therefore, by Knaster-Tarski: $(\!|R|\!)$ is both **the least** prefix point

$$(\!|R|\!) \subseteq X \quad \Leftarrow \quad R \cdot \mathsf{F}\,X \cdot in^\circ \subseteq X \tag{36}$$

and **the greatest** postfix point:

$$X \subseteq (\!|R|\!) \quad \Leftarrow \quad X \subseteq R \cdot \mathsf{F}\,X \cdot in^\circ \tag{37}$$

Corollaries include **reflexion**,

$$(\!|in|\!) \quad = \quad id \tag{38}$$

and two forms of $\kappa\alpha\tau\alpha$-**fusion**:

$$S \cdot (\!|R|\!) \subseteq (\!|T|\!) \quad \Leftarrow \quad S \cdot R \subseteq T \cdot \mathsf{F}\,S \tag{39}$$

$$(\!|T|\!) \subseteq S \cdot (\!|R|\!) \quad \Leftarrow \quad T \cdot \mathsf{F}\,S \subseteq S \cdot R \tag{40}$$

Derived properties

Post-conditioning (make $T := \Phi \cdot R$ in (40) and simplify):

$$(\!|\Phi \cdot R|\!) \quad \subseteq \quad \Phi \cdot (\!|R|\!) \tag{41}$$

Dropping type checks:

$$(\!|R|\!) \subseteq S \cdot (\!|R|\!) \quad \Leftarrow \quad S \xleftarrow{R} \mathsf{F}\, S \tag{42}$$

(among many others)

"Greedy" theorem

My version of theorem 7.2 by Bird and de Moor (1997):

$$(\!|R \upharpoonright S|\!) \subseteq (\!|R|\!) \upharpoonright S \quad \Leftarrow \quad S^\circ \xleftarrow{R} \mathsf{F}\, S^\circ \tag{43}$$

for S transitive. In a diagram, where the side condition is depicted in dashed arrows:

$$\begin{tikzcd}
 & \mu\mathsf{F} \arrow[dl, bend right, "(\!|R \upharpoonright S|\!)"'] \arrow[dl, "(\!|R|\!) \upharpoonright S"'] \arrow[d, "(\!|R|\!)"'] \arrow[r, bend left, "in^\circ"] \arrow[r, phantom, "\cong"] & \mathsf{F}(\mu\mathsf{F}) \arrow[l, bend left, "in"] \arrow[d, "\mathsf{F}(\!|R|\!)"] \\
A & A \arrow[l, "S"'] & \mathsf{F}A \arrow[l, "R"'] \arrow[ll, bend left, "R \upharpoonright S"] \\
 & A \arrow[u, dashed, "S^\circ"] \arrow[ur, phantom, "\supseteq"] & \mathsf{F}A \arrow[l, dashed, "R"] \arrow[u, dashed, "\mathsf{F}S^\circ"']
\end{tikzcd}$$

Calculational proof

$$
\begin{array}{ll}
& (\!|R \upharpoonright S|\!) \;\subseteq\; (\!|R|\!) \upharpoonright S \\
\Leftrightarrow & \qquad \{ \text{ universal property of } (\upharpoonright) \text{ (13) } \} \\
& (\!|R \upharpoonright S|\!) \subseteq (\!|R|\!) \;\wedge\; (\!|R \upharpoonright S|\!) \cdot (\!|R|\!)^\circ \subseteq S \\
\Leftrightarrow & \qquad \{ \text{ monotonicity, since } X \upharpoonright Y \subseteq X \text{ in general } \} \\
& (\!|R \upharpoonright S|\!) \cdot (\!|R|\!)^\circ \subseteq S \\
\Leftrightarrow & \qquad \{ \text{ hylomorphisms: } (\!|S|\!) \cdot (\!|R|\!)^\circ = \langle \mu\, X \;::\; S \cdot \mathsf{F}\, X \cdot R^\circ \rangle \; \} \\
& \langle \mu\, X \;::\; (R \upharpoonright S) \cdot \mathsf{F}\, X \cdot R^\circ \rangle \subseteq S \\
\Leftarrow & \qquad \{ \text{ least (pre)fixpoint } \} \\
& (R \upharpoonright S) \cdot \mathsf{F}\, S \cdot R^\circ \subseteq S
\end{array}
$$

