An intuitionistic theory of types with assumptions of high-arity variables

A. Bossi[1] and S. Valentini[2]

1. Introduction

Since the 70's Per Martin-Löf has developed, in a number of successive variants, an Intuitionistic Theory of Types [Mar75, Mar82, Mar84, Nor89]. The initial aim was that of providing a formal system for Constructive Mathematics but the relevance of the theory also in Computer Science was soon recognized. In fact, Martin-Löf's type theory can equally well be viewed as a programming language with a very rich type structure, as a specification language and as an integrated system to derive correct programs from their specifications [NoP83, NoS84, PeS86]. These pleasant properties of the theory have certainly contributed to the interest for it arisen in the computer science community, especially among those people who believe that program correctness is a major concern in the programming activity [Bac89]. Actually the theory which is quite well known is the one presented in [Mar82, Mar84]. This is the theory we shall consider in this paper and refer to as Martin-Löf's Intuitionistic Type Theory (ITT), even if successive variations have been developed. Sometime, ITT is referred to as the polymorphic theory opposite to the last version [Nor89] which is monomorphic, i.e. each element can be uniquely typed, and decidable.

In this paper we shall present a small extension of ITT whose principal characteristic consists in the possibility of assuming variables denoting higher order functions. Our main motivation in developing this higher order version (HITT) has been the wish to complete the way first opened by Per Martin-Löf. Indeed in the preface of [Mar84], while referring to a series of lectures given in Munich (October 1980), he writes: "The main improvement of the Munich lectures, compared with those given in Padova, was the adoption of a systematic higher level notation ....". This notation is called *expressions with arity* and yields a more uniform and compact writing of the rules of the theory. An expression with arity is built up starting from primitive constants and variables with arity, by means of abstractions and applications. The arity associated to an expression specifies its functionality and constrains the applications, analogously to what the type does for typed lambda-calculus. In our opinion, in order to fully exploit this approach and be able to establish formal properties of the system, it is necessary to extend the formalization of the contexts as given in [Mar84] to assumptions of variables of higher arity. Therefore we have defined this extension that, even if conservative, supplies increased expressivity, advantageous especially when the theory is viewed as a programming and a specification language. In fact, assuming a variable of higher arity corresponds to

[1] Dipartimento di Matematica Pura ed Applicata, via Belzoni 7, I-35131 Padova (ITALY)

[2] Dipartimento di Scienze dell'Informazione, via Moretto da Brescia 9, I-20133 Milano (ITALY)

assuming the possibility of putting together pieces of programs, thus supporting a modular approach in program development [BoV87].

Some properties of HITT are also proved, the principal ones are the consistency of HITT, which also implies the consistency of ITT, and the computability of any judgement derived within HITT. Besides we proved a canonical form theorem: to any derivable judgement we can associate a canonical one whose derivation ends with an introductory rule. This result, even if weaker than a standard normalization theorem [Pra65], suffices to obtain all the useful consequences typical of a normal form theorem, mainly the consistency of the theory. Moreover, by using the computational interpretation of types (i.e. types as problem descriptions and their elements as programs to solve those problems) it immediately follows that the execution of any proved correct program terminates.
We assume the reader is familiar with the Intuitionistic Theory of Types as presented in [Mar82, Mar84] and with typed lambda calculus [Bar81], or enjoyably, with the theory of expressions with arity [BoV85, Nor89].

The following is the outline of the paper. In section 2. our characterization of assumptions of variables of any arity is presented and some consequences of this are briefly sketched. We are extremely grateful to Prof. Aczel for his suggestions on the notation to be used for the new kind of assumptions. Further comments on the properties of our system are given in section 3. Section 4. deals with computability. The definition of computable judgement, which is the basis for the computability theorem, is first given. The rest of the section is devoted to prove that each rule of the theory preserves this property. The computability theorem, as well as some significative corollaries, is presented in the concluding section 5. All the rules of HITT are listed in the appendix. Compared with ITT, besides the changes concerning the assumptions of higher level variables, there are also changes in the notation and in the fact that we explicitly added to the premises of a rule all the requirements that were only informally expressed in ITT.

2. A formulation of Intuitionistic Type Theory with assumptions of high level arity variables.

We assume the theory of expressions with arity [Bee85, BoV85, Nor89] developed by Martin Löf in order to give an uniform and compact presentation of his theory of types. The theory has many similarities with typed lamba-calculus [Bar81] and some familiarity with this system should be sufficient to understand what follows. An expression with arity is built up starting from primitive constants and variables with arity, by means of abstractions and applications. The arity associated to an expression fully specifies its functionality, i.e. it indicates the number and the arity of the expressions to which it can be applied, analogously to what the type does for typed lambda-calculus.

The Intuitionistic Theory of Types [Mar82] consists of a language of constant symbols, each of some arity, a system of computation rules and a system of rules of inference for deriving judgements. Each instance of a rule of inference has the form

$$\frac{J_1 \ ... \ J_n}{J}$$

where $J_1,...,J_n,J$ are judgements. A derivation is a tree of judgements built up in the usual way using instances of the rules of inference. Judgements have the form

$$F[C]$$

where C is a "context", and F has one of the forms

$$A \ \textbf{type}$$
$$A=B$$
$$a \quad A$$
$$a=b \quad A$$

Here A, B, a, b are expressions of arity **0**. A context is a list A_1, ..., A_n of assumptions where, for j=1,...,n, A_j is an assumption over the context $A_1,...,A_{j-1}$. We will call "order between assumptions condition" this requirement on the assumptions of a context[3]. When the context is empty we write only F instead of F[ ], and call J a "closed" judgement as opposed to "hypothetical" judgement, i.e. with non-empty context. In the following we will say that the context C' extends the context C if C' is obtained from C by adding some assumptions satisfying the "order between assumptions" condition. Each assumption has the form

$$x\mathbf{:}A \ [C]$$

where x is a variable of some arity, A is an expression of arity **0** and C is a context. We call x the variable of the assumption and its arity is the arity of the assumption. The variables of the assumptions of a context are also called the variables of the context. They must be pairwise distinct. We will say that the assumption of a variable x depends on all the assumptions of the context. The conditions for forming an assumption over a context involve the notion of derivation, so that the contexts and derivations have to be defined by simultaneous inductive definition. The simple case of an assumption

$$y\mathbf{:}B \ [C]$$

of arity **0** over a context C is the familiar one defined by Martin Löf in the original theory [Mar84]. The conditions are that the judgement

$$B \ \textbf{type} \ [C]$$

should be derivable and that y should not be a variable of C. It is easy to convince ourselves that these conditions are just a formalization of those usually asked for making an assumption in a natural deduction system. The variable y keeps the place of a generic object of type B.

