Dependently-Typed Termination and Embedding of Extensional Universe-Polymorphic Type Theory using Rewriting

Guillaume Genestier

Under the supervision of Frédéric Blanqui and Olivier Hermant

Thursday, December 10th, 2020

The Heart of Logic: Deduction

In logic, we derive **new** knowledge from **axioms** using **rules**.

$$\frac{\text{Socrates is a man} \quad \text{All men are mortal}}{\text{Socrates is mortal}} \qquad \frac{\vdash A \Rightarrow B \quad \vdash B \Rightarrow C}{\vdash A \Rightarrow C}$$

The Heart of Logic: Deduction

In logic, we derive **new** knowledge from **axioms** using **rules**.

$$\frac{\text{Socrates is a man} \quad \text{All men are mortal}}{\text{Socrates is mortal}} \checkmark \qquad \frac{\vdash A \Rightarrow B \quad \vdash B \Rightarrow C}{\vdash A \Rightarrow C}$$

$$\frac{\text{Socrates is mortal} \quad \text{All cats are mortal}}{\text{Socrates is a cat}} ✗$$

We expected Computer Science

Curry-Howard Correspondence

$$\frac{\sin : \mathbb{R} \to \mathbb{R} \qquad \_^2 : \mathbb{R} \to \mathbb{R}^+}{\lambda x. \sin(x)^2 : \mathbb{R} \to \mathbb{R}^+} \qquad \frac{\vdash A \Rightarrow B \qquad \vdash B \Rightarrow C}{\vdash A \Rightarrow C}$$

$$\frac{3 : \mathbb{R} \qquad \mathit{True} : \mathit{Bool}}{(3, \mathit{True}) : \mathbb{R} \times \mathit{Bool}} \qquad \frac{\vdash A \qquad \vdash B}{\vdash A \wedge B}$$

Type	Theorem
Program	Proof
Type Checking	Correctness Verification

Contents

Deduction alone is heavy

$*54{\cdot}43.\quad \vdash :.\ \alpha, \beta \in 1 \,.\supset :\ \alpha \cap \beta = \Lambda \,.\equiv .\ \alpha \cup \beta \in 2$

Dem.

$\vdash .\ *54{\cdot}26 \,.\supset \vdash :.\ \alpha = \iota`x \,.\, \beta = \iota`y \,.\supset :\ \alpha \cup \beta \in 2 \,.\equiv .\ x \neq y \,.$

$[*51{\cdot}231] \qquad \equiv .\ \iota`x \cap \iota`y = \Lambda \,.$

$[*13{\cdot}12] \qquad \equiv .\ \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash .\ (1) \,.\, *11{\cdot}11{\cdot}35 \,.\supset$

$\vdash :.\ (\exists x, y) \,.\, \alpha = \iota`x \,.\, \beta = \iota`y \,.\supset :\ \alpha \cup \beta \in 2 \,.\equiv .\ \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash .\ (2) \,.\, *11{\cdot}54 \,.\, *52{\cdot}1 \,.\supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Alfred N. Whitehead and Bertrand Russel, *Principia Mathematica*, Volume I, p. 379

Dedukti

[Boespflug, 2011] [Saillard, 2015]

Dedukti is a type-checker for the λΠ-calculus modulo rewriting.

Example of dependent type

```
symbol F : ℕ ⇒ TYPE
[]  F 0     ↪ ℕ
[n] F (s n) ↪ ℕ ⇒ F n
```

F n = ℕ ⇒ ℕ ⇒ ... ⇒ ℕ with n arrows.

Example of rewriting rules

Example : sum 5 1 2 3 4 5 $\longrightarrow^*$ 1+2+3+4+5 $\longrightarrow^*$ 15

```
symbol sum : (n: ℕ) ⇒ F n
[]  sum 0         ↪ 0
[]  sum (s 0)     ↪ λx, x
[n] sum (s (s n)) ↪ λx y, sum (s n) (plus x y)
```

