Theory Comput Syst (2019) 63:647–665
https://doi.org/10.1007/s00224-018-9879-9

An Adequacy Theorem for Dependent Type Theory

Thierry Coquand[1] · **Simon Huber**[1]

Published online: 28 July 2018

Abstract We present a domain model of dependent type theory and use it to prove basic metatheoretic properties. In particular, we prove that two convertible terms have the same Böhm tree. The method used is reminiscent of the use of "inclusive predicates" in domain theory.

Keywords Dependent type theory · Domain theory · Finitary projections

1 Introduction

This paper has two main contributions. The first one is to present a domain model of dependent type theory where a type is interpreted as a finitary projection on one "universal" domain. We believe this model to be quite natural and canonical, and it can be presented as a simple *decidable* typing system on finite elements.[1] While this model is based on a "universal" domain, two convertible terms have the *same* semantics, like for the set-theoretic model [3]. This is to be contrasted with an "untyped"

[1]Finitary projections have already been used to model dependent type theory, e.g. in [5], but the observation that it can be presented as a decidable typing system on finite elements seems to be new.

This article is part of the Topical Collection on *Computer Science Symposium in Russia*

✉ Thierry Coquand
thierry.coquand@cse.gu.se

Simon Huber
simon.huber@cse.gu.se

[1] Department of Computer Science and Engineering, University of Gothenburg, 412 96 Göteborg, Sweden

semantics, like the one used in [1] and where one needs to quotient by an extra partial equivalence relation. The second contribution is to show, using this model, purely syntactical properties of dependent type theory. In particular, we can show that dependent product is one-to-one for conversion in a constructive metatheory, involving only induction and recursion on finite objects,[2] a property which is crucial in establishing subject reduction [4, 17]. Furthermore, the technique that is used is similar to the use of "inclusive predicates", fundamental in domain theory [12, 15]. Another technical advantage of our approach is that we don't need to use contexts as Kripke worlds as in previous arguments [2, 6]. We also establish that two convertible terms in type theory (maybe *partial* [10, 11, 13, 14]) have the same Böhm tree.

In this paper, we work in a constructive metatheory, and when we write that a propoosition P is decidable, we mean that $P \vee \neg P$ is provable.

2 Domain and Finite Elements

We shall use the following Scott domain, least solution of a recursive domain equation (see [18, 19] for a lively description of Scott domains and solutions to domain equations):

$$D \;=\; [D \to D] + \Pi\ D\ [D \to D] + \mathsf{N} + \mathsf{0} + \mathsf{S}\ D + \mathsf{U}_i$$

In this equation, $+$ denotes the coalesced sum [18] and $i = 0, 1, 2, \ldots$

We write $a, b, u, v, \ldots$ for the elements of this domain. We define $u(v)$ for u and v in D as follows: it is the application of u to v if u belongs to $D \to D$ and it is $\perp$ otherwise.

A fundamental result of domain theory is that the finite/compact elements of this domain can be described in a purely syntactical way, and both the *order* and the *compatibility* relations on these finite elements are *decidable* [16, 18, 19]. It also has been noticed [16] that this domain is *coherent* in the sense that a finite set is compatible (i.e. has a least upper bound) if, and only if, it is pairwise compatible.

Here is an inductive description of the finite elements

- $\perp$ or
- U_i, N or $\mathsf{0}$ or
- $\mathsf{S}\ u$ where u is finite
- $\Pi\ a\ f$ where a is finite and f is a finite function or
- a finite function

and a finite function f is a least upper bound of basic *step* functions and is of the form $\perp$ or $u_1 \mapsto v_1, \ldots, u_n \mapsto v_n$ (with $n \geqslant 1$ and all u_i, v_i finite) such that whenever u_i

[2] This is to be contrasted with existing proofs [2, 6] which so far require strong logical principles, like induction-recursion, contrary to what is expected for proving a purely syntactical property. The references [4, 17] are in a weak metatheory but do not cover η-conversion.

and u_j are compatible then so are v_i and v_j. Such a function sends an element u to the element $f(u) = \vee\{v_i \mid u_i \leqslant u\}$.

The order relation on finite elements can then be described by the rules

- $\bot \leqslant u$,
- $\mathsf{N} \leqslant \mathsf{N}$ and $\mathsf{0} \leqslant \mathsf{0}$ and $\mathsf{U}_i \leqslant \mathsf{U}_i$,
- $\mathsf{S}\ u \leqslant \mathsf{S}\ v$ if $u \leqslant v$,
- $\Pi\ a\ f \leqslant \Pi\ b\ g$ if $a \leqslant b$ and $f \leqslant g$, and
- $(u_1 \mapsto v_1, \ldots, u_p \mapsto v_p) \leqslant f$ if $v_i \leqslant f(u_i)$ for all i.

In general there are different possible ways to write a finite function f as a least upper bound of step functions. For instance, we have $(\bot \mapsto \mathsf{N}) = (\mathsf{U}_3 \mapsto \mathsf{N}, \bot \mapsto \mathsf{N})$. We say that a description $f = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ is *minimal* if we cannot remove some $u_i \mapsto v_i$ in this description. An important property is the following.

Lemma 1 *If $f = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ is minimal, we have $f\ u < f\ u_i$ whenever $u < u_i$.*

Proof If we have $u < u_i$ and $f\ u_i = f\ u$, then $\vee\{v_j \mid u_j < u_i\} \geqslant f\ u = f\ u_i$ and we can remove $u_i \mapsto v_i$ from the given description of f. □

Corollary 1 *If we have a minimal description of $f = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ and another description $f = (a_1 \mapsto b_1, \ldots, a_m \mapsto b_m)$ (not necessarily minimal), then $u_i = \vee\{a_j \mid a_j \leqslant u_i\}$.*

Proof Indeed, if $u = \vee\{a_j \mid a_j \leqslant u_i\}$ we have $u \leqslant u_i$ and $f(u) = \vee\{b_j \mid a_j \leqslant u\} = \vee\{b_j \mid a_j \leqslant u_i\} = f(u_i)$, so we cannot have $u < u_i$ by the previous lemma. □

We define the rank $rk(u)$ of the finite element u by the equations

$$rk(\bot) = 0 \qquad rk(\mathsf{N}) = rk(\mathsf{0}) = rk(\mathsf{U}_i) = 1 \qquad rk(\mathsf{S}\ u) = 1 + rk(u)$$

$$rk(\Pi\ u\ f) = \max(1 + rk(u), rk(f))$$

and $rk(f) = 1 + \max(rk(u_i), rk(f(u_i)))$ if $f = (u_1 \mapsto v_1, \ldots, u_l \mapsto v_l)$ is minimal and $l > 0$. The rank measures the first time an element u appears in the inductive generation of finite elements. An important property of the rank is that $rk(u \vee v) \leqslant \max(rk(u), rk(v))$ and $rk(f(u)) < rk(f)$ for all u.

Working with universes, we want to consider that U_i is more "complex" than any given finite element which only mentions U_j for $j < i$. In order to capture this notion of complexity, we define

$$lv(\bot) = lv(\mathsf{N}) = lv(\mathsf{0}) = 0 \qquad lv(\mathsf{U}_i) = i \qquad lv(\mathsf{S}\ u) = lv(u)$$

$$lv(\Pi\ u\ f) = \max(lv(u), lv(f))$$

and $lv(f) = \max(lv(u_i), lv(f(u_i)))$ if $f = (u_1 \mapsto v_1, \ldots, u_k \mapsto v_k)$ is minimal. An important property of the (universe) level is that $lv(u \vee v) \leqslant \max(lv(u), lv(v))$ and $lv(f(u)) \leqslant lv(f)$ for all u.

Finally we define the *complexity* of a finite element a as the pair $lv(a), rk(a)$ with the lexicographic ordering.

A *finitary projection* [18, 19] of a Scott domain E is a map $p : E \to E$ such that $p \circ p = p$ and $p\ a \leqslant a$ and the image of p, which is also the set of fixed-points of p, is a Scott domain. If $p\ u = u$ and $p\ v = v$ and u, v are compatible then $p\ (u \vee v) = u \vee v$ since both u and v are $\leqslant p\ (u \vee v)$. A finitary projection is thus completely determined by a set of finite elements which is closed by compatible sups. If F, E are two Scott domains, we write $F \lhd E$ and say that F is a *subdomain* of E if F is the image of a finitary projection of E. Equivalently F is the set of directed sups of a given subset of finite elements of E which is closed by compatible binary sups, and this set is exactly the set of finite elements of F. A fundamental result [18] is that the poset of finitary projections of a Scott domain E is itself a Scott domain, which is a subdomain of $E \to E$.

