Emulating Linear Types in Haskell

Wolfgang Jeltsch

TTÜ Küberneetika Instituut

Teooriaseminar
February 16, 2012

Linear logic

- useful for reasoning about resources
- each hypothesis must be used exactly once
- very different from the normal understanding of logic
- classical and intuitionistic variant
- in this talk, only intuitionistic linear logic

Linear logic formulas

- language:

$$F ::= F \otimes F \mid 1 \mid F \mathbin{\&} F \mid \top \mid F \oplus F \mid 0 \mid F \multimap F \mid {!F}$$

- meanings:

$\alpha \otimes \beta$	α and β hold simultaneously
1	nothing holds
$\alpha \mathbin{\&} \beta$	α and β hold (not necessarily simultaneously)
$\top$	tautology
$\alpha \oplus \beta$	α or β holds
0	absurdity
$\alpha \multimap \beta$	if α holds in addition, then β holds
$!\alpha$	α holds arbitrarily often

Linear logic example

- atomic propositions:

e	I have one euro.
$s/p/i$	I get a soup/a pancake/an icecream.

- derived propositions:
  - For four euros, I get a soup and a pancake:
    $$e \otimes e \otimes e \otimes e \multimap s \otimes p$$
  - For two euros, I get a soup or a pancake (my choice):
    $$e \otimes e \multimap s \mathbin{\&} p$$
  - For two euros, I get a pancake or an icecream (cafeteria's choice):
    $$e \otimes e \multimap p \oplus i$$
  - I am the central bank:
    $$!e$$

Comparison of the two conjunctions

- these propositions hold:

$$\alpha \multimap \alpha \mathbin{\&} \alpha$$
$$\alpha \mathbin{\&} \beta \multimap \alpha$$
$$\alpha \mathbin{\&} \beta \multimap \beta$$

- these do not hold in general:

$$\alpha \multimap \alpha \otimes \alpha$$
$$\alpha \otimes \beta \multimap \alpha$$
$$\alpha \otimes \beta \multimap \beta$$

Linear λ-calculus

- the Curry–Howard analog of intuitionistic linear logic
- values have to be used exactly once:
  - a value can represent the current state of an object
  - changes to the state (destructive updates) expressible as pure functions
- some functions with destructive updates:
  - array update:

$$\mathit{idx} \otimes \mathit{el} \otimes \mathit{Array}\ \mathit{idx}\ \mathit{el} \multimap \mathit{Array}\ \mathit{idx}\ \mathit{el}$$

  - opening a file:

$$\mathit{FileName} \otimes \mathit{World} \multimap \mathit{File} \otimes \mathit{World}$$

  - writing to an opened file:

$$\mathit{String} \otimes \mathit{File} \multimap \mathit{File}$$

  - closing a file:

$$\mathit{File} \otimes \mathit{World} \multimap \mathit{World}$$

Linearity in functional programming languages

- variant of linear types implemented in Clean (uniqueness types)
- no direct support for anything like this in Haskell:
  - ability to duplicate and destroy values is present by default
  - seems impossible to emulate linear types under these circumstances
  - but emulation is possible nevertheless

Products and coproducts

- bicartesian closed categories (BCCCs) as models of intuitionistic logic:
  - finite products for $\wedge$ and $\top$
  - finite coproducts for $\vee$ and $\bot$
  - exponentials for $\to$
- finite products and coproducts also used in models of intuitionistic linear logic:
  - finite producs for $\&$ and $\top$
  - finite coproducts for $\oplus$ and 0
- seems strange that $\wedge$ and $\&$ are modelled by the same construction, although they denote different things
- however, analogous propositions hold for $\wedge$ and $\&$:

$$\alpha \multimap \alpha \mathbin{\&} \alpha \qquad\qquad \alpha \to \alpha \wedge \alpha$$

$$\alpha \mathbin{\&} \beta \multimap \alpha \qquad\qquad \alpha \wedge \beta \to \alpha$$

$$\alpha \mathbin{\&} \beta \multimap \beta \qquad\qquad \alpha \wedge \beta \to \beta$$