The Purpose of a Logical Framework

Better Understanding of Theories

LF + A few symbols and rewrite rules

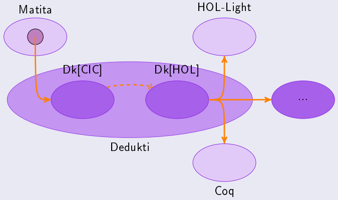
Allow Comparison between Logics

Same feature = Same rewrite rules

Easier to Analyze Proofs

Syntactical analysis of the symbols used

Translation between Logics

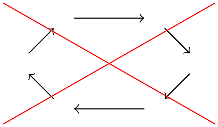


Non-restrictive Rewriting

- overlapping: $x + 0 \hookrightarrow x,\ 0 + x \hookrightarrow x$
- non-linearity: $x - x \hookrightarrow 0$
- defined symbols: $(x + y) + z \hookrightarrow x + (y + z)$
- higher-order: $lam\,(\lambda x.app\,F\,x) \hookrightarrow F$
- there can be rules both at the object and type levels

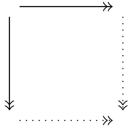
Expected Properties of Rewriting

- Termination: There is no infinite sequence of reduction starting from a well-typed term;



- Typing preservation (*Subject reduction*): If a term is well-typed, its reducts have the same type;

- Confluence: Two reducts of a term have a common reduct.



Expected Properties of Rewriting

- Termination: There is no infinite sequence of reduction starting from a well-typed term;
  - We assume preservation of sorts by reduction.
  - We assume local confluence.
- Typing preservation (*Subject reduction*): If a term is well-typed, its reducts have the same type;
- Confluence: Two reducts of a term have a common reduct.

Contents

Main Goal

If $\Gamma \vdash t : T$, then $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$.

Main Goal

Find a criterion such that:

If $\Gamma \vdash t : T$, then $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$.

Goal

Define $[\![T]\!]$ such :

- $\Gamma \vdash t : T$ implies $t \in [\![T]\!]$,

  Under conditions

- $t \in [\![T]\!]$ implies $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$.

Our Interpretation

- Pre-interpretation of type values,
- Interpretation of the sort $\star$ and of the types simultaneously,
- Interpretation of the sort $\square$ and of kinds.

Adequacy

Lemma (Adequacy)

If for all f, $f \in [\![\Theta_f]\!]$ and $\Gamma \vdash t : T$, then $t \in [\![T]\!]$.

Goal

Define $[\![T]\!]$ such that:

- $\Gamma \vdash t : T$ implies $t \in [\![T]\!]$, Condition: $\forall f, f \in [\![\Theta_f]\!]$
- $t \in [\![T]\!]$ implies $t \in \mathrm{SN}(\rightarrow_{\beta\mathcal{R}})$. ✓

Dependency Pairs

Definition (Dependency Pairs)

A rule $f\,\bar{l} \rightarrow r$ gives rise to the *dependency pairs* $f\,\bar{l} > g\,\bar{m}$ where:

- g is (partially) defined by rewriting,
- $g\,\bar{m}$ is a maximally applied subterm of r.

Theorem (Arts and Giesl, 2000)

First-order case:
$\rightarrow_{\mathcal{R}}$ *terminates iff there is no* $f\,\bar{t} > g\,\bar{u} \rightarrow^*_{arg} g\,\bar{u}' > \ldots$

Higher-Order Case

Static and dynamic definitions: [Blanqui06][Kusakari, Sakai 07][Kop, van Raamsdonk 12][Kop, Fuhs 19]

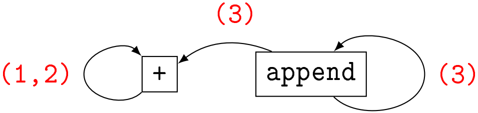
Example

```
symbol infix + : ℕ ⇒ ℕ ⇒ ℕ.
[q]   0     + q     ↪ q.
[p,q] (S p) + q     ↪ S (p + q). (1)
[p,q] p     + (S q) ↪ S (p + q). (2)

symbol append: (p: ℕ) ⇒ List p ⇒
               (q: ℕ) ⇒ List q ⇒ List (p + q).
[q,m]         append _ nil          q m ↪ m.
[x,p,l,q,m]   append _ (cons x p l) q m ↪
               cons x (p + q) (append p l q m). (3)
```

```
(1)              (S p) + q     > p + q
(2)              p     + (S q) > p + q
(3) append _ (cons x p l) q m  > append p l q m
(3) append _ (cons x p l) q m  > p + q
```