3 Concrete Description of the Typing Relation on Finite Elements

We now describe a *type system* on finite elements.

$$\frac{}{\bot : a} \qquad \frac{}{\mathsf{U}_i : \mathsf{U}_j} i < j \qquad \frac{}{\mathsf{N} : \mathsf{U}_j} \qquad \frac{}{\mathsf{0} : \mathsf{N}} \qquad \frac{u : \mathsf{N}}{\mathsf{S}\ u : \mathsf{N}}$$

$$\frac{a : \mathsf{U}_j \qquad u_1 : a \qquad t_1 : \mathsf{U}_j \qquad \ldots \qquad u_n : a \qquad t_n : \mathsf{U}_j}{\Pi\ a\ (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n) : \mathsf{U}_j} (n \geqslant 0)$$

$$\frac{u_1 : a \qquad v_1 : f(u_1) \qquad \ldots \qquad u_n : a \qquad v_n : f(u_n)}{(u_1 \mapsto v_1, \ldots, u_n \mapsto v_n) : \Pi\ a\ f} (n \geqslant 0)$$

Lemma 2 *If $u : a$ and $a \leqslant b$, then $u : b$. If $u : a$, $v : a$, and u and v are compatible, then $u \vee v : a$.*

Proof The first statement is by induction on the derivation of $u : a$. For the second statement, we look at the case where $a = \mathsf{U}_k$ and $u = \Pi\ b\ (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n)$ and $v = \Pi\ b'\ (v_1 \mapsto l_1, \ldots, v_m \mapsto l_m)$. By induction, we have $b \vee b' : \mathsf{U}_k$. Also $u_i : b$ and hence $u_i : b \vee b'$ by the first statement and similarly $v_j : b \vee b'$. The other cases are similar. □

Corollary 2 *If $w : \Pi\ a\ f$ and $u : a$, then $w(u) : f(u)$.*

Proof We can write $w = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ with $v_i : f(u_i)$. We have $v_i : f(u)$ if $u_i \leqslant u$ by Lemma 2. We then have $w(u) = \vee\{v_i \mid u_i \leqslant u\} : f(u)$ by Lemma 2. □

Lemma 3 *If $\Pi\ a\ f : \mathsf{U}_k$ and $f = (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n)$ is minimal, then $u_i : a$ and $f(u_i) : \mathsf{U}_k$.*

Proof We have $f = (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n) = (v_1 \mapsto l_1, \ldots, v_m \mapsto l_m)$ with $v_j : a$ and $l_j : \mathsf{U}_k$. Since $f(u_i) = \vee\{l_j \mid v_j \leqslant u_i\}$ we have $f(u_i) : \mathsf{U}_k$ by Lemma 2. Also $u_i = \vee\{v_j \mid v_j \leqslant u_i\}$ and so $u_i : a$ by Lemma 2. □

Lemma 4 *If $w : \Pi\ a\ f$ and $w = (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n)$ is minimal, then $u_i : a$ and $w(u_i) : f(u_i)$.*

Proof We have $w = (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n) = (v_1 \mapsto l_1, \ldots, v_m \mapsto l_m)$ with $v_j : a$ and $l_j : f(v_j)$. It follows from Corollary 1 that we have $u_i = \vee\{v_j \mid v_j \leqslant u_i\}$ and so $u_i : a$ by Lemma 2. Using Corollary 2, we get $w(u_i) : f(u_i)$. □

Note that if $u : a$ then $lv(u) \leqslant lv(a)$ and if $u : \mathsf{U}_k$ then $lv(u) < k$, by induction on the derivation.

Corollary 3 *The relation $u : a$ is decidable.*

Proof By induction on the complexity of u and a. □

The following Lemma will be useful when connecting syntax and semantics.

Lemma 5 *If $w : \Pi\ b\ f$ and $b \leqslant a$, then for any $u : a$ there exists $v : b$ such that $v \leqslant u$ and $w(u) = w(v)$.*

Proof We write $w = (u_1 \mapsto l_1, \ldots, u_n \mapsto l_n)$ with $u_i : b$ and $l_i : f(u_i)$. We then have $w(u) = w(v)$ with $v = \vee\{u_i \mid u_i \leqslant u\}$ and $v : b$ by Lemma 2. □

We now introduce the predicate a type by the rules:

$$\frac{\overline{\perp \text{ type}} \qquad \overline{\mathsf{U}_i \text{ type}} \qquad \overline{\mathsf{N} \text{ type}}}{a \text{ type} \quad u_1 : a \quad t_1 \text{ type} \quad \ldots \quad u_n : a \quad t_n \text{ type}}{\Pi\ a\ (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n) \text{ type}}\ (n \geqslant 0)$$

Lemma 6 *If $a : \mathsf{U}_j$, then a type. If a type, b type, and a, b are compatible, then $a \vee b$ type. If $\Pi\ a\ (u_1 \mapsto t_1, \ldots, u_n \mapsto t_n)$ type and $u_1 \mapsto t_1, \ldots, u_n \mapsto t_n$ is a minimal description, then $u_i : a$ and t_i type.*

Proof The first statement is by induction on the derivation of $a : \mathsf{U}_j$. The second statement is proved as in Lemma 2, and the last statement as in the proof of Lemma 3. □

Corollary 4 *The predicate a type is decidable.*

Given a finite element a, the set of finite elements u such that $u : a$ is closed by compatible binary sups by Lemma 2. Hence it defines a finitary projection $p\ a$. Similarly the set of finite elements a such that a type defines a finitary projection p_{type}. We write $\mathsf{Type} \lhd D$ for the corresponding subdomain.

By Lemma 2, we have $p\ a \leqslant p\ b$ if $a \leqslant b$ and we can hence define the finitary projection $p\ a$ for an *arbitrary* element a of D, not necessarily finite, as the directed sup of all $p\ a_0$ for $a_0 \leqslant a$ finite, in the Scott domain of finitary projections of D. We write $El\ a \lhd D$ for the image of $p\ a$.

We have $El\ \mathsf{U}_i \lhd El\ \mathsf{U}_{i+1}$ and $El\ \mathsf{U}_i \lhd \mathsf{Type}$.

Let us write $a \to b$ for $\Pi\ a\ (\bot \mapsto b)$. The domain $El\ \mathsf{N} \lhd D$ is exactly the domain of "lazy" natural numbers, that are elements of the form $\mathsf{S}^k\ \mathsf{0}$ or $\mathsf{S}^k \bot$. The poset of finite elements w such that $w : \mathsf{N} \to \mathsf{N}$ is exactly the poset of finite element of the domain of continuous functions $El\ \mathsf{N} \to El\ \mathsf{N}$.

Lemma 7 *If $f = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ is minimal and $f = f \circ p$ where p is a finitary projection, then $p\ u_i = u_i$ for all i.*

Proof We have $f\ u_i = f\ (p\ u_i)$ and so we cannot have $p\ u_i < u_i$ since the description is minimal using Corollary 1, and so $p\ u_i = u_i$. □

Proposition 1 *We have*

$$
\begin{array}{lcll}
p\ \mathsf{N}\ \mathsf{0} & = & \mathsf{0} & \\
p\ \mathsf{N}\ (\mathsf{S}\ u) & = & \mathsf{S}\ (p\ \mathsf{N}\ u) & \\
p\ (\Pi\ a\ f)\ w & = & x \mapsto p\ (f(p\ a\ x))\ (w(p\ a\ x)) & \\
p\ \mathsf{U}_j\ \mathsf{N} & = & \mathsf{N} & \\
p\ \mathsf{U}_j\ (\Pi\ a\ f) & = & \Pi\ (p\ \mathsf{U}_j\ a)\ ((p\ \mathsf{U}_j) \circ f \circ (p\ a)) & \\
p\ \mathsf{U}_j\ \mathsf{U}_i & = & \mathsf{U}_i & \textit{if}\quad i < j
\end{array}
$$

and $p\ a\ b = \bot$ in all other cases. We also have

$$
\begin{array}{lcl}
p_{\mathsf{type}}\ \mathsf{N} & = & \mathsf{N} \\
p_{\mathsf{type}}\ (\Pi\ a\ f) & = & \Pi\ (p_{\mathsf{type}}\ a)\ (p_{\mathsf{type}} \circ f \circ (p\ a)) \\
p_{\mathsf{type}}\ \mathsf{U}_i & = & \mathsf{U}_i
\end{array}
$$

and $p_{\mathsf{type}}\ b = \bot$ in all other cases.

