Canonicity and normalization for type theory

Thierry Coquand

ITC talk, December 2021
Hilbert (1925), Gödel (1941)

Tait (1967), Girard (1970), Martin-Löf (1971), Hancock (1973)

Hilbert introduced primitive recursion with higher functions as parameters

\[ f(0) = a \quad f(n + 1) = g(n, f(n)) \]

Hilbert *Über das Unendliche*, 1925
Some history

Example: \( \iota(f, a, 0) = a \) \( \iota(f, a, n + 1) = f(a, \iota(f, a, n)) \)

\( \varphi(0, a, b) = a + b \) \( \varphi(n + 1, a, b) = \iota(\varphi(n), a, b) \)

We can rewrite the computation rules of \( \varphi \) as

\( \varphi(0, a, b) = a + b \)
\( \varphi(n + 1, a, 0) = a \) \( \varphi(n + 1, a, b + 1) = \varphi(n, a, \varphi(n + 1, a, b)) \)

Hilbert asked his student Ackermann to show that one cannot define the function \( \varphi(n, a, b) \) without using higher functions

Early justification of functional programming!
Gödel 1941 *In what sense is intuitionistic logic constructive*

Starts from the “circularity problem” with \( \rightarrow \) and \( \neg \) in intuitionistic logic

In order to solve this problem, he designed a system \( \Sigma \) (what would becomes system \( T \)) based on Hilbert’s functionals

He wants a justification of intuitionistic arithmetic in a “simpler” system, with only *decidable* statements, as equations between functionals

What is essential then is to have a system where \( f = g \) is *decidable*

First technical occurrence of the notion of definitional equality
Some history

This 1941 talk became the interpretation known as “Dialectica” presented in 

Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes 
Dialectica, 12, pp. 280-287.

and a mathematical proof that $f = g$ is decidable appears in Tait 1967

Intensional interpretations of functionals of finite type, part I 
Journal of Symbolic Logic, 32, pp. 198-212.
The key idea in Tait’s argument is presented very clearly in Shoenfield’s book *Mathematical logic*

The presentation is in term of combinators, and not $\lambda$-calculus, and is actually close to the original one of Gödel

One introduces *constants* with defining equation

$$f \ x_1 \ldots \ x_n = t(x_1, \ldots, x_n)$$

or by recursion equations

$$f \ 0 = a \text{ and } f \ (S \ x) = g \ x \ (f \ x)$$
One defines when a closed term is *computable* by induction on its type:

\[ N'(t) \text{ means } t \text{ convertible to a numeral } S^k 0 \]

\[ (A \rightarrow B)'(t) \text{ means that } A'(u) \text{ implies } B'(t u) \]

One proves then that any closed term is computable by induction on the term:

All Lemmas are simple and have direct proof.
Lemma 1: if a term is built from computable constants, it is computable

Lemma 2: if \( t_0 \) and \( t_1 \) are convertible and \( t_0 \) computable then so is \( t_1 \)

Lemma 3: if \( t \ (S^k0) \) computable for all \( k \) then \( t \) is computable

Lemma 4: all constants are computable

Theorem: all closed terms are computable

In particular, a closed term of type \( N \) is convertible to a numeral!
This is *canonicity*

The proof is elegant, and uses at the meta language a logic with implication and universal quantification.

There is another more combinatorial proof with ordinals $< \epsilon_0$ but this argument is technically much more complex (Turing 1940 had a similar argument with $< \omega^3$ for simply typed $\lambda$-calculus).

Also not clear if we can really explain intuitionistic logic in this way.

Gödel was never completely satisfied with this approach (he never published an English translation of his 58 paper, after several attempts).
Tait also proves *normalisation*

He defines $N'(t)$ to mean that $t$ reduces to a numeral

He then proves that if $t$ is computable of a type $A$ then $t$ is *normalisable* by induction on $A$ using the function $0_A$

$$0_N = 0 \text{ and } 0_{A \rightarrow B} x = 0_B$$

One proves at the same time that $0_A$ is computable and that if $t$ is computable at type $A$ then $t$ is normalisable

Tait (and later Girard) considers the reduction where one only reduces “minimal” inner-most redexes
Canonicity and normalization for type theory

Canonicity and normalisation

*Canonicity*: any closed term of type $N$ is convertible to a numeral $S^k 0$

*Normalisation*: we can reduce any well-typed term (of any type) to a term which does not contain any redex

A corollary of normalisation (and Church-Rosser) is that conversion is *decidable*

One then deduces that type-checking for dependent types is decidable

(Completely different motivation than the one of Gödel!)
In 1971, Girard introduces systems $F$ and $F_\omega$.

