Inductive types

(§6.4, p76 →)

PLC-style encoding of algebraic datatypes

booleans	$\forall\alpha\,(\alpha \to \alpha \to \alpha)$
natural numbers	$\forall\alpha\,(\alpha \to (\alpha \to \alpha) \to \alpha)$
etc	

Calculus of Constructions

is the Pure Type System $\lambda\mathbf{C}$, where $\mathbf{C} = (\mathcal{S}_{\mathrm{C}}, \mathcal{A}_{\mathrm{C}}, \mathcal{R}_{\mathrm{C}})$ is the PTS specification with

$$\begin{aligned}
\mathcal{S}_{\mathrm{C}} &\triangleq \{\mathtt{Prop}, \mathtt{Set}\} \\
\mathcal{A}_{\mathrm{C}} &\triangleq \{(\mathtt{Prop}, \mathtt{Set})\} \\
\mathcal{R}_{\mathrm{C}} &\triangleq \{(\mathtt{Prop}, \mathtt{Prop}, \mathtt{Prop})^{1}, (\mathtt{Set}, \mathtt{Prop}, \mathtt{Prop})^{2}, \\
&\qquad (\mathtt{Prop}, \mathtt{Set}, \mathtt{Set})^{3}, (\mathtt{Set}, \mathtt{Set}, \mathtt{Set})^{4}\}
\end{aligned}$$

1. $\mathtt{Prop}$ has implications, $\phi \to \psi = \Pi x : \phi\ (\psi)$ (where $\phi, \psi : \mathtt{Prop}$ and $x \notin fv(q)$).

2. $\mathtt{Prop}$ has universal quantifications over elements of a type, $\Pi x : A\ (\phi(x))$ (where $A : \mathtt{Set}$ and $x : A \vdash \phi(x) : \mathtt{Prop}$).
N.B. A might be $\mathtt{Prop}$ ($\lambda 2 \subseteq \lambda\mathbf{C}$).

3. $\mathtt{Set}$ has types of function dependent on proofs of a proposition, $\Pi x : p\ (A(x))$ (where $p : \mathtt{Prop}$ and $x : p \vdash A(x) : \mathtt{Set}$).

4. $\mathtt{Set}$ has dependent function types, $\Pi x : A\ (B(x))$ (where $A : \mathtt{Set}$ and $x : A \vdash B(x) : \mathtt{Set}$).

PLC-style encoding of algebraic datatypes in λC

booleans $\quad \Pi p : Prop\,(p \to p \to p)$

natural numbers $\quad \Pi p : Prop\,(p \to (p \to p) \to p)$

etc

PLC-style encoding of algebraic datatypes in λC

$$\mathrm{bool} \triangleq \Pi p : \mathrm{Prop}\,(p \to p \to p)$$

$$\mathrm{nat} \triangleq \Pi p : \mathrm{Prop}\,(p \to (p \to p) \to p)$$

have $\Diamond \vdash \mathrm{bool} : \mathrm{Prop}$

and $\Diamond \vdash \mathrm{nat} : \mathrm{Prop}$

and $\Diamond \vdash t : \mathrm{bool} \leftrightarrow \mathrm{nat}$ for some t

$$\boxed{\text{How can we get bool, nat, etc of type Set?}}$$

PLC-style encoding of algebraic datatypes in λC

$$nat \triangleq \Pi p : Prop\,(p \to (p \to p) \to p)$$

$$\Pi x : Set\,(x \to (x \to x) \to x)$$

is not typeable in λC

(needs a sort s with $Set : s$)

How can we get bool, nat, etc of type Set?

