How to express a theory in Dedukti?

A logical framework

The implementations of Dedukti: Dkcheck, Lambdapi, Kontroli... are not proof-checkers specific to one theory (the Calculus of constructions, Set theory...)

But logical frameworks, where you can define your own theory (like Predicate logic)

- Define your theory
- Check proofs expressed in this theory

Beyond Predicate logic

Logical frameworks: λ-Prolog, Isabelle, The Edinburgh logical framework, Pure type systems, Deduction modulo theory, Ecumenical logic

In Dedukti

- Function symbols can bind variables (like in λ-Prolog, Isabelle, The Edinburgh logical framework)
- Proofs are terms (like in The Edinburgh logical framework)
- Deduction and computation are mixed (like in Deduction modulo theory)
- Both constructive and classical proofs can be expressed (like in Ecumenical logic)

The two features of Dedukti

Dedukti is a typed λ-calculus with

- Dependent types
- Computation rules

No typing rules today, but illustration of these features with examples

What is a theory?

In Predicate logic: a language (sorts, function symbols, and predicate symbols), and a set of axioms

In Dedukti: a set of symbols (replaces sorts, function symbols, predicate symbols, and axioms), and a set of computation rules

I. Catching up with Predicate logic

Predicate logic is a sophisticated framework with notions of sort, function symbol, predicate symbol, arity, variable, term, proposition, proof...

A typed λ-calculus is much more primitive

These notions must be constructed

Building Predicate logic

An easy warm up exercise

An easy way to illustrate the use of dependent types and computation rules

An interest in itself: The first book of Euclid's elements (originally formalized in Coq) can be expressed in Predicate logic + the axioms of geometry and exported to many systems (Géran)

Terms and propositions: a first attempt

I : TYPE
function symbols: $I \to \ldots \to I \to I$

$Prop$: TYPE
predicate symbols: $I \to \ldots \to I \to Prop$
$\Rightarrow$: $Prop \to Prop \to Prop$
$\forall$: $(I \to Prop) \to Prop$

- Symbol declarations only (no computation rules yet)
- Simply typed λ-calculus (no dependent types yet)
- Types are terms of type TYPE
- $\forall$ binds (higher-order abstract syntax: $\forall x\ A$ expressed as $\forall\ \lambda x\ A$)

Works if we want one sort

But if we want several (like in geometry: points, lines, circles...)

I_1 : TYPE

I_2 : TYPE

I_3 : TYPE

Several (an infinite number of?) symbols and several (an infinite number of?) quantifiers

$\forall_1 : (I_1 \to Prop) \to Prop$

$\forall_2 : (I_2 \to Prop) \to Prop$

$\forall_3 : (I_3 \to Prop) \to Prop$

Making the universal quantifier generic

Something like
$\forall : \Pi X : \mathsf{TYPE}, ((X \to \mathit{Prop}) \to \mathit{Prop})$

But does not work for two reasons

- (a minor one) no dependent products on TYPE
- (a major one) many things in TYPE beyond I_1, I_2, and I_3 (e.g. *Prop*)

Making the universal quantifier generic

I : TYPE $\qquad I_1$: TYPE, I_2 : TYPE, I_3 : TYPE

Set : TYPE

ι : Set $\qquad \iota_1 : Set, \iota_2 : Set, \iota_3 : Set$

$El : Set \to$ TYPE

$El\ \iota \longrightarrow I$ $\qquad El\ \iota_1 \longrightarrow I_1, El\ \iota_2 \longrightarrow I_2, El\ \iota_3 \longrightarrow I_3$

$Prop$: TYPE

$\Rightarrow\ : Prop \to Prop \to Prop$

$\forall : \Pi x : Set, (El\ x \to Prop) \to Prop$

Uses dependent types and computation rules

Reminiscent of expression of Simple type theory in Predicate logic, universes *à la* Tarski...

Proofs

So far: terms and propositions. Now: proofs

Proofs are trees, they can be expressed in Dedukti

Curry-de Bruijn-Howard: $P \Rightarrow P$ should be the type of its proofs
But not possible here $P \Rightarrow P : \mathit{Prop} : \mathrm{TYPE}$ is not itself a type

$\mathit{Prf} : \mathit{Prop} \to \mathrm{TYPE}$
mapping each proposition to the type of its proofs: $\mathit{Prf}\,(P \Rightarrow P) : \mathrm{TYPE}$

Not all types are types of proofs (e.g. I, $\mathit{El}\,\iota$, Prop...)