Call-Graph: Example

```
def plus : Nat -> Nat -> Nat.
set infix "+" := plus.
[q]     0     + q      --> q.
[p,q] (S p) + q      --> S (p + q). (1)
[p,q] p      + (S q) --> S (p + q). (2)

def append: (p: Nat) -> List p ->
            (q: Nat) -> List q -> List (p + q).
[q,m]       append _ nil           q m --> m.
[x,p,l,q,m] append _ (cons x p l) q m -->
             cons x (p + q) (append p l q m). (3)
```



Unaccessible Argument: Pure lambda-calculus

```
symbol Term  : ⋆.
symbol abstr : (Term ⇒ Term) ⇒ Term.
symbol app   : Term ⇒ Term ⇒ Term.
[f] app (abstr f) ⟶ f.
```

1^{st} argument of `abstr` is not accessible.

Accessible Argument: Brouwer Ordinals

```
symbol Nat : ⋆
symbol Ord : ⋆
  symbol 0 : Ord
  symbol s : Ord ⇒ Ord
  symbol lim : (Nat ⇒ Ord) ⇒ Ord

symbol ordrec : X ⇒ (Ord ⇒ X ⇒ X)
               ⇒ ((Nat ⇒ Ord) ⇒ (Nat ⇒ X) ⇒ X)
               ⇒ Ord ⇒ X
[x,y,z]   ordrec x y z 0        ↪ x
[x,y,z,o] ordrec x y z (s o)    ↪
                                y o (ordrec x y z o)
[x,y,z,f] ordrec x y z (lim f) ↪
                    z f (λ n, ordrec x y z (f n))
```

With Nat $\prec$ Ord, 1^{st} argument of lim is accessible.

Main Result

Reminder

If for all f, $f \in [\![\Theta_f]\!]$ and $\Gamma \vdash t : T$, then $t \in [\![T]\!]$.

Lemma

Every $f \in [\![\Theta_f]\!]$, if:

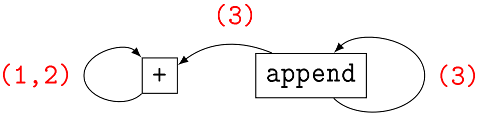
- *The right-hand side of every rule is well-typed,*
- *All variables occurring in a right-hand side are accessible in the left-hand side,*
- $(>\rightarrow^*_{arg})$ *is well-founded.*

Size-Change Termination : Example

Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].

Keeping track of the evolution of the sizes of the arguments:

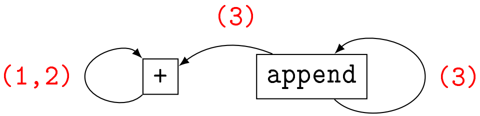
(1) `plus (S p) q > plus p q`	$\begin{array}{c} \\ \text{S p} \\ \text{q} \end{array} \begin{array}{c} \begin{array}{cc} \text{p} & \text{q} \end{array} \\ \begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix} \end{array}$
(3) `append n (cons x p l) q m > plus p q`	$\begin{array}{r} \\ \text{n} \\ \text{cons x p l} \\ \text{q} \\ \text{m} \end{array} \begin{array}{c} \begin{array}{cc} \text{p} & \text{q} \end{array} \\ \begin{pmatrix} \infty & \infty \\ < & \infty \\ \infty & = \\ \infty & \infty \end{pmatrix} \end{array}$



Size-Change Termination : Example

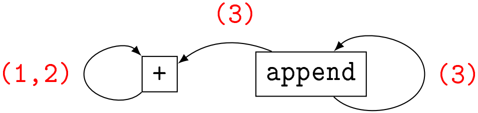
Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].
Keeping track of the evolution of the sizes of the arguments:

(1) `plus (S p) q > plus p q`	$\begin{array}{c|cc} & p & q \\ \hline S\ p & < & \infty \\ q & \infty & = \end{array}$
(3) `append n (cons x p l) q m > plus p q`	$\begin{array}{c|cc} & p & q \\ \hline n & \infty & \infty \\ cons\ x\ p\ l & < & \infty \\ q & \infty & = \\ m & \infty & \infty \end{array}$



