An introduction to linear & modal types

Dominic Orchard
http://dorchard.co.uk
http://github.com/dorchard/granule
University of Kent

Motivation

UnsafeFiles.hs

Today

- Learn about linear types
- Learn how a type system is formally specified
  - Specifically: linear types for the *lambda calculus*
- See examples of linear programs in Haskell and Granule
  - Prequel to

    Granule: A language for fine-grained reasoning via graded modal types

Linear lambda calculus

Syntax

Lambda calculus syntax

$$t, t' ::= x \mid \lambda x.t \mid t\ t'$$

Syntax

variables | functions | function application

`t, t'` ::=	`x`	`\x -> t`	`t t'`	Haskell
`t, t'` ::=	`x`	`fun (x) -> t`	`t t'`	OCaml
`t, t'` ::=	`X`	`fun (X) => t end`	`t(t')`	Erlang
`t, t'` ::=	`x`	`x -> t`	`t.apply(t)`	Java

Typing rules (linear)

Typing syntax and relation

Church syntax adds a type "signature"

$$t ::= x \mid \lambda(x : A).t \mid t\ t$$

Type syntax $A, B ::= A \to B$ $\mid \texttt{Int} \mid \texttt{Bool} \mid \ldots$

cf Haskell: **`t -> t'`**

In a full language we'd want more…

Typing lets us relate expressions to types, e.g.

$$\lambda(x : A).x : A \to A$$

Cf.:

```
id :: a -> a
id = \x -> x
```

Quick exercise:

Q: What is the type of this lambda term?

$$\lambda(x : A).\lambda(y : B).x$$

A: $\lambda(x : A).\lambda(y : B).x : A \to (B \to A)$

Cf.:

```
const :: a -> b -> a
const x y = x
```

Q: What is the type of this lambda term?

$$\lambda(x : A).y$$

A: $\textit{It depends!}$

Typing syntax and relation

Typing *judgement* with *assumptions* about variable types

$$y : B \vdash \lambda(x : A).y : A \to B$$

Assumptions **Term** **Type**

Syntax of assumptions

$$\Gamma ::= \Gamma, x : A \mid \emptyset$$

Typing *judgement* form: $\Gamma \vdash t : A$

Typing rules

Defined inductively

Base case: $\dfrac{}{\textbf{conclusions}}$

Inductive step: $\dfrac{\textbf{premises (inductive hypotheses)}}{\textbf{conclusions}}$

$$\text{var}\ \frac{}{x : A \vdash x : A}$$

A term which is just one variable, has just one assumption

$$\text{abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

Binding free variables

$$\text{app}\ \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash t' : A}{\Gamma, \Delta \vdash t\,t' : B}$$

Two sub terms have **different** contexts of assumptions

Example

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,\text{var} \qquad \dfrac{}{y : A \to B \vdash y : A \to B}\,\text{var}}{x : A, y : A \to B \vdash y\ x : B}\,\text{app}}{x : A \vdash \lambda(y : A \to B).y\ x : (A \to B) \to B}\,\text{abs}}{\emptyset \vdash \lambda(x : A).\lambda(y : A \to B).y\ x : A \to ((A \to B) \to B)}\,\text{abs}
$$

Note that (abs) takes the "first" assumption:

$$\text{abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

What if we want to lambda bind y in the following?

$$???\ \frac{y : A, x : B \vdash t : A'}{x : B \vdash \lambda(y : A).t : A \to A'}$$ ❌

We also have the "exchange" rule:

$$\text{exchange}\ \frac{\Gamma, x : A, y : B, \Gamma' \vdash t : A}{\Gamma, y : B, x : A, \Gamma' \vdash t : A}$$

Now we can do:

$$\text{abs}\ \frac{\text{exchange}\ \dfrac{y : A, x : B \vdash t : A'}{x : B, y : A \vdash t : A'}}{x : B \vdash \lambda(y : A).t : A \to A'}$$ ✔

Non examples:

Can't use var rule

$$
\cfrac{\cfrac{\cfrac{}{x : A, y : B \vdash x : A}\,???}{x : A \vdash \lambda(y : B).\, x : B \to A}\,\text{abs}}{\emptyset \vdash \lambda(x : A).\, \lambda(y : B).\, x : A \to (B \to A)}\,\text{abs}
$$

Ignoring variable y is dissallowed

Linear types
in
GHC Haskell
(development branch)

Linear types in Granule

(and solving the unsafe files problem)

Simple typing
(the usual state of affairs…)

Simple typing = Linear typing + weakening + contraction

Linear lambda calculus typing

$$\text{var}\ \frac{}{x : A \vdash x : A}$$

$$\text{abs}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}$$

$$\text{app}\ \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash t' : A}{\Gamma, \Delta \vdash t\, t' : B}$$

$$\text{exchange}\ \frac{\Gamma, x : A, y : B, \Gamma' \vdash t : A}{\Gamma, y : B, x : A, \Gamma' \vdash t : A}$$