Proofs

Brouwer-Heyting-Kolmogorov: $\lambda x : (\mathit{Prf}\ P), x$ should be a proof of $P \Rightarrow P$
But has type $(\mathit{Prf}\ P) \to (\mathit{Prf}\ P)$ and not $\mathit{Prf}\ (P \Rightarrow P)$
$\mathit{Prf}\ (P \Rightarrow P)$ and $(\mathit{Prf}\ P) \to (\mathit{Prf}\ P)$ must be identified

A computation rule

$$\mathit{Prf}\ (x \Rightarrow y) \longrightarrow (\mathit{Prf}\ x) \to (\mathit{Prf}\ y)$$

In the same way

$$\mathit{Prf}\ (\forall\ x\ p) \longrightarrow \Pi z : (\mathit{El}\ x), (\mathit{Prf}\ (p\ z))$$

The function *Prf* is an injective morphism from propositions to types: it is the Curry-de Bruijn-Howard isomorphism

Connectives

So far: $\Rightarrow$ and $\forall$ only

$\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\exists$ defined *à la* Russell

$\wedge : \mathit{Prop} \rightarrow \mathit{Prop} \rightarrow \mathit{Prop}$
$\mathit{Prf}\,(x \;\wedge\; y) \longrightarrow \Pi z : \mathit{Prop}, ((\mathit{Prf}\,x \rightarrow \mathit{Prf}\,y \rightarrow \mathit{Prf}\,z) \rightarrow \mathit{Prf}\,z)$

Classical connectives

So far: constructive deduction rules only

What if you want to express classical proofs (a logical framework ought to be neutral)

Ecumenical logic: constructive and classical disjunction are governed by different rules: they are different symbols (like inclusive and exclusive disjunction): $\vee$ and $\vee_c$

$\Rightarrow_c$, $\wedge_c$, $\vee_c$, $\forall_c$, $\exists_c$ defined using negative translation as a definition

$\wedge_c : \mathit{Prop} \to \mathit{Prop} \to \mathit{Prop}$

$\wedge_c \longrightarrow \lambda x : \mathit{Prop}, \lambda y : \mathit{Prop}, ((\neg\neg x) \ \wedge \ (\neg\neg y))$

Note: also a symbol Prf_c

If you want to express proofs coming from Predicate logic

e.g. Vampire, VeriT...

You know enough

II. Simple type theory (HOL4, HOL Light, Isabelle/HOL...)

Two ideas

Propositions as objects

Functions

Propositions as objects

$o : Set$
$El\ o \longrightarrow Prop$

$\forall\ o : (El\ o \to Prop) \to Prop$
$\forall\ o : (Prop \to Prop) \to Prop$
$\forall\ o\ (\lambda X : Prop, (X \Rightarrow X)) : Prop$

$Prf\ (\forall\ o\ (\lambda X : Prop, (X \Rightarrow X))) : \mathsf{TYPE}$
$Prf\ (\forall\ o\ (\lambda X : Prop, (X \Rightarrow X))) \longrightarrow \Pi X : Prop, ((Prf\ X) \to (Prf\ X))$

$\lambda X : Prop, \lambda y : (Prf\ X), y : Prf\ (\forall\ o\ (\lambda X : Prop, (X \Rightarrow X)))$

Functions

$\rightsquigarrow : Set \to Set \to Set$

$El\,(x \rightsquigarrow y) \longrightarrow (El\,x) \to (El\,y)$

An infinite number of elements in $Set\,(\iota, o, \rightsquigarrow)$

Polymorphism

In HOL4, HOL Light, Isabelle/HOL... more than in Church's Simple type theory
Object-level prenex polymorphism

In fact: two different features
nil : $\forall X\ (\text{list}\ X)$
$\forall X\ (\text{nil}\ X = \text{nil}\ X)$

Polymorphism

$Scheme : \mathrm{TYPE}$
$\uparrow : Set \to Scheme$
$\forall : (Set \to Scheme) \to Scheme$

$Els : Scheme \to \mathrm{TYPE}$
$Els\ (\uparrow x) \longrightarrow El\ x$
$Els(\forall\ p) \longrightarrow \Pi x : Set, Els\ (p\ x)$

$\breve{\forall} : (Set \to Prop) \to Prop$
$Prf\ (\breve{\forall}\ p) \longrightarrow \Pi x : Set, Prf\ (p\ x)$

