Utrecht University

Concepts of Programming Languages

Curry Howard and Linear Types

Gabriele Keller
Ron Vanderfeesten

Propositional Logic - Natural Deduction

- $\Gamma \vdash t$: We can deduce a conclusion t from a sequence of premises Γ

$$\frac{}{A \vdash A}\ \text{Identity}$$

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ \text{contraction} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ \text{weakening}$$

$$\frac{\Gamma, \Delta \vdash A}{\Delta, \Gamma \vdash A}\ \text{exchange}$$

- Alternatively, we could state that Γ is a set and do with one rule:

$$\frac{A \in \Gamma}{\Gamma \vdash A}$$

Utrecht University

Propositional Logic - Natural Deduction

- Rules come in pairs: introduction and elimination

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-I}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-E1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-E2}$$

Propositional Logic - Natural Deduction

- ∨-introduction and elimination

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

$$\frac{\Gamma \vdash A \vee B \quad \Gamma \vdash A \Rightarrow C \quad \Gamma \vdash B \Rightarrow C}{\Gamma \vdash C} \vee\text{-E}$$

Propositional Logic - Natural Deduction

- Implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow\text{-I} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash A \Rightarrow B}{\Gamma \vdash B} \Rightarrow\text{-E}$$

Propositional Logic - Natural Deduction

- Negation/False

$$\frac{}{\bot \vdash A}$$

- not is an abbreviation for $A \Rightarrow \bot$

Propositional Logic - Natural Deduction

- These rules correspond to intuitionistic or constructive interpretation of propositional logic:
  - no rule to derive $\neg\neg A \Leftrightarrow A$ or $(A \vee \neg A)$
  - Propositional logic is decidable

Propositional Logic - Natural Deduction

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

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

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

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

Utrecht University

Curry Howard Isomorphism (Correspondence)

- In 1934, Curry observed a relationship between logic implication A ⇒ B and function types A ➔ B in the simply typed lambda calculus

- Howard realised in 1969 that this connection is much deeper

Curry Howard Isomorphism

- Consider MinHs, with sum- and product types
  - can be encoded in the λ-calculus, apart from
  - `recfun,` which can't be directly encoded, as the function needs to be named in the body to be referred to
  - we can implement recursion in the λ-calculus though, as we can apply a function to itself:

```
λf. f f
```

- not a problem in the untyped λ-calculus, but not possible in the simply typed λ-calculus
- reductions in the simply typed λ-calculus terminate
- introduce the Y-combinator 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
sumF f n = if (n == 0) then 0 (n + f (n-1))
sum = Y sumF
```

Utrecht University

Propositional Logic - Natural Deduction

- Propositional logic rules directly correspond to our typing rules:

$$\frac{}{A \vdash A}\ \text{Identity} \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ \text{contraction} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ \text{weakening} \qquad \frac{\Gamma, \Delta \vdash A}{\Delta, \Gamma \vdash A}\ \text{exchange}$$

is equivalent to defining Γ to be a set and the rule

$$\frac{A \in \Gamma}{\Gamma \vdash A}$$

which corresponds to the typing rule for variables:

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

Utrecht University

Propositional Logic - Natural Deduction

- Rules for $\wedge$ and product typing rules:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-I} \qquad \frac{\Gamma \vdash \mathtt{t}_1 : A \quad \Gamma \vdash \mathtt{t}_2 : B}{\Gamma \vdash \mathtt{Pair}\ t_1\ t_2\ :\ A * B}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-E1} \qquad \frac{\Gamma \vdash t\ :\ A * B}{\Gamma \vdash \mathtt{Fst}\ t\ :\ A}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-E2} \qquad \frac{\Gamma \vdash t\ :\ A * B}{\Gamma \vdash \mathtt{Snd}\ t\ :\ B}$$

Propositional Logic - Natural Deduction

- Implication and the function type:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow\text{-I} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \texttt{Fun}\ x\,.\,t\ :\ A \to B}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A \Rightarrow B}{\Gamma \vdash B} \Rightarrow\text{-E} \qquad \frac{\Gamma \vdash t_2\ :\ A \quad \Gamma \vdash t_1\ ::A \to B}{\Gamma \vdash \texttt{Apply}\ t_1\ t_2\ :\ B}$$

Propositional Logic - Natural Deduction

- ∨-rules and the sum type:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-I1} \qquad \frac{\Gamma \vdash t\ :\ A}{\Gamma \vdash \mathtt{InL}\ t\ :\ A + B}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-I2} \qquad \frac{\Gamma \vdash t\ :\ B}{\Gamma \vdash \mathtt{InR}\ t\ :\ A + B}$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma \vdash A \Rightarrow C \quad \Gamma \vdash B \Rightarrow C}{\Gamma \vdash C}\ \vee\text{-E}$$

$$\frac{\Gamma \vdash t\ :\ A + B \quad \Gamma, x : A \vdash t_1\ :\ C \quad \Gamma, y : B \vdash t_2\ :\ C}{\Gamma \vdash \mathtt{Case}\ t\ (x\,.\,t_1)\ (y\,.\,t_t)\ :\ C}$$

Proofs and programs

$$A \wedge B \vdash B \wedge A$$

- Proof:

$$\dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash B} \quad \dfrac{\dfrac{}{A \wedge B \vdash A \wedge B}}{A \wedge B \vdash A}$$
$$A \wedge B \vdash B \wedge A$$

- Corresponding typing judgement

$$\dfrac{\dfrac{\dfrac{}{x : A * B \vdash x : A * B}}{x : A * B \vdash \texttt{Snd}\ x\ : B} \quad \dfrac{\dfrac{}{x : A * B \vdash A * B}}{x : A * B \vdash \texttt{Fst}\ x\ : A}}{x : A * B \vdash \texttt{Pair}\ (\texttt{Snd}\ x)\ (\texttt{Fst}\ x)\ :\ B * A}$$

Proofs and programs

- What about this term?