Proof Let $q\ a$ be the function defined by these recursive equations. We show by induction on the complexity of a finite that we have $q\ a\ u = u$, for u finite, if,

and only if, $u : a$. This is clear if $a = \mathsf{N}$. If $a = \Pi\ b\ f$ and $u : a$, then using Lemma 4, we can write $u = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ with $u_i : a$ and $v_i : f(u_i)$. We then have $u(x) = u(q\ b\ x) : f\ (q\ b\ x)$ and so $u(x) = (q\ (\Pi\ b\ f)\ u)(x)$ for any x and so $u = q\ a\ u$. Conversely, if $u = q\ a\ u$, we have $u = u \circ (q\ b)$, and if $u = (u_1 \mapsto v_1, \ldots, u_n \mapsto v_n)$ is a minimal description of u, we have $q\ b\ u_i = u_i$ by Lemma 7. So $u_i : b$ by induction. We then have $v_i = q\ (f(u_i))\ v_i$ and so $v_i : f(u_i)$ by induction.

Finally we prove $q\ \mathsf{U}_k\ a = a$ if, and only if, $a : \mathsf{U}_k$ by induction on the complexity of a finite. We cover the case $a = \Pi\ b\ f$ where $u_1 \mapsto l_1, \ldots, u_n \mapsto l_n$ is a minimal description of f.

If $a : \mathsf{U}_k$, then $b : \mathsf{U}_k$ and so $q\ \mathsf{U}_k\ b = b$ by induction and $u_i : b$ and $l_i : \mathsf{U}_k$. Since b is strictly less complex than U_k, we have by induction $q\ b\ u_i = u_i$ and $q\ \mathsf{U}_k\ f(u_i) = f(u_i)$. It follows that we have $q\ \mathsf{U}_k\ a = a$.

Conversely, if $q\ \mathsf{U}_k\ a = a$ then $q\ \mathsf{U}_k\ b = b$ and so $b : \mathsf{U}_k$ by induction and we have $(q\ \mathsf{U}_k) \circ f \circ (q\ b) = f$. It follows that we have $f(u_i) = q\ \mathsf{U}_k\ (f(q\ b\ u_i))$ for all i and we have $q\ b\ u_i = u_i$ by Lemma 7. So $u_i : b$ since b is simpler than U_k. We then get $f(u_i) = q\ \mathsf{U}_k\ f(u_i)$ and so $f(u_i) : \mathsf{U}_k$ by induction. □

We can now consider the continuous families of domains $El\ a$ and $El\ a \to \mathsf{Type}$ indexed over a in Type. We can form their carteisan products and get a continuous family $El\ a \times (El\ a \to \mathsf{Type})$ indexed over a in Type. We consider then the sum of this family, which is itself a Scott domain [7]

$$E = \Sigma(a \in \mathsf{Type})\ (El\ a \times (El\ a \to \mathsf{Type}))$$

and we have an evaluation function $E \to \mathsf{Type}$, $(a, v, f) \longmapsto f(v)$. This evaluation function is continuous. So, if we have $w_0 \leqslant f(v)$ in Type then we can find $a_0 \leqslant a$ finite in Type, and $u_0 \leqslant u$ finite in $El\ a_0$ and $f_0 \leqslant f$ finite in $El\ a_0 \to \mathsf{Type}$ such that $w_0 \leqslant f(v)$. This remark will be used in a crucial way in connecting syntax and semantics of type theory.

4 Syntax and Semantics of Type Theory

The syntax of type theory is defined as follows.

$$M, N, A, B ::= x \mid \lambda(x : A)M \mid M\ N \mid \Pi(x : A)B \mid \mathsf{N} \mid \mathsf{U}_i \mid 0 \mid \mathsf{S}\ M \mid \mathsf{rec}(\lambda x\ A, M, N)$$

We write $F(x/M)$ the substitution of M for x in F. We may write simply $F(M)$ if x is clear from the context.

The semantics can be defined at this purely untyped syntactic level, exactly like for the set-theoretic semantics presented in [3]. This semantics is described in Fig. 1 where we define $\rho, x : a = u$ to be the update of ρ with the assignment $x = p\ a\ u$.

$$[\![x]\!]\rho = \rho(x) \qquad [\![M\ N]\!]\rho = [\![M]\!]\rho([\![N]\!]\rho) \qquad [\![\mathsf{N}]\!]\rho = \mathsf{N} \qquad [\![\mathsf{U}_i]\!]\rho = \mathsf{U}_i \qquad [\![0]\!]\rho = 0$$

$$[\![\mathsf{S}\ M]\!]\rho = \mathsf{S}\ ([\![M]\!]\rho) \qquad [\![\lambda(x:A)M]\!]\rho = u \mapsto [\![M]\!](\rho, x : [\![A]\!]\rho = u)$$

$$[\![\Pi(x:A)B]\!]\rho = \Pi\ ([\![A]\!]\rho)\ (u \mapsto [\![B]\!](\rho, x : [\![A]\!]\rho = u))$$

Fig. 1 Denotational semantics of type theory

The semantics of rec is the usual lazy semantics of primitive recursion. We define $\mathsf{rec}(d_0, d_1)$ in $D \to D$ by the recursive equations

$$\mathsf{rec}(d_0, d_1)\ 0 = d_0 \qquad \mathsf{rec}(d_0, d_1)\ (\mathsf{S}\ u) = d_1(u)(\mathsf{rec}(d_0, d_1)\ u)$$

and $\mathsf{rec}(d_0, d_1)\ u = \bot$ in the other cases, and then

$$[\![\mathsf{rec}(\lambda x.T, M_0, M_1)]\!]\rho = \mathsf{rec}([\![M_0]\!]\rho, [\![M_1]\!]\rho).$$

(The extra argument $\lambda x.T$ is used in Section 7.)

The typing and conversion rules are in the Appendix. There are two judgments for types, of the form $A\ \mathsf{type}$ and $A\ \mathsf{conv}\ A'$, and two judgments for elements, of the form $M : A$ and $M\ \mathsf{conv}\ M' : A$. Such a judgment is stated in a *context*, which is a list of typing declarations $x : A$. As in [9], we may not write the context explicitly.

We say that ρ *fits* Γ if for all $x : A$ in Γ we have $[\![A]\!]\rho$ in Type and $\rho(x)$ in $El([\![A]\!]\rho)$.

Theorem 1 *If ρ fits Γ, then:*

1. $\Gamma \vdash A\ \mathsf{type}$ *implies* $[\![A]\!]\rho \in \mathsf{Type}$,
2. $\Gamma \vdash A\ \mathsf{conv}\ A'$ *implies* $[\![A]\!]\rho = [\![A']\!]\rho$,
3. $\Gamma \vdash M : A$ *implies* $[\![M]\!]\rho \in El\ ([\![A]\!]\rho)$, *and*
4. $\Gamma \vdash M\ \mathsf{conv}\ M' : A$ *implies* $[\![M]\!]\rho = [\![M']\!]\rho$.

Proof Direct by induction on the derivation. □

Note that the use of finitary projections takes care of η-conversion in the semantics. For instance, we have $[\![\lambda(x : \mathsf{N} \to \mathsf{N})x]\!] = [\![\lambda(x : \mathsf{N} \to \mathsf{N})\lambda(y : \mathsf{N})x\ y]\!]$. Indeed, both are equal to the function $u \mapsto v \mapsto p\ \mathsf{N}\ (u(p\ \mathsf{N}\ v))$.

The main difference with the semantics suggested in [11] and in [13] is that abstraction is not interpreted as a constructor. This is crucial in order to validate the rule of η-conversion that $N\ \mathsf{conv}\ N' : \Pi(x : A)B$ as soon as $N\ x\ \mathsf{conv}\ N'\ x : B\ (x : A)$. If we represent abstraction by a constructor, we would have $w = \lambda(\bot) \neq \bot = w'$ but also $w(u) = \bot = w'(u)$ for any u in D, and so the rule for η-conversion cannot be valid in this case.