To deal with assumptions with arities of higher level we add to the language, for each arity $\alpha=(\alpha_1, \ldots, \alpha_n)$, a constant $\mathbf{T}^\alpha$ of arity

$$(0,(\alpha_1),(\alpha_1)(\alpha_2),..,(\alpha_1)..(\alpha_{n-1}),(\alpha_1)..(\alpha_n)),$$

and if $A_1,...,A_n, A(x_1,..,x_n)$ are expressions of arity **0** and $x_1,...,x_n$ are distinct variables of arities α_1, ..., α_n respectively then we write

[3] As in [Bru91] contexts are inserted like ‘telescopes’.

$$(x_1{:}A_1, .., x_n{:}A_n)\ A(x_1, .., x_n)$$

for the expression of arity **0**

$$\mathbf{T}^{\alpha}(A_1, (x_1)A_2, ..., (x_1, ..., x_{n-1})A_n, (x_1, ..., x_n)\ A(x_1,.., x_n)).$$

If

(*) $B \equiv (x_1{:}A_1,.., x_n{:}A_n)\ A(x_1,.., x_n)$

we write

$\mathbf{:}B\ [C]$

for the judgement

(**) $A(x_1,..,x_n)$ **type** $[C, x_1{:}A_1,.., x_n{:}A_n]$.

When C is empty we write only $:B$ instead of $\mathbf{:}B[\ ]$.

We can now state the conditions for forming the assumption $y\mathbf{:}B[C]$ of arity $\alpha=(\alpha_1,...,\alpha_n)$. These conditions are that

- (*) holds for some choice of variables $x_1, ..., x_n$ not free in B and in C and some expressions $A_1,..., A_n, A(x_1,..,x_n)$ of arity **0**.
- (**) is derivable.
- y is a variable of arity α that is not a variable of the context $C, x_1{:}A_1,.., x_n{:}A_n$.

As an example suppose

$x{:}A\ [C]$

$y{:}\mathbf{T}^{(0)}(V(x),(v)\ B(x,v))[C, x{:}A]$

are correct assumptions. The variable x keeps the place of a generic object of type A, while the variable y keeps the place of a function from a generic object v of type $V(x)$ to objects of type $B(x,v)$. Now the assumption of a variable z, which keeps the place of a function mapping objects x and functions y to objects of type $C(x,y)$

$z\mathbf{:}\ \mathbf{T}^{(0,(0))}(A, (x)\ \mathbf{T}^{(0)}(V(x), (v)\ B(x, v)), (x, y)\ C(x, y))\ [C]$

or, using the abbreviation,

$z\mathbf{:}\ (x{:}A, y{:}\ (v{:}\ V(x))\ B(x,v))\ C(x,y)\ [C]$

is correct if the judgement

$C(x, y)$ **type** $[C,\ x{:}\ A, y{:}\ (v{:}\ V(x))\ B(x, v)]$

is derivable and z does not occur in $[C, x{:}\ A, y{:}\ (v{:}\ V(x))\ B(x, v)]$

In writing the inference rules we will adopt Martin Löf's convention to present explicitly only those assumptions that do not occur in both premises and conclusion. Hence all the assumptions appearing in the premises are to be considered discharged by the application of the rule. Clearly, as usual, the remaining assumptions of the context should not depend on the discharged ones, i.e. they must be an initial segment in the ordered list of assumptions of the context. Moreover we mean that the context of the conclusion of a rule is obtained by merging, without duplication, the contexts (not explicitly present) of the assumptions and (possibly) the assumptions explicitly present in the context of the conclusion.

The assumption rules introduce a new assumption in the context of the conclusion. In order to formulate these rules it is convenient to introduce some abbreviations. When there is a derivation of $:B$ $[C]$ then we use

$$b\textbf{:}\ B\ [C]$$

to abbreviate the judgement

$$b(x_1, ..., x_n) \quad A(x_1,..,x_n)\ [C, x_1{:}A_1, ..., x_n{:}A_n],$$

where the variables $x_1,...,x_n$, of arities $\alpha_1,...,\alpha_n$ respectively, are chosen not free in B and C, so that (*) holds, and b is an expression of arity $(\alpha_1, ..., \alpha_n)$ in which they may appear only variables of the context C.

Similarly we use

$$b{=}b'\textbf{:}\ B[C]$$

to abbreviate the judgement

$$b(x_1, ..., x_n){=}b'(x_1, ..., x_n) \quad A(x_1,..,x_n)\ [C, x_1{:}A_1, ..., x_n{:}A_n]$$

Now, if $y{:}B[C]$ is an assumption, then we have the following assumption rules:

$$\frac{a_1{:}\underline{A}_1\ ...\ a_n{:}\underline{A}_n \quad \textbf{:}B}{y(a_1, ..., a_n) \quad A(a_1,..,a_n)[y{:}B]} \qquad \frac{a_1{=}a_1'{:}\underline{A}_1 \quad ... \quad a_n{=}a_n'{:}\underline{A}_n \quad :B}{y(a_1, ..., a_n){=}y(a_1', ..., a_n') \quad A(a_1,..,a_n)[y{:}B]}$$

where, for j=1, ..., n, $\underline{A}_j \equiv ((x_1, ..., x_{j-1})A_j)(a_1, ..., a_{j-1})$

Note that, in both cases, in the conclusion appears the new assumption $y{:}B$ while in the premises there may appear assumptions which are discharged by the rule.