```
Pair (Snd x) (Snd (Pair (Fst x) (Fst x)))
```

Curry Howard Isomorphism

- In short, it is the observation that
  - propositions can be viewed as types
  - well-typed programs as their (constructive) proof
  - proof normalisation as program evaluation
  - in the simply typed lambda-calculus, all programs terminate
    - what happens if we add the Y-combinator?

Curry Howard Isomorphism

- For which of the following types can we write total, terminating MinHs functions?
  - $\forall\, a.\, \forall\, b.(a * b) \rightarrow (b * a)$
  - $\forall\, a.\, \forall\, b.(a + b) \rightarrow (b + a)$
  - $\forall\, a.\, \forall\, b.(a * b) \rightarrow a$
  - $\forall\, a.\, \forall\, b.(a + b) \rightarrow a$
  - $\forall\, a.\, \forall\, b.(a \rightarrow b) \rightarrow (b \rightarrow a)$
  - $\forall\, a.\, \forall\, b.\, \forall\, c.\, (a \rightarrow b) \rightarrow (b \rightarrow c) \rightarrow (a \rightarrow c)$
- terminating programs to (constructive) proofs, and the type checker acts as a proof checker

Adding True and False

- We discussed the type interpretation of ∧ (*) , ∨ (+), and ⇒ (➔)

- What about the logic constants `True` and `False`?

  - `True` can be represented by a type with exactly one element (unit):

```
()

data Unit = Unit
```

- `False` by the empty type:

```
data Void
```

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

- 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

Propositional logic and simply typed MinHs/λ-calculus

- Propositions represent assumptions, can be used as many types as we like, or be ignored
- Typed identifiers in the environment can also be used as many times as we like (or ignored)
- How can we restrict access to a variable?
  - abstract data types

Example: modelling state

- For example, the fresh variable store in the assignment:

```
module MinHS.TCMonad (TC
                     , freshNames
                     , runTC)

newtype TC a = TC ([Id] -> ([Id], a))

instance Monad TC where
  …

runTC :: TC a -> a
runTC (TC f) = map snd (f freshNames)

fresh :: TC Type
fresh = TC $ \(x:xs) -> (xs,TypeVar x)
```

Utrecht University

Example: modelling state

- The fact that `TC` is a monad means
  - we have `return :: a -> TC a`
  - and `(>>=) :: TC a -> (a -> TC b) -> TC b`
  - and syntactic sugar