Introduced in [Lee, Jones, Ben Amram, 02] and used for MLTT in [Wahlstedt07].
Keeping track of the evolution of the sizes of the arguments:

(1) `plus (S p) q > plus p q`	$\begin{array}{c c} & \begin{array}{cc} p & q \end{array} \\ \begin{array}{r} S\ p \\ q \end{array} & \begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix} \end{array}$
(3) `append n (cons x p l) q m > plus p q`	$\begin{array}{c c} & \begin{array}{cc} p & q \end{array} \\ \begin{array}{r} n \\ cons\ x\ p\ l \\ q \\ m \end{array} & \begin{pmatrix} \infty & \infty \\ < & \infty \\ \infty & = \\ \infty & \infty \end{pmatrix} \end{array}$



Main Result

$(>\to^*_{arg})$ is well-founded if the signature is finite and the set of rules is size-change terminating.

Theorem

$\to_{\beta\mathcal{R}}$ terminates on every typable term in $\lambda\Pi/\mathcal{R}$ if:

- *$\to_{\beta\mathcal{R}}$ is locally confluent and sort preserving,*
- *The right-hand side of every rule is well-typed,*
- *All variables occurring in a right-hand side are accessible in the left-hand side,*
- *The signature is finite,*
- *The set of rules is size-change terminating.*

Contents

Agda in a Nutshell

> *Agda is a dependently typed programming language. It is an extension of Martin-Löf's type theory [...].*
>
> *Because of strong typing and dependent types, Agda can be used as a proof assistant, allowing to prove mathematical theorems (in a constructive setting) and to run such proofs as algorithms.*
>
> — Agda user manual

The Agda we want to translate

- A *Predicative Pure Type System* with an *infinite hierarchy* of types,

$$\mathtt{zero} : \mathtt{Nat} : \mathsf{Set}_0 : \mathsf{Set}_1 : \cdots : \mathsf{Set}_i : \mathsf{Set}_{i+1} : \ldots$$

- with *inductive datatypes*.

```
data Nat : Set₀ where
  zero : Nat
  suc  : (m : Nat) → Nat
```

- with *η-equality* of functions and records,

$$f =_\eta \lambda x : A.f\ x$$

- with *universe polymorphism*,

```
data List (ℓ : Level) (A : Setℓ) : Setℓ where
  []   : List ℓ A
  _::_ : A → List ℓ A → List ℓ A
```

```
constant Univ : (l : Lvl) ⇒ TYPE
```

Agda	Dedukti	
Bool : $\mathtt{Set}_0$	Bool : Univ 0	✓
True : Bool	True : Bool	✗

Terms and types are different objects in Dedukti.

Encoding of Types

[Cousineau, Dowek, 2007]

```
constant Univ : (l : Lvl) ⇒ TYPE
```

Agda	Dedukti	
Bool : $\mathtt{Set}_0$	Bool : Univ 0	✓
True : Bool	True : Bool	✗

Terms and types are different objects in Dedukti.

```
symbol Lift : (l : Lvl) ⇒ (A : Univ l) ⇒ TYPE
```

True : Bool	True : Lift 0 Bool	✓

```
constant prod : (l1 : Lvl) ⇒ (l2 : Lvl)
              ⇒ (A : Univ l1)
              ⇒ (Lift l1 A ⇒ Univ l2)
              ⇒ Univ (max l1 l2)

[l1,l2,A,A] Lift _ (prod l1 l2 A B) ⟶
                (x : Lift l1 A) ⇒ Lift l2 (B x)
```

Agda	Dedukti
Bool → Bool : Set_0	prod 0 0 Bool (λ_,Bool) : Univ 0
neg : Bool → Bool	neg : Lift 0 (prod 0 0 Bool (λ_,Bool)) ↓ (Lift 0 Bool) ⇒ (Lift 0 Bool)

Encoding of Inductive Datatypes

The Agda declaration of natural numbers:

```
data Nat : Set₀ where
  zero : Nat
  suc  : (m : Nat) → Nat
```

is translated in Dedukti by:

```
constant Nat : Univ 0

constant Nat__zero: Lift 0 Nat
constant Nat__suc: Lift 0 (prod 0 0 Nat (λ_, Nat))
```

The Great Principle: Shape Preservation

Definitions in Agda are rewrite rules.

