Syntax for dependent type theories

Nicola Gambino

Leeds, February 20th, 2013

First-order theories vs dependent type theories (I)

Steps to set up a first-order theory:

(1) fix a language L,
(2) define inductively the set of well-formed terms,
(3) define inductively the set of well-formed formulas,
(4) give the axioms for the theory,
(5) define inductively the set of theorems of the theory.

Note. Each step depends only on the previous ones (e.g. the axioms of a theory do not change the set of well-formed terms).

First-order theories vs dependent type theories (II)

Problem. This approach does not work for dependent type theories:

- deduction rules of a dependent type theory specify how the well-formed term expressions are built, e.g.

$$\frac{a : A}{\mathsf{inl}(A, B, a) : A + B}$$

- the sets of term and type expressions cannot be defined independently, e.g.

$$\mathsf{Id}(A, a, b)\,, \quad \mathsf{nil}(A)$$

A solution. Define dependent type theories within meta-theories known as logical frameworks. But logical frameworks are complex and unfamiliar.

Setting up a dependent type theory

Other solution (Aczel)**:**

(1) fix a signature (*a notion that will be defined in the next slide*),
(2) define inductively the set of raw expressions,
(3) give the deduction rules of the dependent type theory,
(4) define inductively the set of theorems of the theory,
(5) the theorems isolate the well-formed expressions.

We will illustrate this in general and in one example.

Signatures for dependent type theories (I)

A signature for an algebraic theory (e.g. theory of groups) consists in:

- a set of operations,
- an assignment of an arity to each operation.

Here, an arity is just a natural number (the number of arguments of the operation).

For dependent type theories, we need more a complex notion of arity, to account for **variable binding** operations, e.g. λ , Π, Σ.

Note. Ongoing research on theories with variable binding (Pitts, Fiore, . . .).

Signatures for dependent type theories (II)

Definition. An **arity** is a tuple of the form

$$\big((n_1, \varepsilon_1), \ldots, (n_k, \varepsilon_k), \varepsilon\big)\,,$$

where $k \in \mathbb{N}$, $n_1, \ldots, n_k \in \mathbb{N}$ and $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon$ are either 0 or 1.

Notation. When $k = 0$, we write (ε).

Idea.

- An operation of arity as above takes k arguments and binds n_i variables in the i-th argument.
- The $\varepsilon_1, \ldots, \varepsilon_k, \varepsilon$ keep track of the distinction between term expressions (0-expressions) and type expressions (1-expressions).

Definition. A **signature** Σ is a set of pairs (s, α) where α is an arity. If $(s, \alpha) \in \Sigma$, we call s a **symbol of arity** α.

Raw expressions

Fix

- an infinite set of variables $\{x_0, x_1, \ldots\}$,
- a signature Σ.

Define the sets of 0-expressions and 1-expressions by the rules:

(i) every variable is a 0-expression,

(ii) if s has arity $((n_1, \varepsilon_1), \ldots, (n_k, \varepsilon_k), \varepsilon)$ and M_i is an ε_i-expression and $\vec{x}_i$ is a vector of n_i distinct variables for $i = 1, \ldots, k$, then

$$s\big((\vec{x}_1)M_1, \ldots, (\vec{x}_k)M_k\big)$$

is an ε-expression.

Notation. When $k = 0$, we write s rather than $s(\,)$. Also, if some $n_i = 0$ we write M_i rather than $(\,)M_i$.

Example

The signature for the type theory $\mathbf{ML_1}$ has the symbols

- Nat of arity (1)
- succ of arity $((0,0),0)$
- nil of arity $((0,1),0)$
- λ of arity $((0,1),(1,1),(1,0),0)$

Thus, we have that

- Nat is a 1-expression
- $\mathsf{succ}(x)$ is a 0-expression
- $\mathsf{nil}(\mathsf{Nat})$ is a 0-expression
- $\lambda(\mathsf{Nat},(x)\mathsf{Nat},(x)x)$ is a 0-expression, usually written $(\lambda x : \mathsf{Nat})x$.