Add irrelevant assumptions

$$\text{weaken}\ \frac{\Gamma \vdash t : A}{\Gamma, x : A' \vdash t : A}$$

Reuse variables

$$\text{contract}\ \frac{\Gamma, y : A', z : A' \vdash t : A}{\Gamma, x : A' \vdash t[x/z][x/y] : A}$$

Weakening:

Couldn't do this in the linear system

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,\text{var}}{x : A, y : B \vdash x : A}\,\textbf{weaken}}{\dfrac{x : A \vdash \lambda(y : B).\, x : B \to A}{\emptyset \vdash \lambda(x : A).\, \lambda(y : B).\, x : A \to (B \to A)}\,\text{abs}}\,\text{abs}$$

Ignoring variable y

Contraction:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\,\text{var} \quad \dfrac{}{y : A \vdash y : A}\,\text{var}}{x : A, y : A \vdash (x, y) : (A, A)}\,\text{pair}}{z : A \vdash (z, z) : (A, A)}\,\text{contract}}{\emptyset \vdash \lambda(z : A).(z, z) : A \to (A, A)}\,\text{abs}$$

Duplicating variable z

$\Box$ modality — use any number of times (or !)

linear types — use exactly once

Non-linearity modality

Typing syntax and relation

Extend syntax of types and typing assumptions

$$A, B ::= A \to B \mid \Box A$$

Non-linear value of type A

$$\Gamma ::= \Gamma, x : A \mid \Gamma, x : \boxdot A \mid \emptyset$$

Non-linear variable x of type A

(var), (abs), (app) stay the same…

…but we add <u>weakening</u> and <u>contraction</u> for $\boxdot$ assumptions

$$\text{weaken}\ \frac{\Gamma \vdash t : A}{\Gamma, x : \boxdot A' \vdash t : A} \qquad \text{contract}\ \frac{\Gamma, y : \boxdot A', z : \boxdot A' \vdash t : A}{\Gamma, x : \boxdot A' \vdash t[x/z][x/y] : A}$$

…and syntax + rules for working with non-linearity

Shift linear variable
to non-linear:
(derelection)

$$\text{der}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma, x : \boxdot A \vdash t : B}$$

Non-linear results
require non-linear variables
(promotion)

$$\text{pr}\ \frac{\boxdot\Gamma \vdash t : B}{\boxdot\Gamma \vdash |t| : \Box B}$$

Composition (substitution) of non-linear value
into non-linear variable

$$\text{let}\Box\ \frac{\Gamma \vdash t_1 : \Box A \qquad \Delta, x : \boxdot A \vdash t_2 : B}{\Gamma + \Delta \vdash \mathbf{let}\ |x| = t_1\ \mathbf{in}\ t_2 : B}$$

Non-linearity modality in Granule

(called `Box`)

Modal types (in general)

Modal logic - possibility & necessity

$\Box A \ = \ A$ is "necessarily" true, A true in all worlds

Previously, $\Box A$ meant A is always available

$\Diamond A \ = \ A$ "possibly" true, A true in some worlds

Its traditional logical structure correspond to a **monad**

Natural deduction possibility

$$\text{intro}\ \frac{\Gamma \vdash A}{\Gamma \vdash \lozenge A} \qquad\qquad \text{cut}\ \frac{\Gamma \vdash \lozenge A \quad \Gamma, A \vdash \lozenge B}{\Gamma \vdash \lozenge B}$$

Typed programs: monads!

$$\text{intro}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash \mathbf{return}\ t : \lozenge A} \qquad\qquad \text{cut}\ \frac{\Gamma \vdash t : \lozenge A \quad \Gamma, x : A \vdash t' : \lozenge B}{\Gamma \vdash \mathbf{do}\ x \leftarrow t; t' : \lozenge B}$$

Today

- Learn how a type system is formally specified
    - Specifically: linear types for the *lambda calculus*
- Learn about some modal types
- See examples of linear programs in Haskell and Granule
    - Prequel to

      Granule: A language for fine-grained reasoning via graded modal types

Extra slides

Alternate formulation in part B

We will not use $\text{contract}\ \dfrac{\Gamma, y : \boxdot A', z : \boxdot A' \vdash t : A}{\Gamma, x : \boxdot A' \vdash t[x/z][x/y] : A}$

but make contraction implicit in rules with multiple sub terms

$$\text{app}\ \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash t' : A}{\Gamma, \Delta \vdash t\, t' : B}$$

becomes

$$\text{app}\ \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash t' : A}{\Gamma + \Delta \vdash t\, t' : B}$$

where

$$(\Gamma, x : A) + (\Delta, x : A) \quad \textit{not defined}$$
$$(\Gamma, x : \boxdot A) + (\Delta, x : \boxdot A) = (\Gamma + \Delta), x : \boxdot A$$