The Agda declaration of addition:

```
_+_  : Nat → Nat → Nat
zero     + n = n
(suc m) + n = suc (n + m)
```

is translated in Dedukti by:

```
symbol + : (Lift 0 Nat) ⇒ (Lift 0 Nat)
                         ⇒ (Lift 0 Nat)
[n]    + Nat__zero     n ⟶ n
[m,n]  + (Nat__suc m) n ⟶ Nat__suc (+ m n)
```

On η-conversion: The Time-Bomb Solution

Declare a symbol η_E in Dedukti, to annotate every subterm with its type, as long as it is required.

x : Nat	x
f : Nat $\rightarrow$ Nat	$\lambda x, f\ x$
$t : X$	η_E _ $X\ t \rightsquigarrow \ldots$

```
symbol ηE : (l : Lvl) ⇒
            (A : Univ l) ⇒
            Lift l A ⇒
            Lift l A.

[x]     ηE _ Nat              x ⟶ x
[n,B,f] ηE _ (prod _ n _ B) f ⟶
                                λx, ηE n (B x) (f x)
```

Contents

New Problem

Question

What is the type of $\forall \ell, \mathsf{Set}_\ell$?

New Problem

Question

What is the type of $\forall \ell, \mathsf{Set}_\ell$?

A brand new sort: Set_ω

- Not typable,
- Type of no sorts,
- On which we cannot quantify,
- For internal use only, not in the syntax.

Encoding Universe Polymorphism in Dedukti

```
constant ω : Lvl

constant ∀_Lvl : (s : (Lvl ⇒ Lvl)) ⇒
                 ((l : Lvl) ⇒ Univ (s l)) ⇒
                 Univ ω.

[s, b] Lift _ (∀_Lvl s b) ⟶
                  (l : Lvl) ⇒ Lift (s l) (b l)
```

Example ($\forall \ell, \mathrm{Set}_\ell$)

```
∀_Lvl (λl, succ l) (λl, univ l).
```

Definition (Translation of sorts)

$\langle \mathrm{Set}_\omega \rangle := \omega; \qquad \langle \mathrm{Set}_1 \rangle := \texttt{succ zero}.$

Definition (Translation of terms)

A well-typed term is translated by:

$$\begin{aligned}
\left|\lambda x^A.t\right| &:= \lambda x : \texttt{Lift}\langle s_A\rangle\ |A|\,.\,|t|; \\
|(x : A) \to B| &:= \texttt{prod}\ \langle s_A\rangle\ \langle s_B\rangle\ |A|\ (\lambda x : \texttt{Lift}\langle s_A\rangle\ |A|\,.\,|B|); \\
|\forall \ell, A| &:= \forall_{Lvl}\ (\lambda \ell : \texttt{Lvl}.\langle s_A\rangle)\ (\lambda \ell : \texttt{Lvl}.\,|A|).
\end{aligned}$$

A Good Encoding

Context for Prenex Universe Polymorphism

The global signature
$f : \forall[\ell_1, \ldots, \ell_n].A$

The level variables
ℓ

The local context
$x : A$

$$\Sigma; \Theta; \Gamma$$

Theorem (Soundness of the encoding)

Assuming $\langle . \rangle$ is such that equality of levels implies convertibility of their translations.
If P is a functional uniform universe polymorphic PTS, then

$$\Sigma; \Theta; \Gamma \vdash_P t : A : s \quad \textit{implies} \quad \texttt{Lift}\ |\Sigma; \Theta; \Gamma| \vdash_{Dk} |t| : \texttt{Lift}\ \langle s \rangle\ |A|\,.$$

The Convertibility Issue

Grammar of universe levels

$$t, u ::= x \in \mathcal{X} \mid 0 \mid \texttt{succ}\ t \mid \texttt{max}\ t\ u$$

Our goal

$$t \leftrightsquigarrow^* u \text{ if and only if } \forall \sigma : \mathcal{X} \to \mathbb{N}, [\![t]\!]_\sigma = [\![u]\!]_\sigma$$

The problems

For all $n > m$ and all σ,

$$[\![\texttt{max}\,(\texttt{succ}^n\,x)\,(\texttt{succ}^m\,x)]\!]_\sigma = [\![\texttt{succ}^n\,x]\!]_\sigma$$
$$[\![\texttt{max}\,(\texttt{succ}^n\,x)\,(\texttt{succ}^m\,0)]\!]_\sigma = [\![\texttt{succ}^n\,x]\!]_\sigma$$

We do not want an infinity of (non-linear) rewrite rules.