The Pure Type System $\boldsymbol{\lambda}\mathbf{U}$

is given by the PTS specification $\mathbf{U} = (\mathcal{S}_{\mathrm{U}}, \mathcal{A}_{\mathrm{U}}, \mathcal{R}_{\mathrm{U}})$, where:

$$\mathcal{S}_{\mathrm{U}} \triangleq \{\mathtt{Prop}, \mathtt{Set}, \mathtt{Type}\}$$

$$\mathcal{A}_{\mathrm{U}} \triangleq \{(\mathtt{Prop}, \mathtt{Set}), (\mathtt{Set}, \mathtt{Type})\}$$

$$\begin{aligned}\mathcal{R}_{\mathrm{U}} \triangleq \{&(\mathtt{Prop}, \mathtt{Prop}, \mathtt{Prop}), (\mathtt{Set}, \mathtt{Prop}, \mathtt{Prop}), (\mathtt{Type}, \mathtt{Prop}, \mathtt{Prop}),\\ &(\mathtt{Set}, \mathtt{Set}, \mathtt{Set}), (\mathtt{Type}, \mathtt{Set}, \mathtt{Set})\}\end{aligned}$$

The Pure Type System $\boldsymbol{\lambda}\mathbf{U}$

is given by the PTS specification $\mathbf{U} = (\mathcal{S}_{\mathrm{U}}, \mathcal{A}_{\mathrm{U}}, \mathcal{R}_{\mathrm{U}})$, where:

$$\begin{aligned}
\mathcal{S}_{\mathrm{U}} &\triangleq \{\mathtt{Prop}, \mathtt{Set}, \mathtt{Type}\} \\
\mathcal{A}_{\mathrm{U}} &\triangleq \{(\mathtt{Prop}, \mathtt{Set}), (\mathtt{Set}, \mathtt{Type})\} \\
\mathcal{R}_{\mathrm{U}} &\triangleq \{(\mathtt{Prop}, \mathtt{Prop}, \mathtt{Prop}), (\mathtt{Set}, \mathtt{Prop}, \mathtt{Prop}), (\mathtt{Type}, \mathtt{Prop}, \mathtt{Prop}), \\
&\qquad (\mathtt{Set}, \mathtt{Set}, \mathtt{Set}), (\mathtt{Type}, \mathtt{Set}, \mathtt{Set})\}
\end{aligned}$$

Theorem (Girard). $\lambda\mathbf{U}$ is logically inconsistent: every legal proposition $\Gamma \vdash P : \mathtt{Prop}$ has a proof $\Gamma \vdash M : P$. (In particular, there is a proof of falsity $\perp \triangleq \Pi p : \mathtt{Prop}\,(p)$.)

Inductive types (informally)

An inductive type is specified by giving

- *constructor functions* that allow us to inductively generate data values of that type
  (Some restrictions on how the inductive type appears in the domain type of constructors is needed to ensure termination of reduction and logical consistency.)
- *eliminators* for constructing functions on the data
- *computation rules* that explain how to simplify an eliminator applied to constructors.

Extending $\boldsymbol{\lambda}\mathbf{C}$ with an inductive type of natural numbers

Pseudo-terms

$$t ::= \cdots \mid \mathtt{Nat} \mid \mathtt{zero} \mid \mathtt{succ} \mid \mathtt{elimNat}(x.t)\ t\ t$$

Extending λC with an inductive type of natural numbers

Pseudo-terms

$$t ::= \cdots \mid \mathtt{Nat} \mid \mathtt{zero} \mid \mathtt{succ} \mid \mathtt{elimNat}(x.t)\; t\; t$$

Typing rules

- formation: $\diamond \vdash \mathtt{Nat} : \mathtt{Set}$
- introduction: $\diamond \vdash \mathtt{zero} : \mathtt{Nat} \qquad \diamond \vdash \mathtt{succ} : \mathtt{Nat} \to \mathtt{Nat}$
- elimination: $\dfrac{\Gamma, x : \mathtt{Nat} \vdash A(x) : s \qquad \Gamma \vdash M : A(\mathtt{zero}) \qquad \Gamma \vdash F : \Pi x : \mathtt{Nat}\; (A(x) \to A(\mathtt{succ}\; x))}{\Gamma \vdash \mathtt{elimNat}(x.A)\; M\; F : \Pi x : \mathtt{Nat}\; (A(x))}$

  (where $A(t)$ stands for $A[t/x]$)