5 Connecting Syntax and Semantics, First Version

We write $M \to M'$ for weak-head reduction. This is defined at a purely syntactical level. The rules are the following.

$$\frac{}{(\lambda(x : A)N)\ M \to N(x/M)} \qquad \frac{N \to N'}{N\ M \to N'\ M}$$

$$\frac{}{\mathsf{rec}(\lambda x\ T, M_0, M_1)\ 0 \to M_0}$$

$$\frac{}{\mathsf{rec}(\lambda x\ T, M_0, M_1)\ (\mathsf{S}\ N) \to M_1\ N\ (\mathsf{rec}(\lambda x\ T, M_0, M_1)\ N)}$$

$$\frac{N \to N'}{\mathsf{rec}(\lambda x\ T, M_0, M_1)\ N \to \mathsf{rec}(\lambda x\ T, M_0, M_1)\ N'}$$

We write $M \to_A M'$ for $M \to M'$ and $M\ \mathsf{conv}\ M' : A$ and write $M \to_A^* M'$ for the corresponding transitive reflexive closure. We write $A \to_{\mathsf{type}} A'$ to mean that $A \to A'$ and $A\ \mathsf{conv}\ A'$, and we write $A \to_{\mathsf{type}}^* A'$ the corresponding transitive reflexive closure. These relations are similar to the relations used in [2, 6].

In this section, we will consider only *closed* terms. The relation $A\ \mathsf{conv}\ B$ defines an equivalence relation on the set of terms A such that $A\ \mathsf{type}$. If $A\ \mathsf{type}$, then, similarly, the relation $M\ \mathsf{conv}\ N : A$ defines an equivalence relation on the set of terms M satisfying $M : A$.

The main goal of this section is to analyze relations refining these predicates. We define $A\ \mathsf{type}\ |_a$ and $A\ \mathsf{conv}\ B\ |_a$ for $a \leqslant [\![A]\!]$ in Type and, if we have $A\ \mathsf{type}\ |_a$, we define $M : A\ |_{u:a}$ and $M\ \mathsf{conv}\ M' : A\ |_{u:a}$ for $u \leqslant [\![M]\!]$ in $El\ a$. The relation $A\ \mathsf{conv}\ B\ |_a$ will be an equivalence relation on the set of terms satisfying the predicate $|_a$, while, if $A\ \mathsf{type}\ |_a$, the relation $M\ \mathsf{conv}\ M' : A\ |_{u:a}$ will be an equivalence relation on the set of terms M such that $M : A\ |_{u:a}$.

These relations are defined first for *finite* elements a, by *recursion* on the complexity of a in Type. More precisely, we define all the relations $A\ \mathsf{type}\ |_a$, $A\ \mathsf{conv}\ B\ |_a$, $M : A\ |_{u:a}$, and $M\ \mathsf{conv}\ M'\ |_{u:a}$ by recursion on the complexity of the finite element a. In particular, this definition is *not* an inductive-recursive one. To incorporate universes we also define at the same time the relations $A\ \mathsf{type}\ |_a^i$ and $A\ \mathsf{conv}\ A'|_a^i$ for $i = 0, 1, 2, \ldots$ in the clauses 5-8. In each of the clauses of the definition below we will have some tacit assumptions suppressed for readability: $A\ \mathsf{type} \mid a$ assumes $A\ \mathsf{type}$ and $a\ \mathsf{type}$; $A\ \mathsf{conv}\ A'\ |_a$ assumes both $A\ \mathsf{type}\ |_a$ and $A'\ \mathsf{type}\ |_a$ and further $A\ \mathsf{conv}\ A'$; $M : A\ |_{u:a}$ assumes $A\ \mathsf{type}\ |_a$, $M : A$, and $u : a$; $M\ \mathsf{conv}\ M' : A\ |_{u:a}$ assumes $M : A\ |_{u:a}$, $M' : A\ |_{u:a}$, and $M\ \mathsf{conv}\ M' : A$.

We distinguish the shape of a.

1. *Case* $\bot$. $A\ \mathsf{type}\ |_\bot$, $A\ \mathsf{conv}\ A'\ |_\bot$, $M : A\ |_{u:\bot}$, and $M\ \mathsf{conv}\ M'\ |_{u:\bot}$ all hold by definition.
2. *Case* $\Pi\ b\ f$. We define:
   - $A\ \mathsf{type}\ |_{\Pi\ b\ f}$ means $A \to_{\mathsf{type}}^* \Pi(x : B)F$ for some B and F with $B\ \mathsf{type}\ |_b$ and $x : B \vdash F\ \mathsf{type}$ and

(a) $N : B \mid_{v:b}$ implies $F(N)$ type $\mid_{f(v)}$, and
(b) N conv $N' : B \mid_{v:b}$ implies $F(N)$ conv $F(N') \mid_{f(v)}$.

– Given A' with $A' \to^*_{\mathsf{type}} \Pi(x : B')F'$ and A as above, then A conv $A' \mid_{\Pi\, b\, f}$ means B conv $B' \mid_b$ and

$$N : B \mid_{v:b} \quad \text{implies} \quad F(N) \text{ conv } F'(N) \mid_{f(v)} .$$

– $M : A \mid_{u:\Pi\, b\, f}$ is defined as (A as above):

(a) $N : B \mid_{v:b}$ implies $M\ N : F(N) \mid_{u(v):f(v)}$, and
(b) N conv $N' : B \mid_{v:b}$ implies $M\ N$ conv $M\ N' : F(N) \mid_{u(v):f(v)}$.

– M conv $M' : A \mid_{u:\Pi\, b\, f}$ is defined as

$$N : B \mid_{v:b} \quad \text{implies} \quad M\ N \text{ conv } M'\ N : F(N) \mid_{u(v):f(v)} .$$

3. *Case* N. We define:

– A type $\mid_{\mathsf{N}}$ means that $A \to^*_{\mathsf{type}} \mathsf{N}$.
– A conv $A' \mid_{\mathsf{N}}$ is always satisfied.
– $M : A \mid_{u:\mathsf{N}}$ is defined by induction on u:

(a) $M : A \mid_{\perp:\mathsf{N}}$ holds by definition,
(b) $M : A \mid_{0:\mathsf{N}}$ if $M \to^*_{\mathsf{N}} 0$, and
(c) $M : A \mid_{\mathsf{S}\, v:\mathsf{N}}$ if $M \to^*_{\mathsf{N}} \mathsf{S}\ N$ and $N : A \mid_{v:\mathsf{N}}$.

– M conv $M' : A \mid_{u:\mathsf{N}}$ is defined by induction on u:

(a) M conv $M' : A \mid_{\perp:\mathsf{N}}$ and M conv $M' : A \mid_{0:\mathsf{N}}$ hold by definition, and
(b) M conv $M' : A \mid_{\mathsf{S}\, v:\mathsf{N}}$ if $M \to^*_{\mathsf{N}} \mathsf{S}\ N$, $M' \to^*_{\mathsf{N}} \mathsf{S}\ N'$ and N conv $N' : A \mid_{v:\mathsf{N}}$.

The rest of the definition involves universes, so let us interrupt the definition to look at an example: A type $\mid_a$, for $a = \Pi\ \mathsf{N}\ (0 \mapsto \mathsf{N},\ \mathsf{S}\ 0 \mapsto \mathsf{N})$, means $A \to^*_{\mathsf{type}} \Pi(x : B)F$ with B type $\mid_{\mathsf{N}}$, that is, $B \to^*_{\mathsf{type}} \mathsf{N}$, and

– if $M \to^*_{\mathsf{N}} 0$, then $F(M)$type $\mid_{\mathsf{N}}$, that is, $F(M) \to^*_{\mathsf{type}} \mathsf{N}$, and
– if $M \to^*_{\mathsf{N}} \mathsf{S}\ 0$, then $F(M) \mid_{\mathsf{N}}$, that is, $F(M) \to^*_{\mathsf{type}} \mathsf{N}$.

We now continue the definition to incorporate universes:

4. *Case* U_j. We define:

– A type $\mid_{\mathsf{U}_j}$ means $A \to^*_{\mathsf{type}} \mathsf{U}_j$, and
– A conv $A' \mid_{\mathsf{U}_j}$ is always satisfied.
– $M : A \mid_{u:\mathsf{U}_j}$ means M type$|^j_u$, and
– M conv $M' \mid_{u:\mathsf{U}_j}$ means M conv $M'|^j_u$.

Where the relations A type $|^i_a$ and A conv $A'|^i_a$ used above are simultaneously defined by recursion on the complexity of $a : \mathsf{U}_i$ according to the following cases. We have similar tacit assumptions in the definition: A type $|^i_a$ always additionally assumes $a : \mathsf{U}_i$ (as finite elements) and $A : \mathsf{U}_i$. And A conv $A'|^i_a$ assumes A type $|^i_a$, A' type $|^i_a$, and A conv $A' : \mathsf{U}_i$.

