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** $\frac{x}{y}$:

- 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* $\frac{x}{y}$?
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):

\[ x \div y = \langle \square z :: z \times y \leq x \rangle \]  

(1)

Second version (involved):

\[ z = x \div y \iff \langle \exists \ r : 0 \leq r < y : x = z \times y + r \rangle \]  

(2)

Third version (clever!):

\[ z \times y \leq x \iff z \leq x \div y \quad (y > 0) \]  

(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:

\[z \times y \leq x \Rightarrow z \leq x \div y \quad (y > 0)\]

(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 \textit{take} function on lists (longest prefix up-to specified length) is the upper adjoint of GC

\[
\text{len } y \leq n \land y \sqsubseteq x \iff y \sqsubseteq \text{take}(n, x)
\]

(Oliveira, 2010). Another GC,

\[
\langle \forall i : i \in \text{inds } y : p(y_i) \rangle \land y \sqsubseteq x \iff y \sqsubseteq \text{takewhile } p \ x
\]

specifies \textit{takewhile }p (longest prefix meeting condition \textit{p}).
Abstract pattern

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

\[ p \ y \land y \sqsubseteq x \iff y \sqsubseteq h \ x \quad (4) \]

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

\[ p \sq y \land y \sq x \iff y \sq h x \]

into (relational) equality

\[ \Phi_p \cdot \sq = \sq \cdot h \]

where “\( \cdot \)” means relational **composition** and \( \Phi_p \) denotes the **partial identity** which captures property \( p \). Details follow.
Relational composition and equality

**Composition:**

\[
B \xleftarrow{R} A \xleftarrow{S} C
\]

\[b(R \cdot S)c \iff \langle \exists a :: b R a \land a S c \rangle\] (7)

In case \(S\) is a function (say \(h\)):

\[b(R \cdot h)c \iff b R (h a)\] (8)

**Equality:**

\[R = S \iff \langle \forall b, a :: b R a \iff b S a \rangle\] (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:

$$a(R^\circ)b \iff b \, R \, a$$  \hspace{1cm} (10)

**Coreflexives:** binary relation encodings of unary predicates:

$$b \Phi_p a \iff b = a \land (p \, a)$$  \hspace{1cm} (11)

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

$$X \subseteq \Phi_p \iff X \subseteq id \land \langle \forall \, a : \, a \, X \, a : \, p \, a \rangle$$
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,

$$\Phi_p \subseteq \Phi_p$$

to mean the same as

$$\subseteq \cdot \Phi_p \subseteq \Phi_p \cdot \subseteq$$  \hspace{1cm} (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

$$x \subseteq y \land (p \ y) \Rightarrow p \ x$$

In other words: property \( p \) is **downward closed**.
Calculating with generic GC

\[ \Phi_p \cdot \sqsubseteq = \sqsubseteq \cdot h \]

\( \Leftrightarrow \quad \{ \text{anti-symmetry} \} \)

\[ \Phi_p \cdot \sqsubseteq \subseteq \sqsubseteq \cdot h \land \sqsubseteq \cdot h \subseteq \Phi_p \cdot \sqsubseteq \]

\( \Leftrightarrow \quad \{ h \subseteq \Phi_p \cdot \sqsubseteq \Leftrightarrow \sqsubseteq \cdot h \subseteq \Phi_p \cdot \sqsubseteq \text{ because } p \text{ is downward closed} \} \)

\[ \Phi_p \cdot \sqsubseteq \subseteq \sqsubseteq \cdot h \land h \subseteq \Phi_p \cdot \sqsubseteq \]

\( \Leftrightarrow \quad \{ \text{converses} ; \text{swap conjuncts} \} \)

\[ h \subseteq \Phi_p \cdot \sqsubseteq \land (\Phi_p \cdot \sqsubseteq) \circ \subseteq h^\circ \cdot \sqsubseteq \]

\( \Leftrightarrow \quad \{ \text{shunting on } h^\circ \} \)

\[ \underbrace{h \subseteq \Phi_p \cdot \sqsubseteq}_{\text{“easy”}} \land \underbrace{h \cdot (\Phi_p \cdot \sqsubseteq) \circ \subseteq \sqsubseteq}_{\text{“hard”}} \]
Calculating with generic GC

Comments:

- Easy part: \( h \subseteq \Phi_p \cdot \sqsubseteq \) — ensures \( h \) yielding approximations satisfying \( p \)
- Hard part: \( h \cdot (\Phi_p \cdot \sqsubseteq) \circ \subseteq \subseteq \) — 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,
define \( R \upharpoonright S \) satisfying universal property:

\[
X \subseteq R \upharpoonright S \iff X \subseteq R \land X \cdot R^\circ \subseteq S \quad (13)
\]

This is explained below with points (and words).
The "$R$ optimized by $S$" combinator

- The $\Leftarrow$ part of the given property

\[
X \subseteq R \upharpoonright S \; \Leftarrow \; X \subseteq R \land X \cdot R^\circ \subseteq S
\]

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 $b' X a \land b Ra$, then $b' S b$ holds ("$b'$ better than $b$").