A New Hope

Reasoning Modulo AC

- for all t, t has a unique normal form (modulo associativity and commutativity),
- for all t and u in $\mathcal{L}$,

$$t{\downarrow} \ \equiv_{AC} \ u{\downarrow} \ \text{ if and only if } \qquad \forall \sigma : \mathcal{X} \to \mathbb{N}, [\![t]\!]_\sigma = [\![u]\!]_\sigma$$

Normal Forms

$$\mathrm{Max}\ i\ \ \{j_1 + x_1, j_2 + x_2, ...\}$$

where:

- $i, j_1, j_2, \ldots$ are closed natural numbers,
- $x_1, x_2, \ldots$ are distinct variables,
- for all k, $i \geqslant j_k$.

Sets Using Rewriting

Signature

```
symbol LvlSet : ⋆.

symbol ∅ : LvlSet.
symbol {_⊕_} : ℕ ⇒ Level ⇒ LvlSet.
associative-commutative _∪_ on LvlSet.
```

Rules on Union

```
[x]       x ∪ ∅ ↪ x.
[i,j,l] {i ⊕ l} ∪ {j ⊕ l} ↪ {(maxℕ i j) ⊕ l}.
```

Good Property of the Encoding

Implementation of the Syntax

```
[]     0          ⟶ Max 0ℕ ∅.
[x]    (succ x)   ⟶ Max 1ℕ {1ℕ ⊕ x}.
[x,y]  (max x y)  ⟶ Max 0ℕ ({0ℕ ⊕ x} ∪ {0ℕ ⊕ y}).
```

Proposition

The absence of variable of type $\mathbb{N}$ or `LvlSet` *ensures the uniqueness of normal form (modulo AC) property.*

Contents

Termination

A new termination criterion for higher-order rewriting with dependent types:

Theorem

$\to_{\beta\mathcal{R}}$ *terminates on every typable term in* $\lambda\Pi/\mathcal{R}$ *if:*

- $\to_{\beta\mathcal{R}}$ *is locally confluent and sort preserving,*
- *The right-hand side of all rules are well-typed,*
- *All variables occurring in a right-hand side are accessible in the left-hand side,*
- *The signature is finite,*
- *The set of rules is size-change terminating.*

Size-Change Tool

- An implementation of the criterion,
- First termination checker to combine higher-order rewriting and dependent types,
- Prove less simply-typed examples than Wanda and SOL, much faster,
- Very similar to Agda's termination checker on orthogonal rewriting rules.

Open-source, available on
`https://github.com/Deducteam/SizeChangeTool`

Future Work

Dependency Pairs

- Adapt more "processors",
- Recover completeness.

Modularity

Modularity results:

- with simple types (like [Harper, Honsell, Plotkin 93]),
- with first-order (like [Jouannaud, Okada97] and [Fuhs, Kop11]),

Non-termination

Analysis of the potential loop detected while constructing the size-change graph.

Agda2Dedukti

- Correct encoding of prenex universe polymorphism,
- Specialization to the universe levels of Agda.

A translator from Agda to Dedukti

Translate 29% of the standard library (162 files out of 562).

Open-source, freely available on:
`https://github.com/Deducteam/Agda2Dedukti`

Future Work

Complete the translator

- Co-inductive types,
- Sized-types,
- Proof irrelevant types,
- Primitive strings and floats,
- etc...

Interoperability

- Other way translator,
- Export proofs in this encoding to weaker logics.

Dependently-Typed Termination and Embedding of Extensional Universe-Polymorphic Type Theory using Rewriting

Guillaume Genestier

Under the supervision of Frédéric Blanqui and Olivier Hermant

Thursday, December 10th, 2020