As an example consider again the assumption

$z{:}\ (x{:}\ A, y{:}\ (v{:}\ V(x))\ B(x, v))\ C(x, y)\ [C]$

and suppose that the judgements:

1) $:(x{:}A, y{:}(v{:}V(x))\ B(x,v))\ C(x,y)\ [C]$, i.e. $C(x,y)$ **type** $[C, x{:}A, y{:}(v{:}V(x))\ B(x,v)]$
2) $a{:}A\ [C_1]$, that is $a \quad A\ [C_1]$
3) $b{:}(s{:}V(a))B(a,s)\ [C_2]$, that is $b(s) \quad B(a,s)[C_2, s{:}V(a)]$

are all derivable judgements, then

$$\frac{a \quad A \quad b(s) \quad B(a,s)[s{:}V(a)] \quad C(x,y)\ \textbf{type}\ [x{:}A, y{:}(v{:}V(x))\ B(x,v)]}{z(a,b) \quad C(a,b)[z{:}(x{:}A, y{:}(v{:}V(x))\ B(x,v))\ C(x,y)]}$$

is an instance of the assumption rule. The context of the conclusion is the merge, without duplication, of the four context C, C_1, C_2, $[z{:}(x{:}A, y{:}(v{:}V(x))\ B(x,v))\ C(x,y)]$. The assumptions of the variables s, x and y are discharged by the rule while the assumption of z is possibly introduced.

The given abbreviations for the hypothetical judgements have the nice consequence of allowing a notation quite close to that used by P. Martin-Löf [Mar84] for variable's substitution, also in the case of high-arity variables. To express the fact that a sequence of variables can be substituted by a given sequence of expressions, we introduce the following concept of *fitting substitutions*.

Definition: (fitting substitution)

The sequences of judgements $b_1{:}\underline{B}_1[C],\ldots,b_n{:}\underline{B}_n[C]$ and $b_1=b'_1{:}\underline{B}_1[C],\ldots,b_n=b'_n{:}\underline{B}_n[C]$, where $\underline{B}_i=((y_1,\ldots,y_{i-1})B_i)(b_1,\ldots,b_n)$, are *substitutions that fit with* any context $[C, y_1{:}B_1,\ldots, y_n{:}B_n]$.

Note that a similar concept of "fitting" is already used in [Bru91] where only variables of arity **0** are considered.

2.1 Modifications due to the new assumptions.

Clearly, the new form of assumptions compel us to extend the substitution rules. They are listed in the appendix among all the other rules but let us analyze an example. The following substitution rule of the previous version

$$\frac{b \quad B \qquad d(y) \quad D(y)[y{:}B]}{d(b) \quad D(b)}$$

which involves a variable y of arity **0**, is extended to the new rule

$$\frac{b_1{:}\ \underline{B}_1 \ \ldots\ b_n{:}\ \underline{B}_n \qquad d(y_1,..,y_n) \quad D(y_1,..,y_n)[y_1{:}\ B_1,..y_n{:}\ B_n]}{d(b_1,.., b_n) \quad D(b_1,.., b_n)}$$

where $\underline{B}_i\equiv((y_1,.., y_{i-1})B_i)(b_1,.., b_{i-1})$, i.e., $b_1{:}\underline{B}_1 \ldots b_n{:}\underline{B}_n$ is a substitution that fits with the context $[y_1{:}B_1,..y_n{:}B_n]$. (If no confusion can arise we will use the abbreviation $\underline{b}$: or $\underline{b}=\underline{c}$: to denote a substitution that fits with a given context). The pattern is rather similar but now the variables $y_1,...,y_n$ may have any arity.

The changes that can be made on other rules are more fundamental. For example, let us analyze the **W**-elimination rule. In the previous version the **W**-elimination rule was (adopting our notation)

$$\frac{c \quad \mathbf{W}(A,B) \quad d(x,y,z) \quad C(\mathbf{sup}(x,y))[x{:}A, y{:}\prod(B(x),(t)\mathbf{W}(A,B)), z{:}\prod(B(x),(u)C(\mathbf{Ap}(y,u)))]}{\mathbf{T}(c,d) \quad C(c)}$$

while the new one is

$$\frac{c \quad \mathbf{W}(A,B) \quad d(x,y,z) \quad C(\mathbf{sup}(x, y))[x{:}A, y{:}(t{:}B(x))\mathbf{W}(A,B), z{:}(u{:}B(x))C(y(u))]}{\mathbf{T}(c,d) \quad C(c)}$$

which is conceptually more straight. In fact while in the previous version y and z stands for functions (i.e. elements of a $\prod$_type), here they are functional expressions: this avoid the vicious application-abstraction circle, as in $(u)C(\mathbf{Ap}(y,u))$, which can now be simply expressed by the application of expressions $C(y(u))$ since it is possible to assume the variable y of arity (**0**)**0**.

3. Some observations on type theory

In the next section we will frequently use some concepts and properties of HITT that we will briefly describe here.

3.1 Associate judgements

Our rules differ from the ones introduced in [Mar84] both for the use of assumptions of variables of higher arity and because when a type A appears in the conclusion of a rule the premisses of the rule are augmented with those of the formation rule for the judgement *A* **type**. This requirement allows us to easily prove the following theorem 3.2 which shows a strong property on the structure of the derivable judgements of HITT. We introduce first the notion of associate judgements.

Definition 3.1: (associate judgements)

The associate judgement(s) of

$a \quad A\ [x_1{:}A_1,\ldots,x_n{:}A_n]$ is	A **type** $[x_1{:}A_1,\ldots,x_n{:}A_n]$;	
$A{=}B\ [x_1{:}A_1,\ldots,x_n{:}A_n]$ are	A **type** $[x_1{:}A_1,\ldots,x_n{:}A_n]$ and	
		B **type** $[x_1{:}A_1,\ldots,x_n{:}A_n]$;
$a{=}b \quad A\ [x_1{:}A_1,\ldots,x_n{:}A_n]$	are	$a \quad A\ [x_1{:}A_1,\ldots,x_n{:}A_n]$ and
		$b \quad A\ [x_1{:}A_1,\ldots,x_n{:}A_n]$.