Judgements

A **judgement** has one of the following four forms:

- $A : \mathsf{type}$ ("A is a well-formed type")
- $A = B : \mathsf{type}$ ("A and B are definitionally equal well-formed types")
- $a : A$ ("a is a well-formed term of type A")
- $a = b : A$ ("a and b are definitionally equal well-formed terms of type A")

Here, A, B stand for 1-expressions and a, b for 0-expressions.

Contexts and hypothetical judgements

A **context** is a sequence of the form

$$x_0 : A_0 \, , x_1 : A_1 \, , \ldots , x_n : A_n$$

where $x_0, \ldots, x_n$ are distinct variables and $A_1, \ldots, A_n$ are 1-expressions.

Notation. When $n = 0$, we write $()$.

A **hypothetical judgement** has the form

$$\Gamma \vdash J$$

where Γ is a context and J is a judgement.

Notation. We write J instead of $() \vdash J$.

Deduction rules

A **deduction rule** has the form

$$\frac{\Gamma_1 \vdash J_1 \qquad \cdots \qquad \Gamma_n \vdash J_n}{\Gamma \vdash J}$$

where $\Gamma_1 \vdash J_1 \,, \ldots, \Gamma_n \vdash J_n \,, \Gamma \vdash J$ are hypothetical judgements.

Notation. When $n = 0$, we simply write $\Gamma \vdash J$.

A dependent type theory is given by specifying

- a signature,
- a set of deduction rules.

Derivability is defined in the standard way.

Well-formed expressions

Definition. Let T be a dependent type theory over a signature Σ.

(i) If $\Gamma \vdash A : \mathsf{type}$ is derivable, then we say that A is a well-formed type in context Γ.

(ii) If $\Gamma \vdash a : A$ is derivable, then we say that a is a well-formed element of type A in context Γ.

(iii) If $\Gamma \vdash A = B : \mathsf{type}$, then we say that A and B are definitionally equal well-formed types in context Γ.

(iv) If $\Gamma \vdash a = b : A$, then we say that a and b are definitionally equal well-formed elements of type A in context Γ.

The dependent type theory $\mathbf{ML_1}$

A dependent type theory with the following forms of type:

$$\mathsf{Bool}\,, \quad \mathsf{Nat}\,, \quad \mathsf{Empty}\,, \quad \mathsf{List}(A) \quad A + B\,,$$
$$\mathsf{Id}(A, a, b)\,, \quad (\Pi x : A)B\,, \quad (\Sigma x : A)B\,, \quad \mathsf{U}$$

Note.

- **ML** stands for Martin-Löf
- The subscript 1 indicates the presence of rules for one type universe

The signature of $\mathbf{ML_1}$ (I)

Symbol	*Arity*
$\mathsf{Bool}, \mathsf{Nat}, \mathsf{Empty}, \mathsf{U}$	(1)
List	$((0,1),1)$
$+$	$((0,1),(0,1),1)$
Id	$((0,1),(0,0),(0,0),1)$
Π	$((0,1),(1,1),1)$
Σ	$((0,1),(1,1),1)$
T	$((0,0),1)$

Notation. We will write

- $A + B$ for $+(A, B)$
- $(\Pi x : A)B$ for $\Pi(A, (x)B)$
- $(\Sigma x : A)B$ for $\Sigma(A, (x)B)$.

The signature of $\mathbf{ML_1}$ (II)

The other symbols of the signature:

true	J
false	λ
boolrec	app
0	pair
succ	split
natrec	Bool_U
nil	Nat_U
cons	$\mathsf{Empty}_\mathsf{U}$
listrec	List_U
inl	$+_\mathsf{U}$
inr	Id_U
case	Π_U
refl	Σ_U

The deduction rules of $\mathbf{ML_1}$

(1) General rules

(2) Rules for the forms of type of $\mathbf{ML_1}$. For each one, we have

- formation rules
- introduction rules
- elimination rules
- computation rules

Note.

- When stating a rule, we omit contexts that are common to premisses and conclusion.
- Sometimes deduction rules are given as schemes.

General deduction rules

Standard rules regarding context formation, substitution, equality.