- The same in a closed formula,

\[
R \upharpoonright S = \underbrace{R \cap S/R^\circ}_{\text{easy}} \; \underbrace{\ (14) \ \text{hard}}_{\text{relational division}}
\]

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

\[
X \cdot R \subseteq S \; \Leftrightarrow \; X \subseteq S/R
\]
Role of division (“hard” part)

With points:

\[ c(S / P)a \iff \langle \forall b : a P b : c S b \rangle \]

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

\[ b' R a \land \langle \forall b : b R a : b' S b \rangle \]

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 $R \uparrow S$

- 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 $R \uparrow S$ where $R$ is a data-structure:

\[
\begin{pmatrix}
\text{Mark} & \text{Student} \\
10 & John \\
11 & Mary \\
12 & John \\
15 & Arthur \\
\end{pmatrix} \uparrow \geq \ = \ 
\begin{pmatrix}
\text{Mark} & \text{Student} \\
11 & Mary \\
12 & John \\
15 & Arthur \\
\end{pmatrix}
\]

- Since then, I’ve been developing the algebra of $R \uparrow S$ on a “call by need” fashion.
Basic properties

Chaotic optimization:

\[ R \upharpoonright \top = R \quad (19) \]

Impossible optimization:

\[ R \upharpoonright \bot = \bot \quad (20) \]

Force determinism:

\( R \upharpoonright id \) = largest deterministic fragment of \( R \)  

Pre-condition fusion:

\( (R \upharpoonright S) \cdot \Phi = (R \cdot \Phi) \upharpoonright S \)  

(22)
Basic properties

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

\[
\begin{align*}
(R \upharpoonright S) \cdot f &= (R \cdot f) \upharpoonright S \\
(f \cdot S) \upharpoonright R &= f \cdot (S \upharpoonright R_f)
\end{align*}
\] (23)

Ensure simplicity (determinism):

\[\begin{array}{c}
R \upharpoonright S \text{ is simple } \iff S \text{ is anti-symmetric}
\end{array}\] (25)

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

\[R \upharpoonright S = R \iff \text{img } R \subseteq S\] (26)

Thus (functions)

\[f \upharpoonright S = f \iff S \text{ is reflexive}\] (27)
Basic properties

Union:

\[(R \cup S) \upharpoonright Q = (R \upharpoonright Q) \cap Q/S^\circ \cup (S \upharpoonright Q) \cap Q/R^\circ\]  \hspace{1cm} (28)

This has a number of corollaries, namely conditionals:

\[(P \rightarrow R, T) \upharpoonright S = P \rightarrow (R \upharpoonright S), (T \upharpoonright S)\]  \hspace{1cm} (29)

Disjoint union:

\[[R, S] \upharpoonright U = [R \upharpoonright U, S \upharpoonright U]\]  \hspace{1cm} (30)

where the \textit{junc} operator

\[[R, S] \triangleq R \cdot i_1^\circ \cup S \cdot i_2^\circ\]  \hspace{1cm} (31)

is associated to relational coproducts.
The “function competition” rule

A corollary of the union rule,

\[(f \cup g) \upharpoonright S = (f \cap S \cdot g) \cup (g \cap S \cdot f)\]  \hspace{1cm} (32)

since \(S/g = S \cdot g\). 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 $f$:

$$(f \cup g) \upharpoonright S = g \iff g \subseteq S \cdot f \land f \subseteq (S \cdot g \Rightarrow g)$$  

(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.
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:

\[
x \sqsubseteq \text{nil} \iff x = \text{nil}
\]
\[
x \sqsubseteq \text{cons}(h, t) \iff x = \text{nil} \lor \exists x' : x = \text{cons}(h, x') : x' \sqsubseteq t
\]

The same in the pointfree style — unique solution of equation

\[
\sqsubseteq \cdot [\text{nil}, \text{cons}] = [\text{nil}, \text{nil} \cup \text{cons}] \cdot (\text{id} + \text{id} \times \sqsubseteq)
\]  \hspace{1cm} (34)