Calculational proof (closing)

$$
\begin{array}{cl}
 & (R \upharpoonright S) \cdot \mathsf{F}\, S \cdot R^\circ \subseteq S \\
\Leftarrow & \{ \text{ side-condition } S^\circ \xleftarrow{R} \mathsf{F}\, S^\circ \text{ ; converses ; monotonicity } \} \\
 & (R \upharpoonright S) \cdot R^\circ \cdot S \subseteq S \\
\Leftarrow & \{ \text{ since } R \upharpoonright S \subseteq S / R^\circ \} \\
 & (S / R^\circ) \cdot R^\circ \cdot S \subseteq S \\
\Leftarrow & \{ \text{ division cancellation (16) } \} \\
 & S \cdot S \subseteq S \\
\Leftarrow & \{ S \text{ assumed transitive } \} \\
 & \textsc{True}
\end{array}
$$

(Re-worked from (Bird and de Moor, 1997).)

Back to the beginning

Resuming what we were doing:

$$\underbrace{h \subseteq \Phi_p \cdot \sqsubseteq}_{\text{"easy"}} \;\wedge\; \underbrace{h \cdot (\Phi_p \cdot \sqsubseteq)^\circ \subseteq \sqsupseteq}_{\text{"hard"}}$$

$$\Leftrightarrow \qquad \{ \text{ introduce optimization combinator (13) } \}$$

$$h \subseteq (\Phi_p \cdot \sqsubseteq) \upharpoonright \sqsupseteq$$

Note that:

- $(\Phi_p \cdot \sqsubseteq) \upharpoonright \sqsupseteq$ is entire because h is so
- $(\Phi_p \cdot \sqsubseteq) \upharpoonright \sqsupseteq$ will be simple in case $\sqsupseteq$ is anti-symmetric.

Thus, for a partial order $\sqsubseteq$, the upper adjoint of the starting GC is

$$h = (\Phi_p \cdot \sqsubseteq) \upharpoonright \sqsupseteq \tag{44}$$

Calculational options

How do we calculate h ? Two ways:

1. Use the pointwise GC implicit in (44) — the one we started from — and use the pointwise properties of $\sqsubseteq$.
   - This was the method used in (Oliveira, 2010) for calculating a number of upper-adjoints, namely *take*.
   - better in forecasting properties of h than in implementing it.
2. Resort to (44) directly, using the "greedy" theorem.

Here is an example:

$$takewhile\ p \quad \subseteq \quad ((\Phi_p)^\star \cdot (\!|[nil, nil \cup cons]|\!)) \upharpoonright \geq_{length}$$

where $(\Phi_p)^\star$ is the "every element meets p" check on lists and

- $(\!|[nil, nil \cup cons]|\!)$ is the inductive definition of $\sqsubseteq$ on finite lists;
- $\geq_{length} = length^\circ \cdot \geq \cdot length$ is the "longer than" preorder.

"The longest prefix of a list is itself"

For economy of exposition, let us consider the more immediate

$$id \quad \subseteq \quad (\![nil, nil \cup cons]\!) \upharpoonright \geq_{length}$$

(="the longest prefix of a list is itself").

For the "greedy" theorem (43) to be of use, side condition

$$\geq^{\circ}_{length} \xleftarrow{[nil, nil \cup cons]} id + id \times \geq^{\circ}_{length}$$

must be checked beforehand. Noting that $\geq^{\circ}_{length} = \leq_{length}$, we have to check

$$[nil, (nil \cup cons) \cdot (id \times \leq_{length})] \quad \subseteq \quad \leq_{length} \cdot [nil, nil \cup cons]$$

"The longest prefix of a list is itself"

From basic properties of relational coproducts this unfolds into

$$nil \subseteq \leq_{length} \cdot nil$$
$$(nil \cup cons) \cdot (id \times \leq_{length}) \subseteq \leq_{length} \cdot (nil \cup cons)$$

which (since $\leq_{length}$ is a preorder) shrinks to monotonicity condition

$$cons \cdot (id \times \leq_{length}) \subseteq \leq_{length} \cdot cons$$

which trivially holds

$$length\ y \leq length\ x \quad \Rightarrow \quad length(cons(h, y) \leq length(cons(h, x))$$

since $length(cons(a, b)) = 1 + length\ b$.