5. *Case* $\bot$. A type $|^i_\bot$ and A conv A' $|^i_\bot$ hold by definition.
6. *Case* $\Pi\ b\ f$. We define:
   - A type $|^i_{\Pi\, b\, f}$ means $A \to^*_{\mathsf{U}_i} \Pi(x : B)F$ for some B and F with B type $|^i_b$ and $x : B \vdash F : \mathsf{U}_i$ and
     (a) $N : B\ |_{v:b}$ implies $F(N)$ type $|^i_{f(v)}$, and
     (b) N conv $N' : B\ |_{v:b}$ implies $F(N)$ conv $F(N')\ |^i_{f(v)}$.
   - Given A' with $A' \to^*_{\mathsf{U}_i} \Pi(x : B')F'$ and A as above, then A conv $A'\ |^i_{\Pi\, b\, f}$ means that B conv $B'|^i_b$ and
$$N : B\ |_{v:b} \quad \text{implies} \quad F(N) \text{ conv } F'(N)\ |^i_{f(v)}\ .$$
7. *Case* N. We define:
   - A type $|^i_{\mathsf{N}}$ means that $A \to^*_{\mathsf{U}_i} \mathsf{N}$.
   - A conv $A'\ |^i_{\mathsf{N}}$ is always satisfied.
8. *Case* U_j with $j < i$. We define:
   - A type $|^i_{\mathsf{U}_j}$ means $A \to^*_{\mathsf{U}_i} \mathsf{U}_j$, and
   - A conv $A'\ |^i_{\mathsf{U}_j}$ is always satisfied.

This concludes the definition of the predicates.

Lemma 8 *Each relation A conv $A'\ |_a$ is an equivalence relation on the set of terms A such that $A\ |_a$. Furthermore, if A conv $A'\ |_a$ then we have $M : A\ |_{u:a}$ iff $M : A'\ |_{u:a}$ and M conv $M' : A\ |_{u:a}$ iff M conv $M' : A'\ |_{u:a}$ for any u in El a.*

Proof This is clear if $a = \bot$ or $a = \mathsf{N}$. If $a = \Pi\ b\ f$, let us prove for instance that the relation A conv $A'\ |_a$ is symmetric. We assume A type $|_a$ and A' type $|_a$ and A conv $A'\ |_a$, and we prove A' conv $A\ |_a$.

We have $A \to^*_{\mathsf{type}} \Pi(x : B)F$ and $A' \to^*_{\mathsf{type}} \Pi(x : B')F'$ and B conv $B'\ |_b$. By induction, we have B' conv $B\ |_b$. Also, we have $N : B'\ |_{v:b}$ iff $N : B\ |_{v:b}$ and this implies $F(x/N)$ conv $F'(x/N)\ |_{f(v)}$ and so $F(x/N')$ conv $F(x/N)\ |_{f(v)}$ by induction. So we get A' conv $A\ |_a$ as required. □

Lemma 9 1. *If $J\ |_a$ and $a' \leqslant a$ in* Type*, then $J\ |_{a'}$ where J is A* type *or A* conv *B.*
2. *If $J|^i_a$ and $a' \leqslant a$ in El U_i, then $J\ |^i_{a'}$ where J is A* type *or A* conv *B.*
3. *If A* type *$|_a$ and $a' \leqslant a$ in* Type *and $J\ |_{u':a'}$, then $J\ |_{u':a}$ where J is $M : A$ or M* conv *$M' : A$.*
4. *Finally, if A* type *$|_a$ and $a' \leqslant a$ in* Type *and $J\ |_{u:a}$ and $u' : a'$ and $u' \leqslant u$, then $J\ |_{u':a'}$ where J is $M : A$ or M* conv *$M' : A$.*

Proof We prove simultaneously the assertions by induction on the complexity of a in Type. We explain two representative cases.

In case $a = \Pi\ b\ f$ and $a' = \Pi\ b'\ f' \leqslant a$ and A type $|_a$, we assume A type $|_a$ and we show A type $|_{a'}$. We first have $A \to^*_{\mathsf{type}} \Pi(x : B)F$ and B type $|_b$ and $N : B\ |_{v:b}$ implies $F(N)$ type $|_{f(v)}$ and N conv $N' : B\ |_{v:b}$ implies $F(N)$ conv $F(N')\ |_{f(v)}$. By induction we have B type $|_{b'}$. Also if $N : B\ |_{v:b'}$ then $N : B\ |_{v:b}$ by induction and so $F(N)$ type $|_{f(v)}$ and so $F(N)$ type $|_{f'(v)}$ by induction. Similarly, N conv $N' B\ |_{v:b'}$ implies $F(N)$ conv $F(N')\ |_{f(v)}$ and so $F(N)$ conv $F(N')\ |_{f'(v)}$ by induction.

If $a = \Pi\ b\ f$ and $a' = \Pi\ b'\ f' \leqslant a$ and A type $|_a$ and $M : A\ |_{u:a'}$, then we claim that $M : A\ |_{u:a}$. We know $A \to^*_{\mathsf{type}} \Pi(x : B)F$ with B type $|_b$. We have to show that $N : B\ |_{v:b}$ implies $M\ N : F(N)\ |_{u(v):f(v)}$. By induction, we know that if $v' \leqslant v$ and v' in $El\ b'$ then $N : B\ |_{v':b'}$. Since $M : A\ |_{u:a'}$ we have $M\ N : F(N)\ |_{u(v'):f'(v')}$. Since $u : a'$ we have $v' \leqslant v$ in $El\ b'$ such that $u(v) = u(v')$ by Lemma 5. For this v' we have $M\ N : F(N)\ |_{u(v):f'(v')}$ and then $M\ N : F(N)\ |_{u(v):f(v)}$ by induction, since $F(N)$ type $|_{f(v)}$ and $f'(v') \leqslant f(v)$. We prove similarly that N conv $P : B\ |_{v:b}$ implies $M\ N$ conv $M\ P : F(N)\ |_{u(v):f(v)}$. □

We use this result to extend the relation A type $|_a$ for a arbitrary (possibly infinite) in Type.

Definition 1 *A type $|_a$ means A type $|_{a_0}$ for all finite $a_0 \leqslant a$ in* Type. *If J is $M : A$ or M conv $M' : A$ then the relations $J\ |_{u:a}$ for u arbitrary in $El\ a$ is defined as follows: for all $u_0 \leqslant u$ finite in $El\ a$ there exists $a_0 \leqslant a$ finite in* Type *such that $J\ |_{u_0:a_0}$.*

Note that if we have $J\ |_{u_0:a_0}$ then we also have $J\ |_{u_0:a_1}$ for any finite a_1 such that $a_0 \leqslant a_1 \leqslant a$ in Type by Lemma 9.

Proposition 2 *We have A type $|_{\Pi\ b\ f}$ if, and only if, $A \to^*_{\mathsf{type}} \Pi(x : B)F$ for B and F with B type $|_b$ and*

1. *$N : B\ |_{v:b}$ implies $F(N)$ type $|_{f(v)}$, and*
2. *N conv $N' : B\ |_{v:b}$ implies $F(N)$ conv $F(N')\ |_{f(v)}$.*

Proof We assume A type $|_{\Pi\ b\ f}$. This means A type $|_{\Pi\ b_0\ f_0}$ for all finite $\Pi\ b_0\ f_0 \leqslant \Pi\ b\ f$ in Type, and, in particular, we have A type $|_{\Pi\ b_0\ f_0}$ for $b_0 = \bot$ and $f_0 = \bot$. This implies $A \to^*_{\mathsf{type}} \Pi(x : B)F$ for some B type and F type $(x : B)$.

If $b_0 \leqslant b$ in Type is finite we have A type $|_{\Pi\ b_0\ \bot}$ and so B type $|_{b_0}$. So we have B type $|_b$.

For $N : B\ |_{v:b}$ we show $F(N)$ type $|_{f(v)}$ by showing that $F(N)$ type $|_{w_0}$ for any $w_0 \leqslant f(v)$ finite in Type. By the remark on continuity of evaluation at the end of Section 2, we can find $b_0 \leqslant b$ finite in Type and $v_0 \leqslant v$ finite in $El\ b_0$ and $f_0 \leqslant f$ finite in $El\ b_0 \to$ Type such that $w_0 \leqslant f_0(v_0)$ in Type. We then have A type $|_{\Pi\ b_0\ f_0}$ and $N : B\ |_{v_0:b_1}$ for some $b_1 \leqslant b$ finite in Type. We can assume $b_0 \leqslant b_1$, maybe changing b_1 to $b_0 \vee b_1$, and using Lemma 9.3. By Lemma 9.4, we also have $N : B\ |_{v_0:b_0}$ and hence $F(N)\ |_{f_0(v_0)}$ as needed to be shown.