This contains polymorphic types like $\Pi\alpha \alpha \to \alpha$ with the polymorphic identity function $\lambda\alpha \lambda x : \alpha x$.

What should computable at type $T = \Pi\alpha \alpha \to \alpha$ mean?

It cannot be: for all $A$ we have $t A$ computable at type $A \to A$ since $A$ may contain $T$ itself and the definition would be circular.
In order to solve this problem, Girard introduces the notion of *computability candidate* which abstracts the main properties of being computable

(1) $C_A(t)$ implies that $t$ is normalisable

(2) If $C_A(u)$ and $t$ reduces to $u$ then $C_A(t)$

(3) We have $C_A(0_A)$

Girard also introduces a constant $0_A$ with $0_{A \rightarrow_B t}$ reduces to $0_B$ and $0_{\Pi_\alpha T} A$ reduces to $0_{T(A)}$ (not definable in his system)
Computability candidate

If one extends system $F$ with a base type $N$ and only wants to prove canonicity the notion of computability candidate becomes much simpler:

A computability candidate just becomes an arbitrary predicate on the set of closed terms of type $A$

This notion of abstract computability will play a crucial rôle in what follows

In general, one expects canonicity to be simpler to prove than normalisation
Girard generalized this to $F_\omega$ which corresponds to propositional higher-order logic.

Martin-Löf 1971 notices that one wants more than higher-order logic, one also wants to be able to form the type of all structures (like algebraic structures, groups, vector spaces, ... ) and quantify over them.

He introduces a type system with a *type of all types* and is able to prove normalisation, using as a meta theory with a type of all types.

This is not at all absurd a priori, since Russell’s paradox does not apply directly.

We will explain later how such a proof works (e.g. for canonicity).
Girard 1972 shows (by chance! He was looking at another system: an extension of system $F$ with a type of propositions) that Martin-Löf 1971 contains a non normalisable term!

Martin-Löf presents then a serie of systems 1972, 1975, 1979 with a notion of predicative universes

We have $\Pi_{x:A} B : U$ if both $A$ is of type $U$ and $B(x) : U$

We don’t have $U : U$ anymore

He proves normalisation for 1972 and 1975 systems, and indicates how to prove canonicity for the 1979 system; normalisation does not hold for the 1979 version
In 71 version it is stated explicitly that consistency follows from normalisation.

There is no normal term of type $\Pi_{X:U} X$.

In the 75 version, $\bot$ is introduced as a data type with no constructor.

In all 71-75 what is stated is that all closed terms of types $N \rightarrow N$ represent a computable function.

Canonicity will be enough to prove this.
Canonicity and normalization for type theory

Type Theory

71-72:
untyped conversion, proof of normalisation, only $\beta$-conversion,

75:
conversion as judgement, proof of normalisation, no $\xi$-rule,

79:
conversion as judgement, canonicity, $\beta$ and $\eta$ conversion, normalisation does not hold, type checking not decidable
Dependent type theory is intuitive (cf. Agda) but not so easy to present in details.

Both 71-72 versions are based on an untyped conversion relation and Church-Rosser property.

Don't contain $\eta$-conversion.

It was known from work’s on Automath that we don’t have Church-Rosser at an untyped level with $\eta$-reduction!
If one wants to present e.g. set theoretic models, one needs to have the conversion relation as a judgement.

Cf. Peter Aczel *On relating Type Theories and Set Theories*, 1998

The equivalence between the two versions is not easy at all. *It needs the key result that Π is one-to-one for conversion.*

Then, one shows subject reduction for system with conversion as judgement.
Conversion as judgement

It does not contain $\xi$-rule, not even less $\eta$-conversion!

(Not clear yet if this system can really be used in practice)

*New* idea for proving that $\Pi$ is one-to-one, due to Peter Hancock

Not clear how to solve the issue that some types may be empty (solved by Girard with introduction of 0 terms): if one introduces constants, is this conservative?
Conversion as judgement

Equality reflection rule: implies function extensionality but also non
normalisation in presence of universes

One can extract a *canonicity* result from the intended semantics but is is more
complex: we have a computability *relation* and not a *predicate*

Furthermore it is justified by a complex inductive recursive process (Stuart
Allen LICS 1987)
The situation in the 80s was the following

- two different presentations of dependent type theory, one with untyped reduction and conversion, the other with conversion as judgement (with or without $\eta$-conversion)

- not clear if the two different versions are equivalent

- for the untyped reduction presentation, *without* $\eta$-conversion, one can prove $\Pi$ one-to-one, and hence subject reduction, but it is not clear how to do it if we want $\eta$-conversion

- for the conversion as judgement presentation, not clear how to prove $\Pi$ one-to-one, and subject reduction
Discussion summary

- For the conversion as judgement presentation, not even clear how to prove that $N$ and $U_k$ or $\Pi_{x:A}B$ are not convertible!