"The longest prefix of a list is itself"

Thus we can rely on the "greedy" theorem (43):

$$
\begin{array}{cl}
 & id \subseteq (\![nil, nil \cup cons]\!) \upharpoonright \geq_{length} \\
\Leftarrow & \{ \text{ (43) followed by (30) ; } nil \upharpoonright \geq_{length} = nil \text{ } \} \\
 & id \subseteq (\![nil, (nil \cup cons) \upharpoonright \geq_{length}]\!) \\
\Leftrightarrow & \{ \text{ function competition (33), details omitted } \} \\
 & id \subseteq (\![nil, cons]\!) \\
\Leftrightarrow & \{ \text{ } \kappa\alpha\tau\alpha\text{-reflexion (38) } \} \\
 & id \subseteq id
\end{array}
$$

takewhile in brief

- The *takewhile* spec,

$$takewhile\ p \quad \subseteq \quad ((\Phi_p)^\star \cdot (\![nil, nil \cup cons]\!)) \upharpoonright \geq_{length}$$

  adds post-condition $(\Phi_p)^\star$ to what produced the identity function above.

- This is another inductive ("map"-like) relation, a coreflexive:

$$(\Phi_p)^\star \quad = \quad (\![nil, cons \cdot (\Phi_p \times id)]\!)$$

  which fuses with prefix $(\![nil, nil \cup cons]\!)$ — recall (41) — yielding

$$(\![nil, (nil \cup cons \cdot (\Phi_p \times id)) \upharpoonright \geq_{length}]\!)$$

takewhile in brief

Thus we meet a variant of function competition which leads to a familiar encoding,

$$(f \cup g \cdot \Phi_p) \upharpoonright S \quad = \quad p \rightarrow g \ , \ f$$

— under the side-conditions of (33) — and thus

$$\mathit{takewhile}\ p \quad = \quad (\![[\mathit{nil}, p \cdot \pi_1 \rightarrow \mathit{cons}\ ,\ \mathit{nil}]]\!)$$

which becomes

```
takewhile :: (a -> Bool) -> [a] -> [a]
takewhile p [] = []
takewhile p (h:t)
      | p h = h: takewhile p t
      | otherwise = []
```

in Haskell notation.

Winding up — related work

- The $R \upharpoonright S$ combinator corresponds to what Bird and de Moor (1997) write as $min\ S \cdot \Lambda R$ where $\mathcal{P}B \xleftarrow{\Lambda R} A$ (a function) is the powerset-transpose of relation $B \xleftarrow{R} A$ and $B \xleftarrow{min\ S} \mathcal{P}B$ computes the minimum of a set (if it exists) according to relation S.
- Currently re-working results of the book so as to check the calculational power of the combinator.
- Also trying to calculate far more complex functions, for instance the *shortest maximally-dense prefix function* (two superlatives!) studied by Mu and Curtis (2010).
- Functions of this kind arise in bioinformatics in finding sections of DNA dense with mutations. Read (Mu and Curtis, 2010).

Last but not least

Towards optimization of probabilistic, or stochastic systems — plan of the work is:

- Shift from relational algebra to linear algebra — cf. "matrices as arrows" (Macedo and Oliveira, 2010)
- Binary relations (Boolean matrices) give place to system behaviour models such as eg. Markov chains, etc
- (Blocked) linear algebra is pointfree "per se"
- Studying conditions for the extension

$$X \leq R \upharpoonright S \quad \Leftrightarrow \quad X \leq R \ \wedge \ X \cdot R^t \leq S$$

to make sense, where X, R, S are stochastic matrices.

References

R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997. C.A.R. Hoare, series editor.

M.A. Ferreira and J.N. Oliveira. Variations on an Alloy-centric tool-chain in verifying a journaled file system model. Technical Report DI-CCTC-10-07, DI/CCTC, University of Minho, Gualtar Campus, Braga, January 2010. Available from the authors' websites.

H.D. Macedo and J.N. Oliveira. Matrices as arrows! a biproduct approach to typed linear algebra, 2010. (Submitted to MPC'10).

Shin-Cheng Mu and S. Curtis. Functional pearl: Maximally dense segments, 2010. Draft: see http://www.iis.sinica.edu.tw/ scm/2010/functional-pearl-maximally-dense-segments/.

J.N. Oliveira. A Look at Program "Galculation", January 2010. Presentation at the IFIP WG 2.1 #65 Meeting.

P.F. Silva and J.N. Oliveira. 'Galculator': functional prototype of a Galois-connection based proof assistant. In *PPDP '08:*

Proceedings of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming, pages 44–55, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-117-0. doi: http://doi.acm.org/10.1145/1389449.1389456. .