The last assertion about conversion has a similar proof.

Conversely assume $A \to^*_{\mathsf{type}} \Pi(x : B)F$ with B type $|_b$ and $N : B\ |_{v:b}$ implies $F(N)\ |_{f(v)}$ and N conv $N' : B\ |_{v:b}$ implies $F(N)$ conv $F(N')\ |_{f(v)}$. We show that A type $|_{\Pi\ b_0\ f_0}$ for all finite $\Pi\ b_0\ f_0 \leqslant \Pi\ b\ f$ in Type. We have B type $|_{b_0}$ by definition, and if $N : B\ |_{v:b_0}$ with v in $El\ b_0$ finite, then $N : B\ |_{v:b}$ and so $F(N)$ type $|_{f(v)}$ and hence $F(N)$ type $|_{f_0(v)}$. We prove similarly that N conv $N' : B\ |_{v:b_0}$ with v finite implies $F(N)$ conv $F(N')\ |_{f_0(v)}$. □

Proposition 3 *Given A type $|_{\Pi\ b\ f}$ and $A \to^*_{\mathsf{type}} \Pi(x : B)F$, we have $M : A\ |_{w:\Pi\ b\ f}$ if, and only if, $N : B\ |_{v:b}$ implies $M\ N : F(N)\ |_{w(v):f(v)}$ and N conv $N' : B\ |_{v:b}$ implies $M\ N$ conv $M\ N' : F(N)\ |_{w(v):f(v)}$.*

Proof Similar to the proof of Proposition 2. □

The two last propositions hold by definition if $\Pi\ b\ f$ is a *finite* element of Type. Note that we could not have used these propositions directly on general, maybe infinite, element as a definition of A type $|_{\Pi\ b\ f}$ since it might be that $f(v)$ is as complex as $\Pi\ b\ f$. The method we have used instead was thus first to define the relation A type $|_{\Pi\ b\ f}$ for $\Pi\ b\ f$ *finite*, and then extend this relation by "continuity" on general elements. This is similar to the use of "inclusive predicates" [12, 15], fundamental in denotational semantics.

Lemma 10 1. *If $A \to_{\mathsf{type}} A'$ and A' type $|_a$, then A type $|_a$ and A conv $A'\ |_a$.*
2. *If A type $|_a$ and $M \to_A M'$ and $M' : A\ |_{u:a}$, then $M : A\ |_{u:a}$ and M conv $M' : A\ |_{u:a}$.*

Proof Both properties are shown by induction on a. The most interesting case is for the second assertion when $a = \Pi\ b\ f$. We then have $A \to^*_{\mathsf{type}} \Pi(x : B)F$ with B type $|_b$. If $N : B\ |_{v:b}$, we have $M\ N \to_{F(N)} M'\ N$ and $M'\ N : F(N)\ |_{u(v):f(v)}$. By induction, we have $M\ N : F(N)\ |_{u(v):f(v)}$ and $M\ N$ conv $M'\ N : F(N)\ |_{u(v):f(v)}$. Similarly, if N conv $N'\ |_{v:b}$, we get $M\ N'$ conv $M'\ N' : F(N)\ |_{u(v):f(v)}$. Since $M' : A\ |_{u:a}$ we have $M'\ N$ conv $M'\ N' : F(N)\ |_{u(v):f(v)}$ and we get by transitivity and symmetry $M\ N$ conv $M\ N' : F(N)\ |_{u(v):f(v)}$. □

We write $\sigma : \Gamma\ |_\rho$ to mean that we have $A\sigma$ type $|_{[\![A]\!]\rho}$ and $\sigma(x) : A\sigma\ |_{\rho(x):[\![A]\!]\rho}$ for all $x : A$ in Γ. Note that, in particular, this implies that ρ fits Γ.

Similarly, we write σ conv $\sigma' : \Gamma\ |_\rho$ to mean that we have $A\sigma$ conv $A\sigma'\ |_{[\![A]\!]\rho}$ and $\sigma(x)$ conv $\sigma'(x) : A\sigma\ |_{\rho(x):[\![A]\!]\rho}$ for all $x : A$ in Γ.

Theorem 2 *The following properties hold, given $\sigma : \Gamma\ |_\rho$ and σ conv $\sigma' : \Gamma\ |_\rho$.*

1. *If $\Gamma \vdash A$ type, then $A\sigma$ type $|_{[\![A]\!]\rho}$.*
2. *If $\Gamma \vdash M : A$, then $M\sigma : A\sigma\ |_{[\![M]\!]\rho:[\![A]\!]\rho}$.*
3. *If $\Gamma \vdash A$ conv A', then $A\sigma$ conv $A'\sigma\ |_{[\![A]\!]\rho}$.*

4. *If* $\Gamma \vdash M$ conv $M' : A$, *then* $M\sigma$ conv $M'\sigma : A\sigma \mid_{[\![M]\!]\rho:[\![A]\!]\rho}$.
5. *If* $\Gamma \vdash A$ type, *then* $A\sigma$ conv $A\sigma' \mid_{[\![A]\!]\rho}$.
6. *If* $\Gamma \vdash M : A$, *then* $M\sigma$ conv $M\sigma' : A\sigma \mid_{[\![A]\!]\rho}$.

Proof This follows from Propositions 2 and 3 and Lemma 10, and the fact that weak-head reduction is stable under substitution. □

Corollary 5 *If* 0 conv $M : \mathsf{N}$, *then* $M \to_{\mathsf{N}}^* 0$. *If* $\mathsf{S}\, M_0$ conv $M : \mathsf{N}$, *then* $M \to_{\mathsf{N}}^* \mathsf{S}\, M_1$ *with* M_0 conv $M_1 : \mathsf{N}$. *If* A_0 conv $\Pi(x : B_1)F_1$, *then* $A_0 \to_{\mathsf{type}}^* \Pi(x : B_0)F_0$ *with* B_0 conv B_1 *and* $N : B_0$ *implies* $F_0(N)$ conv $F_1(N)$.

Proof For the first statement, we have $[\![M]\!] = [\![0]\!] = 0$. Using the previous theorem, we get $M : \mathsf{N} \mid_{[\![M]\!]:\mathsf{N}}$ that is $M : \mathsf{N} \mid_{0:\mathsf{N}}$, which means precisely $M \to_{\mathsf{N}}^* 0$. The proof of the second statement is similar.

If A_0 conv $\Pi(x : B_1)F_1$, then $[\![A_0]\!] = [\![\Pi(x : B_1)F_1]\!] = \Pi\, [\![B_1]\!]\, [\![\lambda(x : B_1)F_1]\!]$ and we have A_0 conv $\Pi(x : B_1)F_1 \mid_{[\![A_0]\!]}$ by Theorem 2. It follows that $A_0 \to_{\mathsf{type}}^* \Pi(x : B_0)F_0$ and B_0 conv $B_1 \mid_{[\![B_0]\!]}$ and $N : B_0 \mid_{v:[\![B_1]\!]}$ implies $F_0(N)$ conv $F_1(N) \mid_{[\![F_0]\!](x=v)}$. In particular, we have B_0 conv B_1 and, for $v = \bot$, we have $F_0(N)$ conv $F_1(N)$ if $N : B_0$. □

Note that we cannot conclude that dependent product is one-to-one for conversion yet, since in the last case we get only that $N : B_0$ implies $F_0(N)$ conv $F_1(N)$, for $N : B_0$ *closed*, which is not enough to conclude F_0 conv F_1 $(x : B_0)$. A simple modification of our argument will apply however, as we shall see in the next section.

6 Connecting Syntax and Semantics, Second Version

We fix a context $\Delta = x_1 : T_1, x_2 : T_2(x_1), \ldots, x_n : T_n(x_1, \ldots, x_{n-1})$. Working in this context Δ corresponds to extend the type system with constants $c_1 : T_1, c_2 : T_2(c_1), \ldots, c_n : T_n(c_1, \ldots, c_{n-1})$. We define the interpretation of these constants by taking $[\![c_i]\!] = \bot$.