Theorem 3.2: (derivability of associate judgements)

The associate judgements of a derivable judgement are derivable

Proof: The three cases should be proved simultaneously and the proof follows almost immediately by induction on the length of the derivation of the considered judgement. Only in some cases structural rules or substitution rules should be carefully used.

Actually, to obtain the previous result it would not be necessary to add in each rule the premises of the formation rule of the judgement *A* **type**, for instance they are superfluous in the Π_introduction rule. We inserted this redundancies for sake of uniformity in view of proving general properties of the theory at a more abstract level.

3.2 Substituted judgements

Substitution is a central operation on judgements. Many concepts we shall introduce in the next section will be based on the two kinds of substitutions we define now.

Definition 3.3: (tail substituted judgements)

Let $D \equiv [C, x_1{:}\ A_1,\ldots,x_n{:}\ A_n]$ be a context, $a_1{:}\ \underline{A}_1[C],\ldots, a_n{:}\ \underline{A}_n[C]$ and $a_1{=}a'_1{:}\ \underline{A}_1[C],\ldots, a_n{=}a'_n{:}\ \underline{A}_n[C]$ be substitutions that fit with the last n assumption in the context *D*, $J \equiv F[D\ ]$ then

1. $J[x_1{:=}a_1,\ldots,x_n{:=}a_n]$ is an abbreviation for the tail substituted judgement of J :

   1.1 $((x_1,\ldots,x_n)A)(a_1,\ldots,a_n)$ **type**$[C]$ if $F \equiv A$ **type**

   1.2 $((x_1,\ldots,x_n)A)(a_1,\ldots,a_n){=}((x_1,\ldots,x_n)B)(a_1,\ldots,a_n)[C]$ if $F \equiv A{=}B$

   1.3 $((x_1,\ldots,x_n)a)(a_1,\ldots,a_n) \quad ((x_1,\ldots,x_n)A)(a_1,\ldots,a_n)[C]$ if $F \equiv a \quad A$

   1.4 $((x_1,\ldots,x_n)a)(a_1,\ldots,a_n){=}((x_1,\ldots,x_n)b)(a_1,\ldots,a_n) \quad ((x_1,\ldots,x_n)A)(a_1,\ldots,a_n)[C]$
   if $F \equiv a{=}b \quad A$

2. $J[x_1{<-}a_1{=}a'_1,\ldots,x_n{<-}a_n{=}a'_n]$ is an abbreviation for the tail substituted judgement of J :

   2.1 $((x_1,\ldots,x_n)A)(a_1,\ldots,a_n){=}((x_1,\ldots,x_n)A)(a'_1,\ldots,a'_n)[C]$ if $F \equiv A$ **type**

   2.2 $((x_1,\ldots,x_n)A)(a_1,\ldots,a_n){=}((x_1,\ldots,x_n)B)(a'_1,\ldots,a'_n)[C]$ if $F \equiv A{=}B$

   2.3 $((x_1,\ldots,x_n)a)(a_1,\ldots,a_n){=}((x_1,\ldots,x_n)a)(a'_1,\ldots,a'_n) \quad ((x_1,\ldots,x_n)A)(a_1,\ldots,a_n)[C]$
   if $F \equiv a \quad A$

2.4 $((x_1,\ldots,x_n)a)(a_1,\ldots,a_n)=((x_1,\ldots,x_n)b)(a'_1,\ldots,a'_n)\ \ \ ((x_1,\ldots,x_n)A)(a_1,\ldots,a_n)[C]$

if $F \equiv a=b\ \ \ A$

In general, the substitutions rules of HITT are sufficient to prove the following theorem.

Theorem 3.4:

The tail substituted judgements of a derivable judgement are derivable.

Proof: Just apply the suitable substitution rule except for the cases 2.2 and 2.4. For these cases first note that if $a_1=a'_1{:}\underline{A}_1[C]$,… , $a_n=a'_n{:}\underline{A}_n[C]$ is a substitution that fits with the tail of D, then also $a_1{:}\underline{A}_1[C]$,… , $a_n{:}\underline{A}_n[C]$, whose derivability is showed by theorem 3.2, is a substitution that fits with the tail of D. Then apply the suitable substitution rule (:=) to $A=B[D]$ (case 2.2) or to $a=b\ \ \ A[D]$ (case 2.4) and the suitable substitution rule (<−) to B **type**$[D]$ (case 2.2) or to $b\ \ \ A[D]$ (case 2.4), which are the associate of $A=B[D]$ and $a=b\ \ \ A[D]$ respectively. The result follows by transitivity.

The substitution of the second kind does not rely directly on the substitutions rules. It substitutes the first part of a context and consequentially it modifies not only the form part of the judgement but also the tail of the context.

Definition 3.5: (head substituted judgements)

Let $D \equiv [x_1{:}\ A_1,\ldots,x_n{:}\ A_n]$ be a context, $a_1{:}\ \underline{A}_1,\ldots,a_i{:}\ \underline{A}_i$ and $a_1=a'_1{:}\ \underline{A}_1\ldots,a_i=a'_i{:}\ \underline{A}_i$, i ≤ n, be substitutions that fit with the first i assumptions of D, $J \equiv F[D\ ]$. Let $A'_j \equiv((x_1,\ldots,x_i)A_j)(a_1,\ldots,a_i)$, i+1 ≤j ≤n, then

1. $J[x_1{:=}a_1,\ldots,x_i{:=}a_i]$ is an abbreviation for the head substituted judgement of J:

1.1 $((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)$ **type**$[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$ if $F \equiv A$ **type**

1.2 $((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)=((x_1,\ldots,x_i)B)(a_1,\ldots,a_i)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv A=B$

1.3 $((x_1,\ldots,x_i)a)(a_1,\ldots,a_i)\ \ \ ((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv a\ \ \ A$