```
do
   newName1 <- fresh
   newName2 <- fresh
   return (Prod newName1 newName2)
```

Tracking access to objects

- Other scenarios where it would be good to track access to objects
  - knowing when to safely change a structure using destructive update
  - knowing when to garbage collect an object

Linear Logic

- In linear logic, logical atoms can be interpreted as resources
  - can’t be arbitrarily copied
  - don’t disappear
- New operators:
  - A ⊗ B : you’ve got A and B
  - A & B : you can pick either A or B
  - A ⊕ B : you get A or B (not your choice)
  - A ⊸ B : consumes A, gives B
  - !A : you’ve got an unlimited amount of A
- Assumptions
  - <A> : linear assumption (weakening and contraction don’t hold)
  - [A] : intuitionistic assumption (weakening and contraction hold)

Linear Logic

- Example:

  lunch special: for 10$, you get one serving of tempura, as much rice as you like, your choice of side salad or veggies, and dessert of the day (fruit or ice cream, depending what's on offer)

  ($1⊗ $1⊗ $1⊗ $1⊗ $1⊗ $1⊗ $1⊗ $1⊗ $1⊗ $1)

  ⊸

  (Tempura ⊗ !Rice ⊗ (Soup & Veggies) ⊗ (Fruit ⊕ IceCream))

$$\frac{}{[A] \vdash A}\ [Id] \qquad \frac{}{\langle A \rangle \vdash A}\ \langle Id \rangle$$

$$\frac{\Gamma, \Delta \vdash A}{\Delta, \Gamma \vdash A}\ \textit{Exchange}$$

$$\frac{\Gamma, [A], [A] \vdash B}{\Gamma, [A] \vdash B}\ \textit{Contraction} \qquad \frac{\Gamma \vdash B}{\Gamma, [A] \vdash B}\ \textit{Weakening}$$

Utrecht University

$$\frac{[\Gamma] \vdash A}{[\Gamma] \vdash\ !A}\quad !-Intro$$

$$\frac{\Gamma \vdash\ !A \quad \Delta, [A] \vdash B}{\Gamma, \Delta \vdash B}\quad !-Elim$$

$$\frac{\Gamma, \langle A \rangle \vdash B}{\Gamma \vdash\ A \multimap B} \quad \multimap - Intro$$

$$\frac{\Gamma \vdash A \multimap B \quad \Delta \vdash\ A}{\Gamma, \Delta \vdash\ B} \quad \multimap - Elim$$

- $A \otimes B$: you've got A and B

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \otimes - Intro$$

$$\frac{\Gamma \vdash A \otimes B \quad \Delta, \langle A \rangle, \langle B \rangle \vdash C}{\Gamma, \Delta \vdash C} \otimes - Elim$$

- A & B : you can pick either A or B

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B} \;\; \& - Intro$$

$$\frac{\Gamma \vdash A \,\&\, B}{\Gamma \vdash A} \;\; \& - Elim - 1$$

$$\frac{\Gamma \vdash A \,\&\, B}{\Gamma \vdash B} \;\; \& - Elim - 1$$

- $A \oplus B$: you get A or B (not your choice)

$$\frac{\Gamma \vdash A}{\Gamma, \Delta \vdash A \oplus B} \oplus - Intro - 1 \qquad \frac{\Gamma \vdash B}{\Gamma, \Delta \vdash A \oplus B} \oplus - Intro - 2$$

$$\frac{\Gamma \vdash A \oplus B \quad \Delta, \langle A \rangle \vdash C \quad \Delta, \langle B \rangle \vdash C}{\Gamma, \Delta \vdash C} \oplus - Elim$$

A linearly type simple language

- Taken from Phil Wadler's 'Linear types can change the world' paper:

$$
\begin{array}{lll}
s,t,u & ::= & x \\
 & & \mid \boldsymbol{\lambda}\langle x\rangle\boldsymbol{.}t \mid s\ t \\
 & & \mid \boldsymbol{!}t \quad \mid \boldsymbol{case}\ s\ \boldsymbol{of}\ \boldsymbol{!}x \rightarrow u \\
 & & \mid \langle s,t\rangle \quad \mid \boldsymbol{case}\ s\ \boldsymbol{of}\ \langle x, y\rangle \rightarrow t \\
 & & \mid \boldsymbol{Inl}\ s \mid \boldsymbol{Inl}\ s \quad \mid \boldsymbol{case}\ s\ \boldsymbol{of}\ \boldsymbol{Inl}\ x \rightarrow t; \boldsymbol{Inr}\ y \rightarrow u \\