Notation “folklore”:

\[
\sqsubseteq = ([\text{nil}, \text{nil} \cup \text{cons}])
\]

where \([\cdots]\) is termed the \(\kappa\alpha\tau\alpha\) combinator.
In general, for $F$ a polynomial functor (relator) and initial

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

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

$$X = (\left[R\right]) \iff X = R \cdot F X \cdot \text{in}^\circ$$ \hspace{1cm} (35)

(Read $(\left[R\right])$ as “καταR”.)
Introducing the $\kappa\alpha\tau\alpha$ combinator

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

\[
(\|R\|) \subseteq X \iff R \cdot F X \cdot in^\circ \subseteq X \tag{36}
\]

and the greatest postfix point:

\[
X \subseteq (\|R\|) \iff X \subseteq R \cdot F X \cdot in^\circ \tag{37}
\]

Corollaries include reflexion,

\[
(\|in\|) = id \tag{38}
\]

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

\[
S \cdot (\|R\|) \subseteq (\|T\|) \iff S \cdot R \subseteq T \cdot F S \tag{39}
\]
\[
(\|T\|) \subseteq S \cdot (\|R\|) \iff T \cdot F S \subseteq S \cdot R \tag{40}
\]
Derived properties

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

$$(\Phi \cdot R) \subseteq \Phi \cdot (R)$$  \hspace{1cm} (41)

Dropping type checks:

$$(R) \subseteq S \cdot (R) \iff S \leftarrow R F S$$  \hspace{1cm} (42)

(among many others)
"Greedy" theorem

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

\[(|R \upharpoonright S|) \subseteq (|R|) \upharpoonright S \iff S^\circ \leftarrow^R F S^\circ \quad (43)\]

for \(S\) transitive. In a diagram, where the side condition is depicted in dashed arrows:
Calculational proof

\((|R \upharpoonright S|) \subseteq (|R|) \upharpoonright S\)

\[\iff \{ \text{universal property of (\upharpoonright) (13)} \} \]

\((|R \upharpoonright S|) \subseteq (|R|) \wedge (|R \upharpoonright S|) \cdot (|R|)^\circ \subseteq S\)

\[\iff \{ \text{monotonicity, since } X \upharpoonright Y \subseteq X \text{ in general} \} \]

\((|R \upharpoonright S|) \cdot (|R|)^\circ \subseteq S\)

\[\iff \{ \text{hylomorphisms: } (|S|) \cdot (|R|)^\circ = \langle \mu X :: S \cdot F X \cdot R^\circ \rangle \} \]

\[\langle \mu X :: (R \upharpoonright S) \cdot F X \cdot R^\circ \rangle \subseteq S\]

\[\iff \{ \text{least (pre)fixpoint} \} \]

\((R \upharpoonright S) \cdot F S \cdot R^\circ \subseteq S\)
Calculational proof (closing)

\[(R \upharpoonright S) \cdot F S \cdot R^\circ \subseteq S\]

\[\iff \{ \text{side-condition } S^\circ \overset{R}{\leftarrow} F S^\circ \text{ ; converses ; monotonicity} \} \]

\[(R \upharpoonright S) \cdot R^\circ \cdot S \subseteq S\]

\[\iff \{ \text{since } R \upharpoonright S \subseteq S / R^\circ \} \]

\[(S / R^\circ) \cdot R^\circ \cdot S \subseteq S\]

\[\iff \{ \text{division cancellation (16)} \} \]

\[S \cdot S \subseteq S\]

\[\iff \{ S \text{ assumed transitive} \} \]

\[\text{TRUE}\]

(Re-worked from (Bird and de Moor, 1997).)
Resuming what we were doing:

\[ h \subseteq \Phi_p \cdot \sqsubseteq \quad \land \quad h \cdot (\Phi_p \cdot \sqsubseteq)^\circ \subseteq \sqsupseteq \]

\begin{align*}
\text{“easy”} & \quad \quad \text{“hard”} \\
\Leftrightarrow & \quad \{ \text{introduce optimization combinator (13)} \} \\
& \quad \quad \quad \quad h \subseteq (\Phi_p \cdot \sqsubseteq) \uparrow \sqsupseteq
\end{align*}

Note that:

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

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

\[ h = (\Phi_p \cdot \sqsubseteq) \uparrow \sqsupseteq \quad (44) \]
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 $\text{take}$.
   - better in forecasting properties of $h$ than in implementing it.

