Software System Design and Implementation

Curry Howard Correspondence
(Curry Howard Isomorphism)

The University of New South Wales
School of Computer Science and Engineering
Sydney, Australia

Let's go back in time

- Different, equivalent models of computation to address Hilbert's Entscheidungsproblem
  - Lambda-calculus (Church)
  - Recursive functions (Gödel)
  - Turing machine

The (untyped) lambda calculus

- Functions can be applied to themselves:

```
λf. f f
```

- As a result, we can have non-terminating reduction sequences:

```
(λf. f f)(λf. f f)
```
$$\longrightarrow_\beta$$
```
(λf. f f)(λf. f f)
```
$$\longrightarrow_\beta$$
```
(λf. f f)(λf. f f)
```
…

Church's Simply Typed Lambda Calculus

- For the presentation, we add the following functions & data constructors to the lambda calculus as short hand
  - `( , )`: like the pair data constructor in Haskell
  - `fst, snd:` like Haskell fst and snd
  - `left, right`: like Left and Right of the Either type
  - `case:` similar to `case` in Haskell, but restricted to Either type

```
case x f g ≃ case x of
                 Left a  -> f a
                 Right b -> g b
```

Church's Simply Typed Lambda Calculus

- Can be encoded in the lambda-calculus

```
(,) = λa. λb. λf. f a b
fst = λa. λb. a
snd = λa. λb. b
```

```
Right = λa. λf. λg. f a
Left  = λa. λf. λg. g a
case  = λa. λf. λg. a f g
```

Church's Simply Typed Lambda Calculus

$$\frac{\texttt{M :: A} \qquad \texttt{N :: B}}{\texttt{(M, N) :: A * B}}$$

read as:
if you can derive
M :: A and N :: B
then
(M,N) :: A * B
is derivable

$$\frac{\texttt{M :: A * B}}{\texttt{fst M :: A}} \qquad \frac{\texttt{M :: A * B}}{\texttt{snd M :: B}}$$

Church's Simply Typed Lambda Calculus

$$\frac{\texttt{M :: A}}{\texttt{left M :: A + B}} \qquad \frac{\texttt{M :: B}}{\texttt{right M :: A + B}}$$

$$\frac{\texttt{M :: A + B} \quad \texttt{K :: A} \rightarrow \texttt{C} \quad \texttt{H:: B} \rightarrow \texttt{C}}{\texttt{case M K H ::C}}$$

Church's Simply Typed Lambda Calculus

$$\frac{\begin{array}{c}[x :: A] \\ \vdots \\ M :: B\end{array}}{\lambda x.\ M :: A \to B}$$

read as:
if we can derive M :: B
from the assumption x :: A
then
λx.M :: A → B
is derivable

$$\frac{\lambda x.\ M :: A \to B \qquad N :: A}{(\lambda x.M)\ N :: B}$$

Church's Simply Typed Lambda Calculus

- The simply typed lambda calculus doesn't have general recursion:

λf. f f   can't be typed!

- For all well-typed terms
  - reduction terminates
  - reduction does not change the type of a term

- Note: the Y-combinator can be added to make it turing-complete again:

```
Y = λf. (λx. f (x x))(λx. f(x x))

Y f = f (Y f)

Y :: (A → A) → A
```

Natural Deduction

- At around the same time, Gerhard Gentzen was working on the logic aspects of the Hilbert program: establishing the consistency of various logics

- Gentzen introduced two new formulations of logic, which remain the main ones used to this day:

  - Sequent calculus

  - Natural deduction

Natural Deduction

- Rules come in pairs: introduction and elimination

$$\frac{A \qquad B}{A \land B}\ \land\text{-I} \qquad\qquad \frac{A \land B}{A}\ \land\text{-E1} \qquad\qquad \frac{A \land B}{B}\ \land\text{-E2}$$

$$\frac{A \land B}{B \land A} \qquad\qquad \frac{\land\text{-E2}\ \dfrac{A \land B}{B} \qquad \dfrac{A \land B}{A}\ \land\text{-E1}}{B \land A}\ \land\text{-I}$$

Natural Deduction

- ∨-introduction and elimination

$$\frac{A}{A \vee B}\ \vee\text{-I1} \qquad \frac{B}{A \vee B}\ \vee\text{-I2}$$

$$\frac{A \vee B \qquad A \Rightarrow C \qquad B \Rightarrow C}{C}\ \vee\text{-E}$$

Natural Deduction

- Implication

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\ \Rightarrow\text{-I} \qquad\qquad \frac{A \Rightarrow B \quad\quad A}{B}\ \Rightarrow\text{-E}$$