 & & \mid \langle\langle s,t\rangle\rangle \mid \boldsymbol{Fst}\ s \mid \boldsymbol{Snd}\ s
\end{array}
$$

Utrecht University

$$\frac{}{[x : A] \vdash x : A}\ [Id] \qquad \frac{}{\langle x : A \rangle \vdash x : A}\ \langle Id \rangle$$

$$\frac{\Gamma, \Delta \vdash t : A}{\Delta, \Gamma \vdash t : A}\ Exchange$$

$$\frac{\Gamma, [y : A], [z : A] \vdash u : B}{\Gamma, [x : A] \vdash u[y := x, z := x] : B}\ Contraction$$

$$\frac{\Gamma \vdash u : B}{\Gamma, [x : A] \vdash u : B}\ Weakening$$

$$\frac{[\Gamma] \vdash t : A}{[\Gamma] \vdash\ !t :\ !A}\ !-\mathit{Intro}$$

$$\frac{\Gamma \vdash s :\ !A \quad \Delta, [x : A] \vdash B}{\Gamma, \Delta \vdash \mathbf{case}\ s\ \mathbf{of}\ !x \to u : B}\ !-\mathit{Elim}$$

$$\frac{\Gamma, \langle x : A \rangle \vdash B}{\Gamma \vdash \lambda\langle x \rangle \,.\, u : A \multimap B} \quad \multimap - Intro$$

$$\frac{\Gamma \vdash s : A \multimap B \quad \Delta \vdash t : A}{\Gamma, \Delta \vdash s\langle t \rangle : B} \quad \multimap - Elim$$

$$\frac{\Gamma \vdash t : A \quad \Delta \vdash u : B}{\Gamma, \Delta \vdash \langle t, u \rangle : A \otimes B} \otimes - Intro$$

$$\frac{\Gamma \vdash s : A \otimes B \quad \Delta, \langle x : A \rangle, \langle y : B \rangle \vdash C}{\Gamma, \Delta \vdash \textbf{case } s \textbf{ of } \langle x, y \rangle \to v : C} \otimes - Elim$$

Utrecht University

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \langle\langle t, u \rangle\rangle : A \,\&\, B} \;\; \& - Intro$$

$$\frac{\Gamma \vdash s : A \,\&\, B}{\Gamma \vdash \mathbf{Fst}\ s : A} \;\; \& - Elim - 1$$

$$\frac{\Gamma \vdash s : A \,\&\, B}{\Gamma \vdash \mathbf{Snd}\ t : B} \;\; \& - Elim - 1$$

$$\frac{\Gamma \vdash t : A}{\Gamma, \Delta \vdash \mathbf{InL}\ t : A \oplus B} \quad \oplus - Intro - 1$$

$$\frac{\Gamma \vdash s : B}{\Gamma, \Delta \vdash \mathbf{InR}\ s : A \oplus B} \quad \oplus - Intro - 2$$

$$\frac{\Gamma \vdash s : A \oplus B \quad \Delta, \langle x : A \rangle \vdash C \quad \Delta, \langle y : B \rangle \vdash w : C}{\Gamma, \Delta \vdash \mathbf{case}\ s\ \mathbf{of}\ \mathbf{Inl}\ \langle x \rangle \to v;\ \mathbf{InR} \langle y \rangle \to w : C} \quad \oplus - Elim$$

Utrecht University

Ownership-types in practice

- A linear type system is very restrictive
  - every function has to return all its linear arguments, otherwise they can't be used by the caller anymore
  - that might be ok for functions which 'update' the arguments, but is annoying for read-only functions
  - there are work-arounds, by allowing read-only access in a limited scope, but it's still onerous

Ownership-types in practice

- The systems programming language Rust uses an ownership type system
  - closer to a so-called affine type system: variables can be used at most once (not exactly once, as in linear)
- Motivation:
  - cleaner and safer than C, so no manual memory management
  - but still efficient, with predictable performance (no garbage collection)

Ownership-types in practice

```
fn foo (){
    let v1 = vec![1,2,3,4,5];
    let v2 = v1;
    printL (v2);
    println!(" {:?}",v2);
}
fn printL (a:Vec<i32>){
    println!("{}",a);
}
```

Ownership-types in practice

```
fn foo (){
    let v1 = vec![1,2,3,4,5];
    let v2 = v1;
    printV (&v2);
    println!(" {:?}",v2);
}
fn printB (a:&Vec<i32>){
    println!("{}",a);
}
```

Utrecht University