1.4 $((x_1,\ldots,x_i)a)(a_1,\ldots,a_i)=((x_1,\ldots,x_i)b)(a_1,\ldots,a_i)\ \ \ ((x_1,\ldots,x_i)A)(a_1,\ldots,a_n)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv a=b\ \ \ A$

2. $J[x_1{<-}a_1=a'_1,\ldots,x_i{<-}a_i=a'_i]$ is an abbreviation for the head substituted judgement of J:

2.1 $((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)=((x_1,\ldots,x_i)A)(a'_1,\ldots,a'_i)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv A$ **type**

2.2 $((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)=((x_1,\ldots,x_i)B)(a'_1,\ldots,a'_i)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv A=B$

2.3 $((x_1,\ldots,x_i)a)(a_1,\ldots,a_i)=((x_1,\ldots,x_i)a)(a'_1,\ldots,a'_i)\ \ \ ((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv a\ \ \ A$

2.4 $((x_1,\ldots,x_i)a)(a_1,\ldots,a_i)=((x_1,\ldots,x_i)b)(a'_1,\ldots,a'_i)\ \ \ ((x_1,\ldots,x_i)A)(a_1,\ldots,a_i)[x_{i+1}{:}A'_{i+1},\ldots,x_n{:}A'_n]$

if $F \equiv a=b\ \ \ A$

<u>Theorem 3.6</u>:

The head substituted judgements of a derivable judgement are derivable.

Proof: For case 1. note that if a_1: $\underline{A}_1,\ldots, a_i$: $\underline{A}_i$ is a substitution that fits with $[x_1{:}A_1,\ldots, x_i{:}A_i]$ then a_1: $\underline{A}_1,\ldots, a_i$: $\underline{A}_i$, $x_{i+1}{:}\underline{A}_{i+1},\ldots, x_n{:}\underline{A}_n$ is a substitution that fits with $[x_1{:}A_1,\ldots, x_n{:}A_n]$. Hence the result follows by using the suitable substitution rule. For case 2., if $a_1=a'_1$: $\underline{A}_1,\ldots, a_i=a'_i$: $\underline{A}_i$ is a substitution that fits with $[x_1{:}A_1,\ldots, x_i{:}A_i]$ then $a_1=a'_1$: $\underline{A}_1,\ldots, a_i=a'_i$: $\underline{A}_i$, $x_{i+1}=x_{i+1}{:}\underline{A}_{i+1},\ldots, x_n=x_n{:}\underline{A}_n$ is a substitution that fits with $[x_1{:}A_1,\ldots, x_n{:}A_n]$. Hence the result follows by using directly the suitable substitution rule except for the subcases 2.2 and 2.4 where the associate judgements must be considered (see theorem 3.4).

Note that we use the same notation for the head and the tail substitutions since the names of the variables and their positions in the context are sufficient to determine the kind of substitution we want to perform.

3.3 The evaluation tree

In HITT, a set of computation rules is associated to each defined type such as Π, Σ, etc. They specify a process for evaluating expressions denoting elements or types. They apply to variable-free and saturated expressions, i.e. expressions of arity **0** in which no variable occurs free. The "normal form" theorem for expressions [BoV85], assures us that a variable-free, saturated expression is always definitionally equivalent to an expression of the form $\mathbf{c}(a_1,\ldots, a_n)$ where $\mathbf{c}$ is a constant. Hence, to evaluate an expression, we first consider its normal form and then detect the suitable computation rule. This can be done by looking at the outermost constant of the expression in normal form and, only in some cases, at the value of its first argument. Then each premise of the selected rule indicate how recursively to continue the process. Clearly, the process of evaluating an expression denoting an element or a type using the computation rules naturally gives rise to a finitary tree: we will refer to it as *the evaluation tree*. Of course an expression evaluates if and only if its evaluation tree is finite. Hence if we know that an expression can be evaluated an induction on the depth of its evaluation tree is a correct proof-method. It can be used to prove the following:

<u>Theorem 3.7:</u>

Let c and C be variable-free and saturated expressions. Then

1) If $c \quad g$ then g is a canonical expression for an element, i.e. exactly one of the following holds: $g\equiv\lambda(b)$, $g\equiv\mathbf{\&}(a,b)$, $g\equiv\mathbf{i}(a)$, $g\equiv\mathbf{j}(b)$, $g\equiv\mathbf{r}$, $g\equiv\mathbf{m_n}$, $g\equiv\mathbf{0}$, $g\equiv\mathbf{s}(a)$, $g\equiv\mathbf{sup}(a,b)$, $g\equiv$❹(a,b), $g\equiv\sigma(a,b)$, $g\equiv+(a,b)$, $g\equiv\mathbf{i}(a,b,d)$, $g\equiv\mathbf{n_n}$, $g\equiv\mathbf{n}$, $g\equiv\mathbf{w}(a,b)$.
2) If C»G then G is a canonical expression for a type, i.e. exactly one of the following holds: $G\equiv\prod(A,B)$, $G\equiv\Sigma(A,B)$, $G\equiv+(A,B)$, $G\equiv\mathbf{I}(A,b,d)$, $G\equiv\mathbf{N_n}$, $G\equiv\mathbf{N}$, $G\equiv\mathbf{W}(A,B)$, $G\equiv\mathbf{U}$.

Note that the objects in the conclusion of a formation rule or an introduction rule are always denoted by canonical expressions. We will call them canonical elements or canonical types respectively. However a canonical expression does not necessarily denote a canonical element or a canonical type. The successive normalization theorem will certify this whenever we consider judgements derived within the theory. More precisely, if the judgement *a*   *A* (or *A* **type**) is derived within the theory, then the canonical expression resulting from the evaluation of the expression *a* (or *A*) denotes a canonical element (or a canonical type). Moreover, under the same hypothesis, the evaluation process of the expression *a* (or *A*) always terminates.

Finally let us also observe that, since the computation rules do not "add" variables, it is obvious that if no variable appears in *a* (respectively *A*) and *a*   *g* (resp. *A*»*G*), then no variable appears in *g* (resp. *G*).