Examples.

$$\frac{\Gamma, \Delta \vdash J \qquad \Gamma \vdash A : \mathsf{type}}{\Gamma, x : A, \Delta \vdash J} \; x \notin \mathrm{FV}(\Gamma) \cup \mathrm{FV}(\Delta)$$

$$\frac{x : A, \Gamma \vdash J \qquad a : A}{\Gamma[a/x] \vdash J[a/x]}$$

$$\frac{a : A \quad A = B : \mathsf{type}}{a : B}$$

The type of Boolean truth values (I)

Formation rule.

$$\mathsf{Bool} : \mathsf{type}$$

Introduction rules.

$$\mathsf{true} : \mathsf{Bool} \qquad \mathsf{false} : \mathsf{Bool}$$

The type of Boolean truth values (II)

Elimination rules.

$$\frac{c : \mathsf{Bool} \quad x : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/x] \quad e : E[\mathsf{false}/x]}{\mathsf{boolrec}(c, (x)E, d, e) : E[c/x]}$$

Computation rules.

$$\frac{x : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/x] \quad e : E[\mathsf{false}/x]}{\mathsf{boolrec}(\mathsf{true}, (x)E, d, e) = d : E[\mathsf{true}/x]}$$

$$\frac{x : \mathsf{Bool} \vdash E : \mathsf{type} \quad d : E[\mathsf{true}/x] \quad e : E[\mathsf{false}/x]}{\mathsf{boolrec}(\mathsf{false}, (x)E, d, e) = e : E[\mathsf{false}/x]}$$

The type of natural numbers (I)

Formation rule.

$$\mathsf{Nat} : \mathsf{type}$$

Introduction rules.

$$0 : \mathsf{Nat} \qquad \frac{n : \mathsf{Nat}}{\mathsf{succ}(n) : \mathsf{Nat}}$$

The type of natural numbers (II)

Elimination rule.

$$\frac{c : \mathsf{Nat} \quad x : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[0/x] \quad x : \mathsf{Nat}, y : E \vdash e : E[\mathsf{succ}(x)/x]}{\mathsf{natrec}(c, (x)E, d, (x, y)e) : E[c/x]}$$

Computation rules.

$$\frac{x : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[0/x] \quad x : \mathsf{Nat}, y : E \vdash e : E[\mathsf{succ}(x)/x]}{\mathsf{natrec}(0, (x)E, d, (x, y)e) = d : E[0/x]}$$

$$\frac{c : \mathsf{Nat} \quad x : \mathsf{Nat} \vdash E : \mathsf{type} \quad d : E[0/x] \quad x : \mathsf{Nat}, y : E \vdash e : E[\mathsf{succ}(x)/x]}{\mathsf{natrec}(\mathsf{succ}(c), (x)E, d, (x, y)e) = e[c/x, \mathsf{natrec}(c, (x)E, d, (x, y)e)/y] : E[\mathsf{succ}(c)/x]}$$

The empty type

Formation rule.

$$\mathsf{Empty} : \mathsf{type}$$

Elimination rule.

$$\frac{c : \mathsf{Empty} \quad x : \mathsf{Empty} \vdash E : \mathsf{type}}{\mathsf{emptyrec}(c, (x)E) : E[c/x]}$$

Types of lists (I)

Formation rule.

$$\frac{A : \mathsf{type}}{\mathsf{List}(A) : \mathsf{type}}$$

Introduction rules.

$$\frac{A : \mathsf{type}}{\mathsf{nil}(A) : \mathsf{List}(A)} \qquad \frac{\ell : \mathsf{List}(A) \quad a : A}{\mathsf{cons}(A, \ell, a) : \mathsf{List}(A)}$$

Types of lists (II)

Elimination rule.

$$\frac{\begin{array}{l}\ell : \mathsf{List}(A)\\ x : \mathsf{List}(A) \vdash E : \mathsf{type}\\ d : E[\mathsf{nil}/x]\\ x : \mathsf{List}(A), y : A, z : E \vdash e : E[\mathsf{cons}(x,y)/x]\end{array}}{\mathsf{listrec}(\ell, (x)E, d, (x,y,z)e) : E[\ell/x]}$$

Computation rules.