2. Resort to (44) directly, using the “greedy” theorem.

Here is an example:

$$\text{takewhile } p \subseteq ((\Phi_p)^* \cdot ([\text{nil}, \text{nil} \cup \text{cons}])] \uparrow \geq \text{length}$$

where $(\Phi_p)^*$ is the “every element meets $p$” check on lists and
- $([\text{nil}, \text{nil} \cup \text{cons}])]$ is the inductive definition of $\sqsubseteq$ on finite lists;
- $\geq \text{length} = \text{length}^\circ \cdot \geq \cdot \text{length}$ is the “longer than” preorder.
“The longest prefix of a list is itself”

For economy of exposition, let us consider the more immediate

\[ id \subseteq ([nil, nil \cup cons]) \uparrow \geq_{\text{length}} \]

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

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

\[ \geq_{\text{length}} \circ \geq_{\text{length}} \subseteq \leq_{\text{length}} \]

must be checked beforehand. Noting that \( \geq_{\text{length}} = \leq_{\text{length}} \), we have to check

\[ [\text{nil}, (\text{nil} \cup \text{cons}) \cdot (id \times \leq_{\text{length}})] \subseteq \leq_{\text{length}} \cdot [\text{nil}, \text{nil} \cup \text{cons}] \]
“The longest prefix of a list is itself”

From basic properties of relational coproducts this unfolds into

\[ \text{nil} \subseteq \leq_{\text{length}} \cdot \text{nil} \]

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

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

\[ \text{cons} \cdot (\text{id} \times \leq_{\text{length}}) \subseteq \leq_{\text{length}} \cdot \text{cons} \]

which trivially holds

\[ \text{length } y \leq \text{length } x \Rightarrow \text{length}(\text{cons}(h, y)) \leq \text{length}(\text{cons}(h, x)) \]

since \( \text{length}(\text{cons}(a, b)) = 1 + \text{length } b \).
“The longest prefix of a list is itself”

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

\[
id \subseteq ([nil, \text{nil} \cup \text{cons}]) \uparrow \geq \text{length}
\]

\[
\Leftrightarrow \quad \{ \text{(43) followed by (30); nil} \uparrow \geq \text{length} = \text{nil} \}
\]

\[
id \subseteq ([nil, (\text{nil} \cup \text{cons})] \uparrow \geq \text{length})]
\]

\[
\Leftrightarrow \quad \{ \text{function competition (33), details omitted} \}
\]

\[
id \subseteq ([nil, \text{cons}])
\]

\[
\Leftrightarrow \quad \{ \kappa \alpha \tau \alpha\text{-reflexion (38)} \}
\]

\[
id \subseteq id
\]
takewhile in brief

• The takewhile spec,

\[
\text{takewhile } p \subseteq ((\Phi_p)^* \cdot ([\text{nil}, \text{nil } \cup \text{cons}])) \uparrow \geq \text{length}
\]

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

• This is another inductive (“map”-like) relation, a coreflexive:

\[
(\Phi_p)^* = ([\text{nil}, \text{cons} \cdot (\Phi_p \times \text{id})])
\]

which fuses with prefix \(([\text{nil}, \text{nil } \cup \text{cons}])\) — recall (41) — yielding

\[
([\text{nil}, (\text{nil } \cup \text{cons} \cdot (\Phi_p \times \text{id})) \uparrow \geq \text{length}])
\]
**takewhile in brief**

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

\[(f \cup g \cdot \Phi_p) \restriction S = p \to g, f\]

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

\[
takewhile p = ([\text{nil}, p \cdot \pi_1 \to \text{cons}, \text{nil}])
\]

which becomes

\[
takewhile :: (a -> \text{Bool}) \to [a] -> [a]
\]

\[
takewhile p [] = []
\]

\[
takewhile p (h:t)
\]

\[
\quad | \ p \ h = h : \ \text{takewhile} \ p \ t
\]

\[
\quad | \ \text{otherwise} = []
\]

in Haskell notation.
Winding up — related work

- The $R \upharpoonright S$ combinator corresponds to what Bird and de Moor (1997) write as $\text{min } S \cdot \Lambda R$ where $\wp B \leftarrow^{\Lambda R} A$ (a function) is the powerset-transpose of relation $B \leftarrow^{R} A$ and $B \leftarrow^{\text{min } S} \wp B$ 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).
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

\[ X \leq R \upharpoonright S \iff X \leq R \land X \cdot R^t \leq S \]

to make sense, where \( X, R, S \) are stochastic matrices.
References