4. Computability

In this section we introduce the main notions of the paper: the definitions of computation tree and computable judgement.

To prove a canonical-form theorem for the system we are considering, and whose complete set of rules is reported in the Appendix, we will follow a proof style similar to the one used by Martin-Löf [Mar71] based on the method of Tait [Tai67] to prove normalization theorems. Therefore we will introduce the notion of computable judgement. This notion applies both to closed judgements and to hypothetical ones. Essentially, to express the computability of a judgement is equivalent to express what it is necessary to know in order to be allowed to formulate that judgement. Hence the definition formally summarizes the meaning of all the forms of judgements which can be obtained by a derivation in type theory. Of course, it is directly inspired by the informal explanation of the rules given in [Mar84], but the needs of formalization make it a very long definition. We will base it on the concept of computation tree which represents the full process needed to recognize the computability of a given judgement. The nodes of a computation tree are labelled by derivable judgements and if *J* is the label of a node then the labels of its sons are all the judgements whose computability is required in order to establish the computability of *J*.

As regards hypothetical judgements, their computability is referred to the computability of any closed judgement that can be obtained by substituting, in any possible way, computable judgements to the open assumptions.

As regards closed judgements, the definition obviously differs when considering one form of judgement or another. Still there are some basic common points:

- any term appearing in the judgement must be (syntactically) valuable (evaluation) to a canonical term. This requirement is directly expressed for the two forms *A* **type** and *a*   *A* and indirectly, by requiring the computability of the associate judgements (associate), for the forms *A*=*B* and *a*=*b*   *A*.
- the equality between a term and its corresponding evaluated form must be a provable judgement (correct evaluation)
- the computability of a judgement is recursively referred to the computability of the components (parts) of the judgement built up with the evaluated canonical terms.

4.1 The main definitions

<u>Definition 4.1</u>: (Computable judgement)

The judgement $J \equiv F[C]$ is computable if it is derivable and

<u>Case 1</u>: There is no assumption, i.e. the context C is empty

<u>Subcase 1.1</u>: $F \equiv A$ **type** then

1.1.1) (*evaluation*) $A \gg G_A$

1.1.2) (*correct evaluation*) the judgement $A=G_A$ is provable

1.1.3) (*parts*) the parts of G_A are computable type(s), i.e.

- if $G_A \equiv \prod(A_1,A_2)$ then the judgements A_1 **type** and $A_2(x)$ **type**$[x:A_1]$ are computable
- if $G_A \equiv \Sigma(A_1,A_2)$ then the judgements A_1 **type** and $A_2(x)$ **type**$[x:A_1]$ are computable
- if $G_A \equiv +(A_1,A_2)$ then the judgements A_1 **type** and A_2 **type** are computable
- if $G_A \equiv \mathbf{I}(A_1,b,d)$ then the judgements A_1 **type**, $b \quad A_1$ and $d \quad A_1$ are computable
- if $G_A \equiv \mathbf{N_n}$ no condition
- if $G_A \equiv \mathbf{N}$ no condition
- if $G_A \equiv \mathbf{W}(A_1,A_2)$ then the judgements A_1 **type** and $A_2(x)$ **type**$[x:A_1]$ are computable
- if $G_A \equiv \mathbf{U}$ (i.e. $A \equiv \mathbf{U}$) no condition

<u>Subcase 1.2</u>: $F \equiv A=B$ then

1.2.1) (*associate judgements*) the associate judgements A **type** and B **type** are computable (and hence $A \gg G_A$ and $B \gg G_B$).

1.2.2) (*parts*) G_A and G_B are equal computable types, i.e.

- $G_A \equiv \prod(A_1,A_2)$ iff $G_B \equiv \prod(B_1,B_2)$ and the judgements $A_1=B_1$ and $A_2(x)=B_2(x)[x:A_1]$ are computable
- $G_A \equiv \Sigma(A_1,A_2)$ iff $G_B \equiv \Sigma(B_1,B_2)$ and the judgements $A_1=B_1$ and $A_2(x)=B_2(x)[x:A_1]$ are computable
- $G_A \equiv +(A_1,A_2)$ iff $G_B \equiv +(B_1,B_2)$ and the judgements $A_1=B_1$ and $A_2=B_2$ are computable
- $G_A \equiv \mathbf{I}(A_1,a,c)$ iff $G_B \equiv \mathbf{I}(B_1,b,d)$ and the judgements $A_1=B_1$, $a=b \quad A_1$ and $c=d \quad A_1$ are computable
- $G_A \equiv \mathbf{N_n}$ iff $G_B \equiv \mathbf{N_n}$
- $G_A \equiv \mathbf{N}$ iff $G_B \equiv \mathbf{N}$
- $G_A \equiv \mathbf{W}(A_1,A_2)$ iff $G_B \equiv \mathbf{W}(B_1,B_2)$ and the judgements $A_1=B_1$ and $A_2(x)=B_2(x)[x:A_1]$ are computable
- $G_A \equiv \mathbf{U}$ iff $G_B \equiv \mathbf{U}$

<u>Subcase 1.3</u>: $F \equiv c \quad A$ then

1.3.1) (*associate judgements*) The associate judgement A **type** is computable (and hence $A \gg G_A$)

1.3.2) (*evaluation*) $c \quad g$

1.3.3) (*correct evaluation*) $c=g \quad A$ is provable

1.3.4) (*parts*) the parts of g are computable element(s) in G_A, i.e.