- For the conversion as judgement presentation, not even clear how to prove canonicity!
Proof of canonicity

Since canonicity should be simpler than normalisation, a natural attempt is first to understand how to get a better proof of *canonicity*

I will start by explaining what is the problem
We want to define computable at type $T$

For $T = N$ it means convertible to a numeral $S^k 0$

For $T = \Pi_{x:A} B$ it means $t \, u$ computable at type $B(u/x)$ if $u$ computable at type $A$

For $T = \Sigma_{x:A} B$ it means $t.1$ computable at type $A$ and $t.2$ computable at type $B(t.1/x)$

For $T = U_k$ not so clear what to take as a definition
Inductive-recursive definition? This is what is done in the 1972 version

For instance, means that $t$ is convertible to $N$ or to $\Pi_{x:A}B$ or to $\Sigma_{x:A}B$

The problem is that, a priori, it may be that $N$ is convertible to a type of the form $\Pi_{x:A}B$

Also, we don’t know a priori that $\Pi$ is one-to-one for conversion

This works well if we have a Church-Rosser untyped reduction relation
This can be solved by introducing a reduction relation and considering a computability relation instead of a predicate.

The argument gets technically quite complex however.

This is what is done (for normalisation) in

*An algorithm for testing conversion in type theory*, Th. C., 1991

A similar argument has been checked in Agda!

A. Abel, J. Öhman, A. Vezzosi

*Decidability of conversion for type theory in type theory*, 2018
Is there a canonicity proof as clear as Shoenfield’s argument?

Yes! The solution is to replace the notion of computability *predicate* by computability *structure*

We define $A'(t)$ to be a *set* instead of being a *proposition*

All problems disappear like by magic!

This solution is presented in

*Canonicity and normalisation for dependent type theory*, Th.C. 2018
The definition becomes an *interpretation/non standard model* of type theory

Definition of $t'$ by induction on $t$ (informal presentation)

For a type $T$ the interpretation $T'$ is a family of sets over the set of closed term of type $T$ *modulo conversion*

If $t : T$ then $t'$ is an element of $T'(t)$
If $B(x)$ is a family of types over $A$

Then $B'(u, u')(v)$ is a family of sets for $v$ closed term of type $B(u/x)$ (modulo conversion) provided $u$ closed term of type $A$ and $u'$ element of the set $A'(u)$

Note that $B'(u, u')$ depends both of $u$ and $u'$
Canonicity and normalization for type theory

Computability structure

\[ T = N, \quad T'(t) \text{ is } \{k \mid t \text{ conv } S^k 0\} \]

\[ T = \Pi_{x:A} B, \quad T'(t) \text{ is } \Pi_{u \in \text{Elem}(A)} \Pi_{u' \in A'(u)} B'(u, u')(t \ u) \]

\[ T = \Sigma_{x:A} B, \quad T'(t) \text{ is } \Sigma_{u' \in A'(t.1)} B'(t.1, u')(t.2) \]
The key clause is

$$T = U_k, T'(X) \text{ is } \text{Elem}(X) \rightarrow U_k$$

\text{Elem}(A) \text{ set of closed terms of type } A \text{ (modulo conversion)}

We assume to have a sequence of universes $U_k$ in our meta theory

One recognises the set of computability candidate!

A similar definition was in Martin-Löf 73 (work with Peter Hancock) but he also introduced a reduction relation and did not cover $\xi$-rule
Canonicity and normalization for type theory

Computability *structure*

\[ T = N, \ T'(t) = \{k \mid t \text{ conv } S^k 0\} \]

Even for \( T = N \) we have a set which a priori may not be a *proposition*

Maybe we have 0 and \( S \ 0 \) convertible a priori!
Computability structure

(t u)' is \( t' \ u \ u' \)

\((\lambda x: A t)'\) is \( \lambda u \in \text{Elem}(A) \lambda u' \in A'(u) t'(u, u') \)

(t.i)' is \( t'.i \)

(u, v)' is \( (u', v') \)
Computability *structure*

Note that there is a *uniform* treatment of terms and types.