Extending λC with an inductive type of natural numbers

Pseudo-terms

$$t ::= \cdots \mid \mathtt{Nat} \mid \mathtt{zero} \mid \mathtt{succ} \mid \mathtt{elimNat}(x.t)\, t\, t$$

Typing rules

- formation: $\diamond \vdash \mathtt{Nat} : \mathtt{Set}$
- introduction: $\diamond \vdash \mathtt{zero} : \mathtt{Nat} \qquad \diamond \vdash \mathtt{succ} : \mathtt{Nat} \to \mathtt{Nat}$
- elimination: $$\frac{\Gamma, x : \mathtt{Nat} \vdash A(x) : s \qquad \Gamma \vdash M : A(\mathtt{zero}) \qquad \Gamma \vdash F : \Pi x : \mathtt{Nat}\,(A(x) \to A(\mathtt{succ}\, x))}{\Gamma \vdash \mathtt{elimNat}(x.A)\, M\, F : \Pi x : \mathtt{Nat}\,(A(x))}$$

  (where $A(t)$ stands for $A[t/x]$)

gives us (dep. typed) functions defined by primitive recursion, eg

addition $\lambda x : \mathrm{Nat}\,(\mathrm{elimNat}(y.\mathrm{Nat})\, x\, (\lambda y : \mathrm{Nat}\,(\mathrm{succ})))$

Extending λC with an inductive type of natural numbers

Pseudo-terms

$$t ::= \cdots \mid \mathtt{Nat} \mid \mathtt{zero} \mid \mathtt{succ} \mid \mathtt{elimNat}(x.t)\, t\, t$$

Typing rules

- formation: $\diamond \vdash \mathtt{Nat} : \mathtt{Set}$
- introduction: $\diamond \vdash \mathtt{zero} : \mathtt{Nat}$ $\qquad \diamond \vdash \mathtt{succ} : \mathtt{Nat} \to \mathtt{Nat}$
- elimination: $\dfrac{\Gamma, x : \mathtt{Nat} \vdash A(x) : s \qquad \Gamma \vdash M : A(\mathtt{zero}) \qquad \Gamma \vdash F : \Pi x : \mathtt{Nat}\, (A(x) \to A(\mathtt{succ}\, x))}{\Gamma \vdash \mathtt{elimNat}(x.A)\, M\, F : \Pi x : \mathtt{Nat}\, (A(x))}$

  (where $A(t)$ stands for $A[t/x]$)

Computation rules

$$\mathtt{elimNat}(x.A)\, M\, F\, \mathtt{zero} \longrightarrow M$$

$$\mathtt{elimNat}(x.A)\, M\, F\, (\mathtt{succ}\, N) \longrightarrow F\, N\, (\mathtt{elimNat}(x.A)\, M\, F\, N)$$

Extending λC with an inductive type of natural numbers

Pseudo-terms

$$t ::= \cdots \mid \texttt{Nat} \mid \texttt{zero} \mid \texttt{succ} \mid \texttt{elimNat}(x.t)\, t\, t$$

Typing rules

- formation: $\diamond \vdash \texttt{Nat} : \texttt{Set}$
- introduction: $\diamond \vdash \texttt{zero} : \texttt{Nat} \qquad \diamond \vdash \texttt{succ} : \texttt{Nat} \to \texttt{Nat}$
- elimination: $$\frac{\Gamma, x : \texttt{Nat} \vdash A(x) : s \qquad \Gamma \vdash M : A(\texttt{zero}) \qquad \Gamma \vdash F : \Pi x : \texttt{Nat}\, (A(x) \to A(\texttt{succ}\, x))}{\Gamma \vdash \texttt{elimNat}(x.A)\, M\, F : \Pi x : \texttt{Nat}\, (A(x))}$$

  (where $A(t)$ stands for $A[t/x]$)

also gives us <u>proof by induction</u>

$\varphi(\text{zero}) \wedge \forall x\, (\varphi(x) \to \varphi(\text{succ}\, x))$