- $G_A \equiv \prod(A_1,A_2)$ iff $g \equiv \lambda(b)$ and the judgement $b(x) \quad A_2(x)[x:A_1]$ is computable

• $G_A \equiv \Sigma(A_1,A_2)$ iff $g \equiv \mathbf{\&}(a,b)$ and the judgements
$a \quad A_1$ and $b \quad A_2(a)$ are computable

• $G_A \equiv +(A_1,A_2)$ iff either $g \equiv \mathbf{i}(a)$ and the judgement $a \quad A_1$ is computable
or $g \equiv \mathbf{j}(b)$ and the judgement $b \quad A_2$ is computable

• $G_A \equiv \mathbf{I}(A_1,b,d)$ iff $g \equiv \mathbf{r}$ and the judgement $b=d \quad A_1$ is computable

• $G_A \equiv \mathbf{N_n}$ iff $g \equiv \mathbf{m_n}$ for some 0≤m≤n-1

• $G_A \equiv \mathbf{N}$ iff either $g \equiv \mathbf{0}$
or $g \equiv \mathbf{s}(a)$ and the judgement $a \quad \mathbf{N}$ is computable

• $G_A \equiv \mathbf{W}(A_1,A_2)$ iff $g \equiv \mathbf{sup}(a,b)$ and the judgements
$a \quad A_1$ and $b(x) \quad \mathbf{W}(A_1,A_2)[x{:}A_2(a)]$ are computable

• $G_A \equiv \mathbf{U}$ iff either $g \equiv ❹(a,b)$ and the judgements
$a \quad \mathbf{U}$ and $b(x) \quad \mathbf{U}[x{:}<a>]$ are computable
or $g \equiv \sigma(a,b)$ and the judgements
$a \quad \mathbf{U}$ and $b(x) \quad \mathbf{U}[x{:}<a>]$ are computable
or $g \equiv +(a,b)$
and the judgements $a \quad \mathbf{U}$ and $b \quad \mathbf{U}$ are computable
or $g \equiv \mathbf{i}(a,b,d)$ and the judgements
$a \quad \mathbf{U}$, $b \quad <a>$ and $d \quad <a>$ are computable
or $g \equiv \mathbf{n_n}$
or $g \equiv \mathbf{n}$
or $g \equiv \mathbf{w}(a,b)$ and the judgements
$a \quad \mathbf{U}$ and $b(x) \quad \mathbf{U}[x{:}<a>]$ are computable

<u>Subcase 1.4</u>: $F \equiv a=b \quad A$ then

1.4.1) (*associate judgements*) the associate judgements $a \quad A$ and $b \quad A$ are computable
(hence $a \quad g_a$, $b \quad g_b$ and $A \rangle\!\rangle G_A$).

1.4.2) (*parts*) the parts of g_a and g_b are computable equal elements in G_A, i.e.

• $G_A \equiv \prod(A_1,A_2)$ iff $g_a \equiv \lambda(a')$ and $g_b \equiv \lambda(b')$ and the judgement
$a'(x)=b'(x) \quad A_2(x)[x{:}A_1]$ is computable

• $G_A \equiv \Sigma(A_1,A_2)$ iff $g_a \equiv \mathbf{\&}(a',a'')$ and $g_b \equiv \mathbf{\&}(b',b'')$ and the judgements
$a'=b' \quad A_1$ and $a''=b'' \quad A_2(a')$ are computable

• $G_A \equiv +(A_1,A_2)$ iff either $g_a \equiv \mathbf{i}(a')$ and $g_b \equiv \mathbf{i}(b')$
and the judgement $a'=b' \quad A_1$ is computable
or $g_a \equiv \mathbf{j}(a'')$ and $g_b \equiv \mathbf{j}(b'')$
and the judgement $a''=b'' \quad A_2$ is computable

• $G_A \equiv \mathbf{I}(A_1,c,d)$ iff $g_a \equiv \mathbf{r}$ and $g_b \equiv \mathbf{r}$ and the judgement $c=d \quad A_1$ is computable

• $G_A \equiv \mathbf{N_n}$ iff $g_a \equiv \mathbf{m_n}$ and $g_b \equiv \mathbf{m_n}$ for some 0≤m≤n-1

• $G_A \equiv \mathbf{N}$ iff either $g_a \equiv \mathbf{0}$ and $g_b \equiv \mathbf{0}$
or $g_a \equiv \mathbf{s}(a')$ and $g_b \equiv \mathbf{s}(b')$
and the judgement $a'=b' \quad \mathbf{N}$ is computable

• $G_A \equiv \mathbf{W}(A_1,A_2)$ iff $g_a \equiv \mathbf{sup}(a',a'')$ and $g_b \equiv \mathbf{sup}(b',b'')$

and the judgements $a'=b' \quad A_1$

and $a''(x)=b''(x) \quad \mathbf{W}(A_1,A_2)[x{:}A_2(a')]$ are computable

• $G_A \equiv \mathbf{U}$ iff either $g_a \equiv$❹(a',a'') and $g_b \equiv$❹(b',b'') and the judgements

$a'=b' \quad \mathbf{U}$ and $a''(x)=b''(x) \quad \mathbf{U}[x{:}<a'>]$ are computable

or $g_a \equiv \sigma(a',a'')$ and $g_b \equiv \sigma(b',b'')$ and the judgements

$a'=b' \quad \mathbf{U}$ and $a''(x)=b''(x) \quad \mathbf{U}[x{:}<a'>]$ are computable

or $g_a \equiv +(a',a'')$ and $g_b \equiv +(b',b'')$ and the judgements

$a'=b' \quad \mathbf{U}$ and $a''=b'' \quad \mathbf{U}$ are computable

or $g_a \equiv \mathbf{i}(a',c,d)$ and $g_b \equiv \mathbf{i}(b',e,f)$ and the judgements

$a'=b' \quad \mathbf{U}$, $c=e \quad <a'>$ and $d=f \quad <a'>$ are computable

or $g_a \equiv \mathbf{n_n}$ and $g_b \equiv \mathbf{n_n}$

or $g_a \equiv \mathbf{n}$ and $g_b \equiv \mathbf{n}$

or $g_a \equiv \mathbf{w}(a',a'')$ and $g_b \equiv \mathbf{w}(b',b'')$ and the judgements

$a'=b' \quad \mathbf{U}$ and $a''(x)=b''(x) \quad \mathbf{U}[x{:}<a'>]$ are computable