We define $A'$ for $A$ type: family of sets $A'(u)$ for $u$ closed term of type $A$.

*and* we define $u'$ which is an element of the set $A'(u)$.
We can prove that if \( t : T \) then \( t' \) is an element of \( T'(t) \).

If \( t_0 \) conv \( t_1 : T \) then \( t'_0 = t'_1 \) in \( T'(t_0) = T'(t_1) \).

In particular for \( t : N \) we have \( t' \) which is a numeral \( k \) such that and \( t \) conv \( S^k 0' \).
“Equational” presentation of type theory

Present models of type theory with sorts, operations and equations

1982 John Cartmell

1988 Thomas Ehrhard

1992 Eike Ritter

1996 Peter Dybjer, cwf

2010 Vladimir Voevodsky, C-system

The term model is the initial model

Exactly like model of an equational theory!
The proof of canonicity can be seen as an instance of “sconing” or “glueing” or “Freyd cover” of the term/initial model of type theory with the set theoretic model.


The analogy between sconing and proving canonicity were clear in the 80s e.g. *A Note on Freyd Cover and Friedman Slash*, A. Scedrov and P.J. Scott, 1982
Starting from an arbitrary model $M$ we build a new model $M^*$

A closed type of $M^*$ is a closed type $A$ of $M$ together with a family of sets over the set of closed terms of type $A$

We always have a projection map $M^* \rightarrow M$

For the initial/term model $M_0$ we have the initial map $M_0 \rightarrow M_0^*$

and this should be a section of the projection map
Computability method and sconing

The metalanguage does not need to be set theory

It can be type theory extended with types of closed terms

Interesting project to make this precise!
In topos theory, one can apply this method for two models $M_1$ and $M_2$ and a left exact functor $F : M_1 \to M_2$

A new object will be $A_1, A_2, f$ with $f : A_2 \to F(A_1)$

The same method works for models of type theory!!

*This is an important result due to Simon Huber 2018*

The only condition is that $F$ is a pseudo-morphism of cwf
Canonicity is the special case where $F(\Gamma)$ is the set of closed instance $1 \to \Gamma$

$M_1$ term model and $M_2$ set model

This explains the analogy between *canonicity* and *parametricity* (Ambrus Kaposi, 2018)

Parametricity is the special case where $F$ is the identity functor $F(\Gamma) = \Gamma$

$M_1$ term model and $M_2$ term model
Canonicity and normalization for type theory

Computability method and parametricity

\[ T = N, \ T'(t) \text{ is defined inductively } T'(0) \text{ and } T'(u) \rightarrow T'(S \ u) \]

\[ T = \Pi x: A B, \ T'(t) \text{ is } \Pi u: A \Pi u': A'(u) B'(u, u')(t \ u) \]

\[ T = \Sigma x: A B, \ T'(t) \text{ is } \Sigma u': A'(t.1) B'(t.1, u')(t.2) \]

\[ T = U_k, \ T'(X) \text{ is } X \rightarrow U_k \]
The method works as well with inductive families!

E.g. we define \((\text{Id } A \, t \, u)\)' as an inductive family with constructor

\((\text{Id } A \, u \, u)'(\text{refl } u)\)

Canonicity works as well
Canonicity and normalization for type theory

Computability method and parametricity

Parametricity works as well if we have $U : U$

One defines $U'(X)$ to be $X \to U$

We can do a canonicity proof taking $U'(X)$ to be $\text{Elem}(X) \to U$

We use $\mathcal{U}$ in $\mathcal{U}$ in the meta theory

This explains what happens for the proof in Martin-Löf 1971
The same method will work for canonicity for dependent type theory extended with *modal* operations

We use as metalanguage a modal dependent type theory!
Not so easy to represent this “proof relevant” argument in Agda
(This will even be more so for the normalisation proof)
On the other hand, the argument is simple at a conceptual level
Important challenge for proof assistants!
In general, the “proof relevant” computability method is such that what is used at the metalevel is “almost the same” as what is going on in the system we analyse.

So the argument is relevant at a technical level, e.g. for ensuring decidability of conversion.

But it cannot be used to argue for consistency.