We then have $c_1 : T_1 \mid_{[\![c_1]\!]:[\![T_1]\!]}$, $c_2 : T_2(c_1) \mid_{[\![c_2]\!]:[\![T_2(c_1)]\!]}$, ... All the reasoning of the previous section applies with this addition of constants c_i. Moving between constants and variables, we deduce the following proposition, which does not mention constants:

Proposition 4 *If* $\Delta \vdash A_0$ conv $\Pi(x : B_1)F_1$, *then* $A_0 \to_{\mathsf{type}}^* \Pi(x : B_0)F_0$ *with* $\Delta \vdash B_0$ conv B_1 *and* $\Delta \vdash N : B_0$ *implies* $\Delta \vdash F_0(N/x)$ conv $F_1(N/x)$.

Note that for this proposition, the context Δ is completely *arbitrary*. We can thus deduce the following fact:

Corollary 6 *If* $\Delta \vdash A_0 \ \mathsf{conv}\ \Pi(x : B_1)F_1$, *then* $A_0 \to^*_{\mathsf{type}} \Pi(x : B_0)F_0$ *such that* $\Delta \vdash B_0 \ \mathsf{conv}\ B_1$ *and* $\Delta, x : B_0 \vdash F_0 \ \mathsf{conv}\ F_1$.

Proof Since all judgments stay valid by extension of the context, we not only have $\Delta \vdash A_0 \ \mathsf{conv}\ \Pi(x : B_1)F_1$. but also $\Delta, x : B_0 \vdash A_0 \ \mathsf{conv}\ \Pi(x : B_1)F_1$. We can then apply the previous proposition, using $\Delta, x : B_0$ instead of Δ and taking x for u. □

As in [2], an important application of the injectivity of dependent product for conversion is *subject-reduction*, i.e. the following result.

Corollary 7 *If* $A \ \mathsf{type}$ *and* $A \to A'$ *then* $A' \ \mathsf{type}$ *and* $A \ \mathsf{conv}\ A'$. *If* $M : A$ *and* $M \to M'$ *then* $M' : A$ *and* $M \ \mathsf{conv}\ M' : A$.

7 Connecting Syntax and Semantics, Third Version

We refine the domain as follows

$$D \ = \ [D \to D] + \Pi\ D\ [D \to D] + \mathsf{U}_i + \mathsf{N} + 0 + \mathsf{S}\ D + \mathsf{T}$$

and we add the following typing rules:

$$\frac{}{\mathsf{T}\ \mathsf{type}} \qquad \frac{}{\mathsf{T} : \mathsf{U}_i} \qquad \frac{}{\mathsf{T} : \mathsf{N}} \qquad \frac{}{\mathsf{T} : \mathsf{T}}$$

We extend the application operation $u(v)$ by taking $\mathsf{T}(v)$ to be T for any value T. An operational intuition about T is that it represents the semantics of an "exception". We also extend the definition of rec by $\mathsf{rec}(d_0, d_1)\ \mathsf{T} = \mathsf{T}$. We finally refine the definition of the projection function by adding the clauses

$$\begin{array}{lcl} p\ \mathsf{N}\ \mathsf{T} & = & \mathsf{T} \\ p\ \mathsf{U}_j\ \mathsf{T} & = & \mathsf{T} \\ p\ \mathsf{T}\ \mathsf{T} & = & \mathsf{T} \\ p_{\mathsf{type}}\ \mathsf{T} & = & \mathsf{T} \end{array}$$

We now introduce the special class of "neutral" terms

$$k \ ::= \ c_i \mid k\ N \mid \mathsf{rec}(\lambda x\ A, M, M)\ k$$

and the predicate $G(k)$ of "typable" neutral terms, which is defined by the following clauses, where we define at the same time the type function $\tau(k)$:

1. Any constant c_i is typable, $G(c_i)$, and $\tau(c_i) = T_i$ is the given type of c_i.
2. If $G(k)$ and $\tau(k) \to^*_{\mathsf{type}} \Pi(x : B)F$ and $N : B$, then $G(k\ N)$ and $\tau(k\ N) = F(N)$
3. If $G(k)$ and $\tau(k) \to^*_{\mathsf{type}} \mathsf{N}$ and $T\ \mathsf{type}$ $(x : \mathsf{N})$ and $M_0 : T(0)$ and $M_1 : \Pi(x : \mathsf{N})(T \to T(\mathsf{S}\ x))$, then $G(\mathsf{rec}(\lambda x\ T, M_0, M_1)\ k)$ and $\tau(\mathsf{rec}(\lambda x\ T, M_0, M_1)\ k) = T(k)$

We define next an equivalence relation $Q(k, k')$ on elements satisfying G by the clauses:

1. $Q(c_i, c_i)$
2. $Q(k\ N, k'\ N')$ if $Q(k, k')$ and $\tau(k) \to^*_{\mathsf{type}} \Pi(x : B)F$ and $N : B$ and $\tau(k') \to^*_{\mathsf{type}} \Pi(x : B')F'$ and $N' : B'$ and B conv B' and F conv F' $(x : B)$
3. $Q(\mathsf{rec}(\lambda x\ T, M_0, M_1)\ k, \mathsf{rec}(\lambda x\ T', M_0', M_1')\ k')$ if $Q(k, k')$ and $\tau(k) \to^*_{\mathsf{type}} \mathsf{N}$ and $\tau(k') \to^*_{\mathsf{type}} \mathsf{N}$ and T conv T' $(x : \mathsf{N})$ and M_0 conv M_0' : $T(\mathsf{0})$ and M_1 conv $M_1' : \Pi(x : \mathsf{N})(T \to T(\mathsf{S}\ x))$.

We refine then the definitions of $J\ |_a$ and $J\ |_{u:a}$ by the clauses:

1. A type $|_{\mathsf{T}}$ means that $A \to^*_{\mathsf{type}} k$ for some k
2. A conv $A'\ |_{\mathsf{T}}$ means that A conv A'
3. $M : A\ |_{\mathsf{T}:a}$, where a is T or U_i or N, means $M \to^*_A k$ for some k
4. M conv $M' : A\ |_{\mathsf{T}:a}$, where a is T or U_i or N, means M conv $M' : A$

Lemma 11 *If $G(k)$ and $\tau(k)$ type $|_a$, then $k : \tau(k)\ |_{p\ a\ \mathsf{T}:a}$. If $Q(k, k')$ and $\tau(k)$ type $|_a$, then k conv $k' : \tau(k)\ |_{p\ a\ \mathsf{T}:a}$.*

Proof By induction on a type. Let us for instance prove the first assertion in the case where $a = \Pi\ b\ f$. We have $\tau(k) \to^*_{\mathsf{type}} \Pi(x : B)F$ with B type $|_b$ and $N : B\ |_{v:b}$ implies $F(N)$ type $|_{f(v)}$ and N conv $N' : B\ |_{v:b}$ implies $F(N)$ conv $F(N')\ |_{f(v)}$. It follows that $N : B\ |_{v:b}$ implies $G(k\ N)$ and $\tau(k\ N) = F(N)$, so that $k\ N : F(N)\ |_{p\ (f(v))\ \mathsf{T}:f(v)}$ by induction. Similarly we show that N conv $N'\ |_{v:b}$ implies $Q(k\ N, k\ N')$ and so $k\ N$ conv $k\ N' : F(N)\ |_{p\ (f(v))\ \mathsf{T}:f(v)}$ by induction. □

We explain now the semantics of the constants $c_1 : T_1,\ c_2 : T_2(c_1), \ldots$. We take $[\![c_1]\!] = p\ [\![T_1]\!]\ \mathsf{T}$ and then $[\![c_2]\!] = p\ [\![T_2(c_1)]\!]\ \mathsf{T}$ and so on. This is justified since T_1 does not refer to any constant, and T_2 refers at most to the constant c_1, and so on. It follows from the last lemma that we have T_i type $|_{[\![T_i]\!]}$ and $c_i : T_i\ |_{[\![c_i]\!]:[\![T_i]\!]}$.

Theorem 2 holds then with this semantics, since it holds for the constants c_i.

We then have the following application, using as in the previous section the fact that the context Δ is arbitrary.