Structure for $\otimes$, 1 and $\multimap$

- $\otimes$ and 1 modelled by a monoidal category structure:
  - $\otimes$ is associative and commutative
  - 1 is its neutral element
  - nothing more
- monoidal closed category for also modelling $\multimap$:
  - we have a natural transformation e with

    $$e_{A,B} : (A \multimap B) \otimes A \to B$$

    and an isomorphism

    $$\Lambda : \mathrm{Hom}(C \otimes A, B) \cong \mathrm{Hom}(C, A \multimap B)$$

    that fulfill certain conditions
  - corresponds to the definition of exponentials with $\times$ replaced by $\otimes$

Using products and sums

- finite products and sums in Haskell can be modelled by finited products and coproducts in category theory
- thus we can use products and sums for encoding $\&$, $\top$, $\oplus$, and 0
- algebraic data types can be used
- (x, y) now represents two possible resources of which we have to use exactly one
- framework has to make sure that we use exactly one
- duplication and disposal of values is possible, but intuition is different:

$\lambda x \to (x, x)$	if we have x, we can choose between x and x
$\lambda(x, y) \to x$	if we have the choice between x and y, we can choose x

Encoding $\otimes$, 1, and $\multimap$

- use (,), (), and $\to$ internally:

$$\begin{aligned}
&\textbf{newtype}\ \mathit{Blank} &&= \mathit{LinUnit}\ && ()\\
&\textbf{newtype}\ \alpha \otimes \beta &&= \mathit{LinPair}\ && (\alpha, \beta)\\
&\textbf{newtype}\ \alpha \multimap \beta &&= \mathit{LinFunction}\ && (\alpha \to \beta)
\end{aligned}$$

- export only operations from categorical models:

$$\begin{aligned}
\mathit{bimap} &:: (\alpha \to \alpha') \to (\beta \to \beta') \to (\alpha \otimes \beta \to \alpha' \otimes \beta')\\
\mathit{assoc} &:: (\alpha \otimes \beta) \otimes \gamma \to \alpha \otimes (\beta \otimes \gamma)\\
\mathit{drop}_1 &:: \mathit{Blank} \otimes \alpha \to \alpha\\
\mathit{drop}_2 &:: \alpha \otimes \mathit{Blank} \to \alpha\\
\mathit{swap} &:: \alpha \otimes \beta \to \beta \otimes \alpha\\
\mathit{apply} &:: (\alpha \multimap \beta) \otimes \alpha \to \beta\\
\mathit{curry} &:: (\gamma \otimes \alpha \to \beta) \to (\gamma \to \alpha \multimap \beta)
\end{aligned}$$

  and the inverses of *assoc*, drop_1, and drop_2

A problem with tensorial strength

- as shown last week, Haskell gives us tensorial strength automatically
- reason is that $\rightarrow$ is used for both morphisms and functions on morphisms
- the latter are thus morphisms themselves
- example of unsafe operator that can be derived from tensorial strength:

$$\lambda p \rightarrow \mathit{bimap}\ (\mathit{const}\ p)\ \mathit{id}\ p :: \alpha \otimes \beta \rightarrow (\alpha \otimes \beta) \otimes \beta$$

A solution using the Q-functor

- last week's talk introduced applicative functor Q for ensuring start time consistency in FRP
- technique can be generalized to work with other extensions of BCCCs
- if type α is modelled by object A, then $Q\ \alpha$ corresponds to $\mathrm{Hom}(1, A)$, where 1 is the initial object of the category
- therefore if α and β are modelled by A and B, $Q\ (\alpha \to \beta)$ corresponds to $\mathrm{Hom}(1, B^A) \cong \mathrm{Hom}(A, B)$
- represent morphisms from A to B by values of type $Q\ (\alpha \to \beta)$
- illegal values can only be constructed under at least two layers of Q
- make sure that values under two Q-layers are not used