$$\frac{\begin{array}{l}x : \mathsf{List}(A) \vdash E : \mathsf{type}\\ d : E[\mathsf{nil}/x]\\ x : \mathsf{List}(A), y : A, z : E \vdash e : E[\mathsf{cons}(x,y)/x]\end{array}}{\mathsf{listrec}(\mathsf{nil}, (x)E, d, (x,y,z)e) = d : E[\mathsf{nil}/x]}$$

$$\frac{\begin{array}{l}\ell : \mathsf{List}(A)\\ a : A\\ x : \mathsf{List}(A) \vdash E : \mathsf{type}\\ d : E[\mathsf{nil}/x]\\ x : \mathsf{List}(A), y : A, z : E \vdash e : E[\mathsf{cons}(x,y)/x]\end{array}}{\begin{array}{l}\mathsf{listrec}(\mathsf{cons}(\ell,a), (x)E, d, (x,y,z)e) =\\ \quad e[\ell/x, a/y, \mathsf{listrec}(\ell, (x)E, d, (x,y,z)e/z] : E[\mathsf{cons}(\ell,a)/x]\end{array}}$$

Sum types (I)

Formation rule.

$$\frac{A : \mathsf{type} \quad B : \mathsf{type}}{A + B : \mathsf{type}}$$

Introduction rules.

$$\frac{a : A}{\mathsf{inl}(A, B, a) : A + B} \qquad \frac{b : B}{\mathsf{inr}(A, B, b) : A + B}$$

Sum types (II)

Elimination rule.

$$\frac{c : A + B \quad z : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/z] \quad y : B \vdash e : E[\mathsf{inr}(y)z]}{\mathsf{case}(c, (z)E, (x)d, (y)e) : E[c/z]}$$

Computation rules.

$$\frac{a : A \quad z : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/z] \quad y : B \vdash e : E[\mathsf{inr}(y)z]}{\mathsf{case}(\mathsf{inl}(a), (z)E, (x)d, (y)e) = d[a/x] : E[\mathsf{inl}(a)/z]}$$

$$\frac{b : B \quad z : A + B \vdash E : \mathsf{type} \quad x : A \vdash d : E[\mathsf{inl}(x)/z] \quad y : B \vdash e : E[\mathsf{inr}(y)z]}{\mathsf{case}(\mathsf{inr}(b), (z)E, (x)d, (y)e) = e[b/y] : E[\mathsf{inr}(b)/z]}$$

Identity types (I)

Formation rule.

$$\frac{A : \mathsf{type} \quad a : A \quad b : A}{\mathsf{Id}(A, a, b) : \mathsf{type}}$$

Introduction rule.

$$\frac{a : A}{\mathsf{refl}(A, a) : \mathsf{Id}(A, a, a)}$$

Identity types (II)

Elimination rule.

$$\frac{p : \mathsf{Id}(a,b) \quad x : A, y : A, u : \mathsf{Id}(x,y) \vdash E : \mathsf{type} \quad x : A \vdash d : E[x/y, \mathsf{refl}(x)/u]}{\mathsf{J}(a,b,p,(x,y,u)d) : E[a/x, b/y, p/u]}$$

Computation rule.

$$\frac{a : A \quad x : A, y : A, u : \mathsf{Id}(x,y) \vdash E : \mathsf{type} \quad x : A \vdash d : E[x/y, \mathsf{refl}(x)/u]}{\mathsf{J}(a,a,\mathsf{refl}(a),(x,y,u)d) = d[a/x] : E[a/y, \mathsf{refl}(a)/u]}$$

Dependent product types (I)

Formation rule.

$$\frac{x : A \vdash B : \mathsf{type}}{(\Pi x : A)B : \mathsf{type}}$$

Introduction rule.

$$\frac{x : A \vdash b : B}{(\lambda x : A)b : (\Pi x : A)B}$$

Notation. Here $(\lambda x : A)b$ abbreviates $\lambda(A, (x)B, (x)b)$.

Special case. If $x \notin \mathrm{FV}(B)$, write $A \to B$ for $(\Pi x : A)B$.