Proof normalisation

- Gentzen observed that all proofs for propositional logic can be normalised, so they only contain sub formulas of premise or conclusion:

$$\dfrac{\wedge\text{-E2}\ \dfrac{A \wedge B}{B} \qquad \dfrac{A \wedge B}{A}\ \wedge\text{-E1}}{B \wedge A}\ \wedge\text{-I}$$

$$\dfrac{\dfrac{A \wedge B}{B} \qquad \dfrac{\dfrac{\dfrac{A \wedge B}{A} \qquad \dfrac{A \wedge B}{A}}{A \wedge A}}{A}}{B \wedge A}$$

Curry Howard Isomorphism

- In 1934, Curry observed a relationship between logic implication A ⇒ B and function types A ➔ B
- Howard realised in 1969 that this connection is much deeper

Curry Howard Isomorphism

$$\frac{M :: A \qquad N :: B}{(M, N) :: A * B}$$

$$\frac{A \qquad B}{A \wedge B}$$

$$\frac{M :: A * B}{\mathtt{fst}\ M :: A}$$

$$\frac{A \wedge B}{A}$$

$$\frac{M :: A * B}{\mathtt{snd}\ M :: B}$$

$$\frac{A \wedge B}{B}$$

$$\frac{\texttt{M :: }\color{green}{\texttt{A}}}{\texttt{left M :: }\color{green}{\texttt{A + B}}} \qquad \frac{\texttt{M :: }\color{green}{\texttt{B}}}{\texttt{right M :: }\color{green}{\texttt{A + B}}}$$

$$\frac{A}{A \vee B} \qquad \frac{B}{A \vee B}$$

$$\frac{\texttt{M :: }\color{green}{\texttt{A + B}} \quad \texttt{K :: }\color{green}{\texttt{A → C}} \quad \texttt{H:: }\color{green}{\texttt{B → C}}}{\texttt{case M K H :: }\color{green}{\texttt{C}}}$$

$$\frac{A \vee B \quad A \Rightarrow C \quad B \Rightarrow C}{C}$$

$$\frac{\begin{matrix}[x :: A] \\ \vdots \\ M :: B\end{matrix}}{\lambda x.\ M :: A \to B}$$

$$\frac{\begin{matrix}[A] \\ \vdots \\ B\end{matrix}}{A \Rightarrow B} \Rightarrow\!-\mathrm{I}$$

$$\frac{\lambda x.\ M :: A \to B \qquad N :: A}{(\lambda x. M)\ N :: B}$$

$$\frac{A \Rightarrow B \qquad A}{B} \Rightarrow\!-\mathrm{E}$$

$$\dfrac{\dfrac{A \wedge B}{B} \qquad \dfrac{A \wedge B}{A}}{B \wedge A}$$

$$\dfrac{\dfrac{x :: A * B}{\texttt{snd}\ x :: B} \qquad \dfrac{x :: A * B}{\texttt{fst}\ x :: A}}{(\texttt{snd}\ x,\ \texttt{fst}\ x) :: B * A}$$

- Proof normalisation corresponds to evaluation!

$$
\dfrac{\dfrac{\texttt{A * B}}{\texttt{B}} \qquad \dfrac{\dfrac{\dfrac{\texttt{A * B}}{\texttt{fst x :: A}} \qquad \dfrac{\texttt{A * B}}{\texttt{fst x :: A}}}{\texttt{(fst x, fst x)::A * A}}}{\texttt{snd(fst x, fst x):: A}}}{\texttt{(snd x, snd(fst x, fst x)) :: B * A}}
$$

`(snd x, fst x)`

Curry Howard Isomorphism

- Howard proposed extension for for-all and existentially quantified types (now known as dependent types) to predicate logic
  - de Bruijn's Automath
  - Martin-Löf's type theory (Agda, Idris)
  - PRL, nuPRL
  - Coquant and Huet's calculus of constructions (Coq proof assistant)

Curry Howard Isomorphism

- In short, it is the observation that
  - propositions can be viewed as types
  - programs as their (constructive) proof
  - proof normalisation as program evaluation

Curry Howard Isomorphism

- The pattern of logicians/computer scientist discovering the same system independently has repeated since then multiple times:
  - Second order lambda calculus (Jean-Yves Girard, John Reynolds), basis for Java, C#
  - Principal type inference, by Roger Hindley and Robin Milner (e.g., Haskell)
  - Existential quantification in second order logic as basis for abstraction (John Mitchell, Gordon Plotkin)
  - Girard's linear logic, linear types
  - …?