Arrow instead of applicative functor

- use an arrow $\mapsto$ instead of the Q-functor:
  - makes more sense
  - seems easier to use
- if α and β are modelled by objects A and B, then $\alpha \mapsto \beta$ corresponds to $\mathrm{Hom}(A, B)$
- making $\mapsto$ an instance of the *Arrow* and *ArrowChoice* classes makes the following operations available:
  - transformation from $\rightarrow$ to $\mapsto$:
    $$\mathit{arr} :: (\alpha \rightarrow \beta) \rightarrow (\alpha \mapsto \beta)$$
  - composition of morphisms:
    $$(\ggg) :: (\alpha \mapsto \beta) \rightarrow (\beta \mapsto \gamma) \rightarrow (\alpha \mapsto \gamma)$$
  - bifunctor applications for products and sums:
    $$\begin{aligned} (***) &:: (\alpha \mapsto \alpha') \rightarrow (\beta \mapsto \beta') \rightarrow \\ &\quad ((\alpha, \beta) \mapsto (\alpha', \beta')) \\ (+++) &:: (\alpha \mapsto \alpha') \rightarrow (\beta \mapsto \beta') \rightarrow \\ &\quad (\mathit{Either}\ \alpha\ \beta \mapsto \mathit{Either}\ \alpha'\ \beta') \end{aligned}$$

Encoding further operations

- operations for dealing with $\otimes$, *Blank*, and $\multimap$:

$$
\begin{aligned}
\mathit{bimap} &:: (\alpha \mapsto \alpha') \to (\beta \mapsto \beta') \to (\alpha \otimes \beta \mapsto \alpha' \otimes \beta') \\
\mathit{assoc} &:: (\alpha \otimes \beta) \otimes \gamma \mapsto \alpha \otimes (\beta \otimes \gamma) \\
\mathit{drop}_1 &:: \mathit{Blank} \otimes \alpha \mapsto \alpha \\
\mathit{drop}_2 &:: \alpha \otimes \mathit{Blank} \mapsto \alpha \\
\mathit{swap} &:: \alpha \otimes \beta \mapsto \beta \otimes \alpha \\
\mathit{apply} &:: (\alpha \multimap \beta) \otimes \alpha \mapsto \beta \\
\mathit{curry} &:: (\gamma \otimes \alpha \mapsto \beta) \to (\gamma \mapsto \alpha \multimap \beta)
\end{aligned}
$$

  and the inverses of *assoc*, drop_1, and drop_2
- a variant of *curry* for (,) and $\to$:

$$
\mathit{curry} :: ((\gamma, \alpha) \mapsto \beta) \to (\gamma \mapsto \alpha \to \beta)
$$

Purity is not enough

- arrows used for computations with some effect
- effects are the controversial thing that must be safely encapsulated
- so *arr* is perhaps the most uncontroversial arrow operation, as it only makes effectless computations available
- however, we do not want to encapsulate effects
- we want even less than ordinary pure computations, as we do not want tensorial strength
- so *arr* is actually controversial in our case

How we can ensure consistency

- analogy to Q:
  - illegal values can occur
  - but only under at least two layers of $\mapsto$
- for example, this is possible:

$$\alpha \mapsto (() \mapsto \alpha \otimes \alpha)$$

- but this one is not:

$$\alpha \mapsto \alpha \otimes \alpha$$

- this is possible, but unproblematic, as we cannot construct resource values out of nothing:

$$\alpha \to (() \mapsto \alpha \otimes \alpha)$$

- make sure that values under two $\mapsto$-layers are not used

Conclusions and outlook

- the Curry–Howard analog to intuitionistic linear logic can be encoded in Haskell
- enables us to deal with stateful computations in a more functional way
- ongoing effort to combine this with FRP
- possible application:

  purely functional programming of GUIs
  with highly dynamic structure
- experimental Haskell code in the following darcs repositories:
  - `http://darcs.wolfgang.jeltsch.info/haskell/categorical-computing/main`
  - `http://darcs.wolfgang.jeltsch.info/haskell/linear/main`
  - `http://darcs.grapefruit-project.org/grapefruit-frp/main`