Dependent product types (II)

Elimination rule.

$$\frac{f : (\Pi x : A)B \qquad a : A}{\mathsf{app}(f, a) : B[a/x]}$$

Computation rule.

$$\frac{x : A \vdash b : B \qquad a : A}{\mathsf{app}((\lambda x : A)b, a) = b[a/x] : B[a/x]}$$

Dependent sum types (I)

Formation rule.

$$\frac{x : A \vdash B : \mathsf{type}}{(\Sigma x : A)B : \mathsf{type}}$$

Introduction rule.

$$\frac{a : A \qquad b : B[a/x]}{\mathsf{pair}(a, b) : (\Sigma x : A)B : \mathsf{type}}$$

Special case. If $x \notin \mathrm{FV}(B)$, write $A \times B$ for $(\Sigma x : A)B$.

Dependent sum types (II)

Elimination rule.

$$\frac{c : (\Sigma x : A)B(x) \quad z : (\Sigma x : A)B(x) \vdash E : \mathsf{type} \quad x : A, y : B \vdash d : E[\mathsf{pair}(x, y)/z]}{\mathsf{split}(c, (z)E, (x, y)d) : E[c/z]}$$

Computation rule.

$$\frac{a : A \quad b : B[a/x] \quad z : (\Sigma x : A)B(x) \vdash E : \mathsf{type} \quad x : A, y : B \vdash d : E[\mathsf{pair}(x, y)/z]}{\mathsf{split}(\mathsf{pair}(a, b), (z)E, (x, y)d) = d[a/x, b/y] : E[\mathsf{pair}(a, b)/z]}$$

A type universe (I)

Formation rule.

$$\mathsf{U} : \mathsf{type}$$

Elimination rule.

$$\frac{A : \mathsf{U}}{\mathsf{T}(A) : \mathsf{type}}$$

A type universe (II)

Introduction and computation rules.

$$\mathsf{Bool}_\mathsf{U} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Bool}_\mathsf{U}) = \mathsf{Bool} : \mathsf{type}$$

$$\mathsf{Nat}_\mathsf{U} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Nat}_\mathsf{U}) = \mathsf{Nat} : \mathsf{type}$$

$$\mathsf{Empty}_\mathsf{U} : \mathsf{U} \qquad \mathsf{T}(\mathsf{Empty}_\mathsf{U}) = \mathsf{Empty} : \mathsf{type}$$

A type universe (III)

Introduction and computation rules.

$$\frac{A : \mathsf{U}}{\mathsf{List}_{\mathsf{U}}(A) : \mathsf{U}} \qquad \frac{A : \mathsf{U}}{\mathsf{T}(\mathsf{List}_{\mathsf{U}}(A)) = \mathsf{List}(\mathsf{T}(A)) : \mathsf{type}}$$

$$\frac{A : \mathsf{U} \quad B : \mathsf{U}}{A +_{\mathsf{U}} B : \mathsf{U}} \qquad \frac{A : \mathsf{U} \quad B : \mathsf{U}}{\mathsf{T}(A +_{\mathsf{U}} B) = \mathsf{T}(A) + \mathsf{T}(B) : \mathsf{type}}$$

A type universe (IV)

Introduction and computation rules.

$$\frac{A : \mathsf{U} \quad a : \mathsf{T}(A) \quad b : \mathsf{T}(A)}{\mathsf{Id}_{\mathsf{U}}(A, a, b) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \quad a : \mathsf{T}(A) \quad b : \mathsf{T}(A)}{\mathsf{T}(\mathsf{Id}_{\mathsf{U}}(A, a, b)) = \mathsf{Id}(\mathsf{T}(A), a, b) : \mathsf{type}}$$

$$\frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\Pi_{\mathsf{U}}(A, (x)B) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\mathsf{T}(\Pi_{\mathsf{U}}(A, (x)B)) = \Pi(\mathsf{T}(A), (x)\mathsf{T}(B)) : \mathsf{type}}$$

$$\frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\Sigma_{\mathsf{U}}(A, (x)B) : \mathsf{U}} \qquad \frac{A : \mathsf{U} \qquad x : \mathsf{T}(A) \vdash B : \mathsf{U}}{\mathsf{T}(\Sigma_{\mathsf{U}}(A, (x)B)) = \Sigma(\mathsf{T}(A), (x)\mathsf{T}(B)) : \mathsf{type}}$$