$\to \forall x\, \varphi(x)$

Inductive types of vectors

For a fixed parameter $\Gamma \vdash A : s$, the indexed family $(\mathtt{Vec}_A\, x \mid x : \mathtt{Nat})$ of types $\mathtt{Vec}_A\, x$ of *lists of A-values of length x* is inductively defined as follows:

Inductive types of vectors

For a fixed parameter $\Gamma \vdash A : s$, the indexed family $(\mathtt{Vec}_A\, x \mid x : \mathtt{Nat})$ of types $\mathtt{Vec}_A\, x$ of *lists of A-values of length x* is inductively defined as follows:

Formation:

$$\frac{\Gamma \vdash N : \mathtt{Nat}}{\Gamma \vdash \mathtt{Vec}_A\, N : \mathtt{Set}}$$

Introduction:

$$\Gamma \vdash \mathtt{vnil}_A : \mathtt{Vec}_A\, \mathtt{zero}$$

$$\Gamma \vdash \mathtt{vcons}_A : A \to \Pi x : \mathtt{Nat}\, (\mathtt{Vec}_A\, x \to \mathtt{Vec}_A\, (\mathtt{succ}\, x))$$

Elimination and Computation:

Inductive types of vectors

For a fixed parameter $\Gamma \vdash A : s$, the indexed family $(\mathtt{Vec}_A\, x \mid x : \mathtt{Nat})$ of types $\mathtt{Vec}_A\, x$ of *lists of A-values of length x* is inductively defined as follows:

Formation:

$$\frac{\Gamma \vdash N : \mathtt{Nat}}{\Gamma \vdash \mathtt{Vec}_A\, N : \mathtt{Set}}$$

Introduction:

$$\Gamma \vdash \mathtt{vnil}_A : \mathtt{Vec}_A\, \mathtt{zero}$$

$$\Gamma \vdash \mathtt{vcons}_A : A \to \Pi x : \mathtt{Nat}\, (\mathtt{Vec}_A\, x \to \mathtt{Vec}_A\, (\mathtt{succ}\, x))$$

Elimination and Computation:

[do-it-yourself]

Inductive identity propositions

For fixed parameters $\Gamma \vdash A : s$ and $\Gamma \vdash a : A$, the indexed family $(\mathtt{Id}_{A,a}\, x \mid x : A)$ of propositions $\mathtt{Id}_{A,a}\, x$ that a *and* x *are equal elements of type* A is inductively defined as follows:

Inductive identity propositions

For fixed parameters $\Gamma \vdash A : s$ and $\Gamma \vdash a : A$, the indexed family $(\mathtt{Id}_{A,a}\, x \mid x : A)$ of propositions $\mathtt{Id}_{A,a}\, x$ that a *and* x *are equal elements of type* A is inductively defined as follows:

Formation:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathtt{Id}_{A,a}\, M : \mathtt{Prop}}$$

Introduction:

$$\Gamma \vdash \mathtt{refl}_{A,a} : \mathtt{Id}_{A,a}\, a$$

Elimination:

$$\frac{\Gamma, x : A, p : \mathtt{Id}_{A,a}\, x \vdash B(x,p) : s \qquad \Gamma \vdash N : B(a, \mathtt{refl}_{A,a})}{\Gamma \vdash \mathtt{J}_{A,a}(x,p.\, B)\, N : \Pi x : A\, (\Pi p : \mathtt{Id}_{A,a}\, x\, (B(x,p)))}$$

Computation:

$$\mathtt{J}_{A,a}(x,p.\, B)\, N\, a\, \mathtt{refl}_{A,a} \rightarrow N$$

Inductive identity propositions

programming/proving using eliminators gets tricky very rapidly (cf. Ex.Sh. qu 19 about proving $\forall n\,(n = 0+n)$)

Elimination:

$$\frac{\Gamma, x : A, p : \mathtt{Id}_{A,a}\,x \vdash B(x,p) : s \qquad \Gamma \vdash N : B(a, \mathtt{refl}_{A,a})}{\Gamma \vdash \mathtt{J}_{A,a}(x,p.\,B)\,N : \Pi x : A\,(\Pi p : \mathtt{Id}_{A,a}\,x\,(B(x,p)))}$$