III. Dependency

Dependent function type

Non dependent function types

$\leadsto : Set \to Set \to Set$

$El\,(x \leadsto y) \longrightarrow (El\,x) \to (El\,y)$

can be made dependent

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

$El\,(x \leadsto_d y) \longrightarrow \Pi z : El\,x, El\,(y\,z)$

No need to choose: you can have both (Ecumenism)

Better: $A \leadsto_d \lambda z : El\,A, B$ can be replaced with $A \leadsto B$ each time z does not occur in B

Dependent implication

In the same way $\Rightarrow$ can be made dependent

$$\Rightarrow_d \,:\, \Pi x : \mathit{Prop}, (\mathit{Prf}\; x \rightarrow \mathit{Prop}) \rightarrow \mathit{Prop}$$
$$\mathit{Prf}\;(x \Rightarrow_d y) \longrightarrow \Pi z : \mathit{Prf}\; x, \mathit{Prf}\;(y\; z)$$

The Calculus of constructions

With $\Rightarrow_d$, $\rightsquigarrow_d$, $\forall$, and a similar symbol π
$(\langle *, *, * \rangle, \langle \Box, \Box, \Box \rangle, \langle \Box, *, * \rangle,$ and $\langle *, \Box, \Box \rangle)$
an expression of the Calculus of constructions

Reverse engineering proofs (Thiré)

A proof of Fermat's little theorem in Matita

- Express it Dedukti with $\Rightarrow_d$, $\leadsto_d$, $\forall$, and π
- Replace $\Rightarrow_d$, with $\Rightarrow$ and $\leadsto_d$ with $\leadsto$ when possible
- Remark that $\Rightarrow_d$, $\leadsto_d$, and π are not used anymore

A proof of Fermat's little theorem in Simple type theory (HOL4, HOL Light, Isabelle/HOL...)

IV. Predicate subtyping

$$psub : \Pi t : Set, (El\ t \to Prop) \to Set$$
$$pair : \Pi t : Set, \Pi p : El\ t \to Prop, \Pi m : El\ t, Prf\ (p\ m) \to El\ (psub\ t\ p)$$

$$pair^{\dagger} : \Pi t : Set, \Pi p : El\ t \to Prop, El\ t \to El\ (psub\ t\ p)$$
$$pair\ t\ p\ m\ h \longrightarrow pair^{\dagger}\ t\ p\ m$$

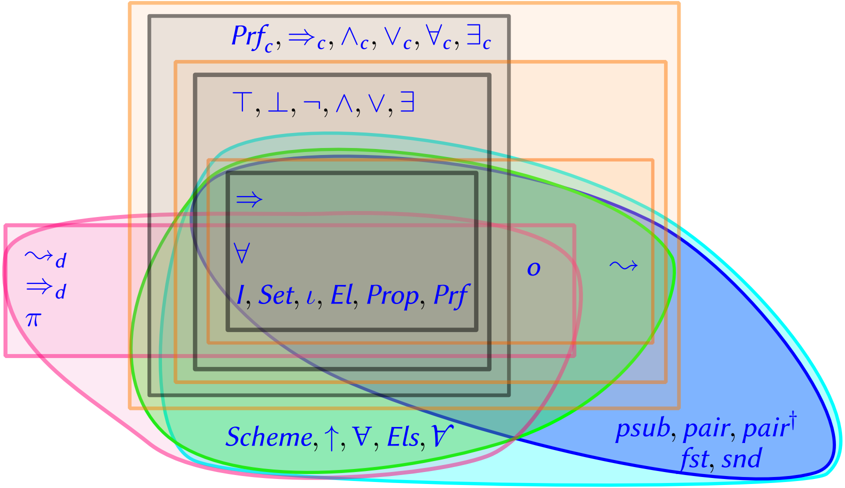
$$fst : \Pi t : Set, \Pi p : El\ t \to Prop, El\ (psub\ t\ p) \to El\ t$$
$$fst\ t\ p\ (pair^{\dagger}\ t'\ p'\ m) \longrightarrow m$$
$$snd : \Pi\ t : Set, \Pi p : El\ t \to Prop, \Pi m : El\ (psub\ t\ p), Prf\ (p\ (fst\ t\ p\ m))$$

$$(psub\ nat\ even) : Set$$
$$(pair\ nat\ even\ 6\ u) : (psub\ nat\ even)$$

PVS in Dedukti (Hondet)

How to express a theory in Dedukti?

Pick cherries according to your taste



Enough to express Predicate logic, Simple type theory, Simple type theory with predicate subtyping, The Calculus of constructions...

More advanced features in the next courses: universes, universe polymorphism, predicativity, inductive types...