This was also the conclusion understood by Martin-Löf around 1979: presents meaning explanation instead of normalisation proof (this is discussed in the “Bibliopolis” 1984 book).
In the Princeton notes, Gödel observes that it is not difficult to prove calculability of the functionals in the following sense: a functional \( F \) of type \( \sigma_1, \ldots, \sigma_n \to o \) is said to be calculable if for arbitrary calculable \( t_1, \ldots, t_n \) of types \( \sigma_1, \ldots, \sigma_n \) respectively, \( Ft_1 \ldots t_n \) can be proved to be equal to a numeral. Gödel goes on to say “I don't want to give this proof in more detail because it is of no great value for our purpose for the following reason: if you analyse this proof it turns out that it makes use of logical axioms, also for expressions containing quantifiers and it is exactly these axioms which we want to deduce from the system \( \Sigma \).”
Tait’s (and Girard’s) proof used special terms $0_A$

How to have such terms for dependent type theory?

Martin-Löf suggested the addition of *constants*

Technically one needs to iterate this: adding constants for closed typed will create more closed types

Furthermore, it is not clear if we get a conservative extension when adding constants if one presents the system with conversion as judgement
In 88, I suggested the use of contexts as Kripke worlds in order to deal with this issue.

The motivating observation was that the canonicity proof is constructive and hence makes sense in any Kripke/presheaf model.

I was very pleased by the fact that it gives a notion of “partial terms” which provides exactly what is needed for normalisation.

At first, I saw this only as an alternative of introducing constants, but it actually solves the problem of showing that this addition of (partial) constants is conservative.
Use of contexts as Kripke world

This is presented in

*An algorithm for testing conversion in type theory*, Th. C., 1991

and

Th.C. and J. Gallier *A proof of Strong Normalisation for the Theory of Constructions Using a Kripke-Like Interpretation*, 1990
Use of contexts as Kripke world

This technique applies for dependent type theory

$A'(t)$ is then a dependent presheaf

Not clear yet how to formalise this argument in a proof assistant

If the computability predicate/relaion is not proof relevant there is no such problem

Important challenge for proof assistants!
Also, because of the type restrictions on the rules of conversion and reduction, the method for proving the Church-Rosser property developed in combinatory logic apparently no longer works. Instead, the uniqueness of normal form and the Church-Rosser property are proved, almost without effort, as corollaries to the construction of the term model by a new method, due to Peter Hancock.
Syntactic category

Given any model $M$ we build a “syntactic” category $C$ from it.

An object of $C$ is a telescope $X = A_1.A_2.\ldots.A_n$ of types in $M$.

To any object $X$ of $C$ we can associate a context $\langle X \rangle$ of $M$.

For $A$ in $\text{Type}(X)$ we define the set $\text{Term}(X, A)$ which is the set of syntactical expressions of type $A$ (terms without quotienting by conversion).

A morphism $\sigma : Y \to X$ of $C$ is defined by induction on the length of $X$ and we associate $\langle \sigma \rangle : \langle Y \rangle \to \langle X \rangle$ of $M$.

We can have $\langle \sigma_0 \rangle = \langle \sigma_1 \rangle$ without having $\sigma_0 = \sigma_1$. 
Syntactic category

Any expression \( e \) in \( \text{Term}(X, A) \) defines an element \( \langle e \rangle \) in \( \text{Elem}(\langle X \rangle, A) \) (by quotienting modulo conversion)

Any context \( \Gamma \) of \( M \) defines a presheaf \( |\Gamma| \) of \( \hat{C} \) where \( |\Gamma|(X) \) is the set of substitutions \( \langle X \rangle \rightarrow \Gamma \)
If $A$ type over $\langle X \rangle$ and $a$ in $\text{Elem}(\langle X \rangle, A)$ we define $\text{Term}(X, A)|a$ the set of syntactic expressions $e$ of type $A$ such that $\langle e \rangle = a$ in $\text{Elem}(\langle X \rangle, A)$

We work with presheaves over $\mathcal{C}$
Syntactic category

In this presheaf category

we have a cumulative sequence of types $\text{Type}_n$

for $A : \text{Type}_n$ we have a type $\text{Elem}(A)$ and a type $\text{Term}(A)$

We have a quotient map

$\text{Term}(A) \rightarrow \text{Elem}(A)$

$u \mapsto \langle u \rangle$

This map has for each $\Gamma$ a section (not natural in $\Gamma$)
Canonicity and normalization for type theory

**Syntactic category**

\[
\text{Elem}(A \to U_n) \text{ is canonically isomorphic to } \text{Term}(A) \to \text{Elem}(U_n)
\]

If \( B : \text{Elem}(A) \to \text{Elem}(U_n) \) then

\[
\text{Elem}(\Pi_A B) \text{ is canonically isomorphic to } \Pi_{k: \text{Term}(A)} \text{Elem}(B\langle k \rangle)
\]