Computation:

$$\mathtt{J}_{A,a}(x,p.\,B)\,N\;a\,\mathtt{refl}_{A,a} \to N$$

Agda proof of $\forall x \in \mathbb{N}\ (x = 0 + x)$

```
data Nat : Set where
  zero : Nat
  succ : Nat -> Nat

add : Nat -> Nat -> Nat
add x zero     = x
add x (succ y) = succ (add x y)
```

Agda proof of $\forall x \in \mathbb{N}\,(x = 0 + x)$

```
data Nat : Set where
  zero : Nat
  succ : Nat -> Nat

add : Nat -> Nat -> Nat
add x zero     = x
add x (succ y) = succ (add x y)

data Id (A : Set)(x : A) : A -> Set where
  refl : Id A x x

cong : (A B : Set)(f : A -> B)(x y : A) ->
       Id A x y -> Id B (f x) (f y)
cong A B f x .x refl = refl
```

Agda proof of $\forall x \in \mathbb{N}\ (x = 0 + x)$

```
data Nat : Set where
  zero : Nat
  succ : Nat -> Nat

add : Nat -> Nat -> Nat
add x zero     = x
add x (succ y) = succ (add x y)

data Id (A : Set)(x : A) : A -> Set where
  refl : Id A x x

cong : (A B : Set)(f : A -> B)(x y : A) ->
       Id A x y -> Id B (f x) (f y)
cong A B f x .x refl = refl

P : (x : Nat) -> Id Nat x (add zero x)
P zero     = refl
P (succ x) = cong Nat Nat succ x (add zero x) (P x)
```

Uniqueness of identity proofs

In $\lambda\mathbf{C}$ extended with inductive identity propositions, there are some types $\Gamma \vdash A : s$ for which it is impossible to prove that all equality proofs in $\mathtt{Id}_{A,x}\, y$ (where $x, y : A$) are identical.
That is, there is no pseudo-term uip satisfying

$$\Gamma \vdash uip : \Pi x, y : A\, (\Pi p, q : \mathtt{Id}_{A,x}\, y\, (\mathtt{Id}_{(\mathtt{Id}_{A,x}\, y),p}\, q))$$

Uniqueness of identity proofs

In $\lambda\mathbf{C}$ extended with inductive identity propositions, there are some types $\Gamma \vdash A : s$ for which it is impossible to prove that all equality proofs in $\mathtt{Id}_{A,x}\, y$ (where $x, y : A$) are identical.
That is, there is no pseudo-term uip satisfying

$$\Gamma \vdash uip : \Pi x, y : A\, (\Pi p, q : \mathtt{Id}_{A,x}\, y\, (\mathtt{Id}_{(\mathtt{Id}_{A,x}\, y),p}\, q))$$

By contrast, in Agda we have:

```
data Id (A : Set)(x : A) : A -> Set where
  refl : Id A x x

uip : (A : Set)(x y : A)(p q : Id A x y) -> Id (Id A x y) p q
uip A x .x refl refl = refl
```

Dependent function types $(\Pi x : A)\, B$

ML — type schemes, function types

PLC — $\forall$-types, function types

F_ω, λC — Π-types

PTS's

Dependent function types $(\Pi x : A)\, B$

ML "Turing powerful"
termination undecidable

PTS's: PLC; F_ω, λC

only total functions:
termination decidable
$\rightsquigarrow$ decidable type-checking

Dependent function types $(\Pi x : A)\,B$

ML "impure" computation has side-effects

PLC pure

PTS's $\left\{ \begin{array}{l} \text{PLC} \\ F_\omega, \lambda C \end{array} \right.$

Dependent function types $(\Pi x : A)\,B$

ML	"impure" computation has side-effects
PLC	pure
PTS's: F_ω, λC	

? make sense of Propositions-as-Types in presence of side-effects ?