Theorem 3 *If $\Delta \vdash M$ conv $N : A$, then $M : A$ and $N : A$ have the same Böhm tree.*

Proof Corollary 7 implies that a term is convertible to (and hence as the same semantics as) its weak head normal form. Theorem 2 shows then that, given any two convertible terms, if one has a weak head normal form, so does the other term and these weak head normal form have the same shape. □

8 Conclusion

We have shown that constructors are one-to-one for dependent type theory with conversion as judgment *and* η-conversion in a weak metatheory, while all existing proofs [2] use strong logical principles. Our argument applies as well to *partial* type theory, where we may have non terminating computations. An example is given in the reference [11]: one introduces a new base type Ω, which is like the type of natural numbers N with $\mathsf{0}$ deleted, and an element $\omega : \Omega$ such that $\omega \ \mathsf{conv}\ \mathsf{S}\ \omega : \Omega$. The type Ω will be represented by a new finite element of the domain, while the element ω will be the least upper bound of the sequence $\bot$, $\mathsf{S}\ \bot$, $\mathsf{S}\ (\mathsf{S}\ \bot), \ldots$

Using strong logical principles, it should be possible to define a semantical notion of totality on elements of the domain, and prove that a total element corresponds to a finite Böhm tree. If we are only interested in the evaluation of closed expressions, the techniques we have presented are enough to show canonicity of type theory extended with bar recursion, as in [8], but with η-conversion in the type system.

On the other hand it is not clear how to extend the present method to a type system with a type of all types. Do we still have adequacy in this case?

Appendix: Typing and Conversion Rules of Type Theory

Rules for Contexts

$$\frac{}{() \vdash} \qquad \frac{\Gamma \vdash A\ \mathsf{type}}{\Gamma, x : A \vdash} \qquad \frac{\Gamma \vdash}{\Gamma \vdash x : A}\ (x : A \text{ in } \Gamma)$$

Like in [9], we don't write explicitly the context in the next rules.

Typing Rules

$$\frac{M : A \qquad A\ \mathsf{conv}\ B}{M : B} \qquad \frac{A\ \mathsf{type} \qquad B\ \mathsf{type}\ (x : A)}{\Pi(x : A)B\ \mathsf{type}}$$

$$\frac{A\ \mathsf{type} \qquad B\ \mathsf{type}\ (x : A) \qquad N : B\ (x : A)}{\lambda(x : A)N : \Pi(x : A)B}$$

$$\frac{A\ \mathsf{type} \qquad B\ \mathsf{type}\ (x : A) \qquad N : \Pi(x : A)B \qquad M : A}{N\ M : B(x/M)}$$

Conversion Rules

$$\frac{M \text{ conv } N : A \qquad A \text{ conv } B}{M \text{ conv } N : B} \qquad \frac{A \text{ type}}{A \text{ conv } A} \qquad \frac{A \text{ conv } C \qquad B \text{ conv } C}{A \text{ conv } B}$$

$$\frac{M : A}{M \text{ conv } M : A} \qquad \frac{M \text{ conv } P : A \qquad N \text{ conv } P : A}{M \text{ conv } N : A}$$

$$\frac{A_0 \text{ conv } A_1 \qquad B_0 \text{ conv } B_1 \ (x : A_0)}{\Pi(x : A_0)B_0 \text{ conv } \Pi(x : A_1)B_1}$$

$$\frac{A \text{ type} \qquad B \text{ type } (x : A) \qquad N \text{ conv } N' : \Pi(x : A)B \qquad M : A}{N\ M \text{ conv } N'\ M : B(x/M)}$$

$$\frac{A \text{ type} \qquad B \text{ type } (x : A) \qquad N : \Pi(x : A)B \qquad M \text{ conv } M' : A}{N\ M \text{ conv } N\ M' : B(x/M)}$$

$$\frac{A \text{ type} \qquad B \text{ type } (x : A) \qquad N : B\ (x : A) \qquad M : A}{(\lambda(x : A)N)\ M \text{ conv } N(x/M) : B(x/M)}$$

$$\frac{A \text{ type} \quad B \text{ type } (x : A) \quad N : \Pi(x : A)B \quad N' : \Pi(x : A)B \quad Nx \text{ conv } N'x : B\ (x : A)}{N \text{ conv } N' : \Pi(x : A)B}$$

Rules for Natural Numbers

$$\frac{}{\mathsf{N} : \mathsf{U}_0} \qquad \frac{}{0 : \mathsf{N}} \qquad \frac{M : \mathsf{N}}{\mathsf{S}\ M : \mathsf{N}}$$

$$\frac{T \text{ type } (x : \mathsf{N}) \qquad M_0 : T(0) \qquad M_1 : \Pi(x : \mathsf{N})\ (T \to T(\mathsf{S}\ x))}{\mathsf{rec}(\lambda x\ T, M_0, M_1) : \Pi(x : \mathsf{N})T}$$

$$\mathsf{rec}(\lambda x\ T, M_0, M_1)\ 0 \text{conv } M_0$$
$$\mathsf{rec}(\lambda x\ T, M_0, M_1)\ (\mathsf{S}\ n) \text{conv } M_1\ n\ (\mathsf{rec}(\lambda x\ T, M_0, M_1)\ n)$$

Rules for Universes

$$\frac{A : \mathsf{U}_i \qquad B : \mathsf{U}_i\ (x : A)}{\Pi(x : A)B : \mathsf{U}_i} \qquad \frac{A : \mathsf{U}_i}{A : \mathsf{U}_j} i < j \qquad \frac{}{\mathsf{U}_i : \mathsf{U}_j} i < j \qquad \frac{A : \mathsf{U}_i}{A \text{ type}}$$

$$\frac{M \text{ conv } N : \mathsf{U}_i}{M \text{ conv } N : \mathsf{U}_j} i < j \qquad \frac{M \text{ conv } N : \mathsf{U}_i}{M \text{ conv } N}$$

$$\frac{A_0 \text{ conv } A_1 : \mathsf{U}_i \qquad B_0 \text{ conv } B_1 : \mathsf{U}_i\ (x : A_0)}{\Pi(x : A_0)B_0 \text{ conv } \Pi(x : A_1)B_1 : \mathsf{U}_i}$$

References

1. Abel, A., Coquand, Th., Dybjer, P.: Normalization by evaluation for martin-löf type theory with typed equality judgements. In: Proceedings of LICS '07 (2007)
2. Abel, A., Scherer, G.: On irrelevance and algorithmic equality in predicative type theory. Logical Methods Comput. Sci. **8**(1), 1–36 (2012)
3. Aczel, P.: On relating type theories and set theories. Types for proofs and programs. LNCS **1657**, 1–18 (1998)
4. Adams, R.: Pure type systems with judgemental equality. J. Funct. Program. **16**(2), 219–246 (2006)
5. Cardelli, L.: A polymorphic λ-calulus with type:type. SRC Research Report 10 (1986)
6. Coquand, Th.: An algorithm for testing conversion in type theory. In: Logical Frameworks, pp. 255–279 (1991)
7. Coquand, Th., Gunter, C., Winskel, G.: Domain theoretic models of polymorphisms. Inf. Comput. **81**, 123–167 (1989)
8. Coquand, Th., Spiwack, A.: A proof of strong normalisation using domain theory. In: Proceeding of LICS '06, pp. 307–316 (2006)
9. Martin-Löf, P.: Constructive mathematics and computer programming. Stud. Logic the Found. Math. **104**, 153–175 (1982)
10. Martin-Löf, P.: Lecture Notes on the Domain Interpretation of Type Theory. Workshop on Semantics of Programming Languages. Chalmers University, Göteborg (1983)
11. Martin-Löf, P.: Unifying Scott's Theory of Domains for Denotational Semantics and Intuitionistic Type Theory (Abstract). Atti del Congresso Logica e Filosofia della Scienza San Gimignano, December 1983. Vol. I
12. Milne, R.: The formal semantics of computer languages and their implemnetation. PhD thesis, Oxford University Computing Laboratory (1974)
13. Palmgren, E., Stoltenberg-Hansen, V.: Domain interpretations of martin-löf's partial type theory. Annals Pure Appl. Logic **48**(2), 135–196 (1990)
14. Palmgren, E., Stoltenberg-Hansen, V.: Remarks on martin-löf's partial type theory. BIT **32**, 70–82 (1992)
15. Reynolds, J.: On the relation between direct and continuation semantics. In: Proceedings of the Second Colloquim on Automata, Languages and Programming, pp. 141–156 (1974)
16. Schwichtenberg, H., Wainer, S.S.: Proofs and Computations (Perspectives in Logic). Cambridge University Press, Cambridge (2012)
17. Siles, V., Herbelin, H.: Equality is typable in semi-full pure type systems. In: Proceedings of LICS '10 (2010)
18. Scott, D.: Lectures on a mathematical theory of computation. Oxford University PRG Technical Monograph (1981)
19. Scott, D.: Some ordered sets in computer science. NATO Advanced Study Institutes Series book series (ASIC, vol. 83), pp. 677–718 (1982)