If \( F \) is a dependent presheaf over \textbf{Type}, we have a canonical isomorphism

\[
F^\text{Term}(X, A) \simeq F(X.A, Ap)
\]
Non standard interpretation

Any term $t$ is interpreted by a “semantical” element $\bar{t}$

If $T$ is a type $\bar{T}$ is a tuple $T', K, \alpha_T, \beta_T$ where

- $T'$ is a family of sets over $\text{Elem}(T)$
- $K$ is a term in $\text{Term}(U_n)|T$
- $\alpha_T u \bar{u}$ is in $\text{Term}(T)|u$ if $\bar{u}$ is in $T'(u)$
- $\beta_T k$ is in $T'\langle k \rangle$ if $k$ is in $\text{Term}(T)$

If $t : T$ then $\bar{t}$ is an element of $T'(t)$
Non standard interpretation

\( \alpha_T \ u \ \bar{u} \) is in \( \text{Term}(T)|u \) if \( \bar{u} \) is in \( T'(u) \)

Proof relevant way to state: if \( u \) “satisfies” \( T' \) then \( u \) is normalizable

Note that \( \alpha_T \ u \ \bar{u} \) may depend on \( \bar{u} \)

\( \beta_T \ k \) is in \( T'(k) \) if \( k \) is in \( \text{Term}(T) \)

Proof relevant replacement of the use of \( 0_T \) in Tait's and Girard's proof!!
Canonicity and normalization for type theory

Non standard interpretation

$\overline{U_n} = U'_n, U_n, \alpha_{U_n}, \beta_{U_n}$

$U'_n(T)$ is the set of tuples $T', K, \alpha_T, \beta_T$

$\alpha_{U_n} T (T', K, \alpha, \beta) \text{ is } K$

$\beta_{U_n} K \text{ is } (K', K, \alpha, \beta) \text{ where } K'(t) \text{ is }$

$\text{Term}(K)|t \text{ and } \alpha t k = k \text{ and } \beta k = k$
We can define

$$\Pi^I_A B \text{ in } \text{Elem}(U_n)$$

for $A : \text{Elem}(U_n)$ and $B : \text{Term}(A) \rightarrow \text{Elem}(U_n) \simeq \text{Elem}(A \rightarrow U_n)$

$$\Pi^S_A B \text{ in } \text{Term}(U_n)$$

for $A : \text{Term}(U_n)$ and $B : \text{Term}\langle A \rangle \rightarrow \text{Term}(U_n)$

such that $\langle \Pi^S_A B \rangle = \Pi^I_{\langle A \rangle}(\lambda k \langle B \ k \rangle)$
Non standard interpretation

For $T = \Pi_{x:A} B$

Given $A_0, A', \alpha_A, \beta_A$ and $B_0(u, \overline{u}), B'(u, \overline{u}), \alpha_B(u, \overline{u}), \beta_B(u, \overline{u})$

We define $Gk = B_0(\langle k \rangle, \beta_A(k))$ and

$T_0 = \Pi_{A_0}^S G$

$T'(w) = \Pi_{u:\text{Elem}(A)} \Pi_{\overline{u}:A'(u)} B'(u, \overline{u})(w \ u)$

$\alpha_T \ w \ \overline{w} = \lambda^S \ A_0 \ G \ (\lambda_k \alpha_B(\langle k \rangle, \beta_A(k))(w \ \langle k \rangle)(\overline{w} \ \langle k \rangle \ \beta_A(k)))$

$\beta_T \ k = \lambda_{u:\text{Elem}(A)} \lambda_{\overline{u}:A'(u)} \beta_B(u, \overline{u})(k \ \alpha_A \ u \ \overline{u})$
Non standard interpretation

If $t$ closed element of type $T$ then $\alpha_T t \bar{t}$ is a closed normal expression such that $\langle \alpha_T t \bar{t} \rangle = t$ in $\text{Elem}(T)$

In particular if $t_0$ and $t_1$ closed element of type $T$ then $t_0 = t_1$ in $\text{Elem}(T)$ iff $\alpha_T t_0 \bar{t}_0 = \alpha_T t_1 \bar{t}_1$ in $\text{Term}(T)$

So conversion is decidable

We can follow Peter Hancock’s argument and prove that $\Pi$ is one-to-one

We never have to introduce a reduction relation, but we deduce from this subject reduction w.r.t. untyped reduction