<u>Case 2</u>: There are assumptions, i.e. $C \equiv x_1{:}A_1,\ldots, x_n{:}A_n$, n>0. The judgement J is computable if for any computable closed substitution (c.c.s.) $a_1{:}\underline{A}_1,\ldots,a_n{:}\underline{A}_n$ (i.e. $a_i{:}\underline{A}_i$, $1 \le i \le n$, are computable judgements), and for any computable closed substitution (c.c.s.) $a_1=c_1{:}\underline{A}_1,\ldots,a_n=c_n{:}\underline{A}_n$ (i.e. $a_i=c_i{:}\underline{A}_i$, $1 \le i \le n$, are computable judgements) that fit with C:

<u>Subcase 2.1:</u> $F \equiv B(x_1,\ldots, x_n)$ **type**

2.1.1) (*substitution* :=) the judgement $B(a_1,\ldots, a_n)$ **type** is computable

2.1.2) (*substitution* <−) the judgement $B(a_1,\ldots, a_n)=B(c_1,\ldots, c_n)$ is computable

<u>Subcase 2.2</u>: $F \equiv B(x_1,\ldots, x_n)=D(x_1,\ldots, x_n)$ then

2.2.1) (*associate*) the judgement $B(x_1,\ldots, x_n)$ **type** $[C]$ is computable

2.2.2) (*substitution* :=) the judgement $B(a_1,\ldots, a_n)=D(a_1,\ldots, a_n)$ is computable

2.2.3) (*substitution* <−) the judgement $B(a_1,\ldots, a_n)=D(c_1,\ldots, c_n)$ is computable

<u>Subcase 2.3</u>: $F \equiv b(x_1,\ldots, x_n) \quad B(x_1,\ldots, x_n)$ then

2.3.1) (*associate*) the judgement $B(x_1,\ldots, x_n)$ **type** $[C]$ is computable

2.3.2) (*substitution* :=) the judgement $b(a_1,\ldots, a_n) \quad B(a_1,\ldots, a_n)$ is computable

2.3.3) (*substitution* <−) the judgement $b(a_1,\ldots,a_n)=b(c_1,\ldots,c_n) \quad B(a_1,\ldots,a_n)$ is computable

<u>Subcase 2.4:</u> $F \equiv b(x_1,\ldots, x_n)=d(x_1,\ldots, x_n) \quad B(x_1,\ldots, x_n)$ then

2.4.1) (*associate*) the judgement $b(x_1,\ldots, x_n) \quad B(x_1,\ldots, x_n)[C]$ is computable

2.4.2) (*substitution* :=) the judgement $b(a_1,\ldots, a_n)=d(a_1,\ldots, a_n) \quad B(a_1,\ldots, a_n)$ is computable

2.4.3) (*substitution* <−) the judgement $b(a_1,\ldots, a_n)=d(c_1,\ldots, c_n) \quad B(a_1,\ldots, a_n)$ is computable

Note that the asymmetry in the conditions on associate judgements (point 2.2.1 and 2.4.1) reflects the asymmetry in the rules of the theory. Actually we will prove that also the other associate judgement is computable but the reduced requirement simplifies the following inductive proofs.

By looking at the above definition as a "generalized process" to search for computability of a judgement, a search tree is naturally associate to any derivable judgement. It is clear that whenever *J* is recognized to be a computable judgement its search tree is well founded. In such a case we give the definition of *computation tree* .

Definition 4.2: (Computation tree)

The computation tree of the computable judgement *J* is a tree whose root is *J* and whose principal sub-trees are the computation trees of all the judgements whose computability is asked to prove that *J* is computable.

For instance, the computable judgement λ(**s**) Π(**N**,(*x*)**N**) has the following computation tree:

N type **0** **N**

N type **N type** **N type** **0** **N** **N type** **s**(**0**) **N**

N type **N type**[*x*:**N**] **N type**[*x*:**N**] **s**(**0**) **N** **s**(**s**(**0**)) **N** … … …

Π(**N**,(*x*)**N**) **type** **s**(*x*) **N**[*x*:**N**]

λ(**s**) Π(**N**,(*x*)**N**)

In general the computation tree of a judgement *J* is an infinitary tree: a node has a finite number of branches when we deal with closed judgements, and this number is related to the *parts*, and a possibly infinite one when we deal with hypothetical judgements.

Note that if we know that a judgement is computable the use of induction on the complexity of its well founded computation tree is a correct proof-method.

Definition 4.3: (Computational complexity)

Let *J* be a computable judgement. We will call computational complexity of *J* the ordinal which measures the complexity of its computation tree T, in the following way:

0	if T is a leaf.
$_{i \quad I}(\alpha_i + 1)$	if T has principal sub-trees T_i of computational complexity α_i, (i I) .

We will use both the notation "comp(*J*)=β" and the notation "*J* comp β" to mean that *J* is a computable judgement of computational complexity β.

4.2 The lemmas.

We are now going to prove that any judgement derivable in the theory is computable. The proof will consist in proving that any rule preserves computability, that is, if the judgements in the premisses of a rule are computable then also the judgement in the conclusion of the rule is computable. Of course, this is the inductive step in a proof by induction on the depth of the derivation of the considered judgement.

Note that the computability of the judgements in the base cases is given by definition. Generally, the inductive step for a particular rule will be carried on by subordinate induction on the computational complexity of one of the judgements appearing in the premisses of the rule, usually the first one which has no assumption discharged.

We will consider only "full-context" derivations, i.e. derivations build up by applying a rule only if the assumptions which are not discharged by the rule are equal in all the premises, with the only exception of the assumption rules. Note that this is not restrictive since every derivable judgement can be derived by a full-context derivation.

Before starting this analysis of the rules we state some results which follow rather directly from the definition of computable judgement and which are useful in simplifying the subsequent lemmas.

<u>Fact 4.4</u>: ($\mathbf{N_0}$ is empty)

The closed judgement $c \quad \mathbf{N_0}$ is not computable.

It is stated by the definition of computable judgement since there are no canonical elements in $\mathbf{N_0}$.

<u>Fact 4.4 (a)</u>:

Every hypothetical judgement with open assumption x: $\mathbf{N_0}$ is computable.

<u>Fact 4.5</u>: (Evaluation-free)

1. If A **type** <u>comp</u> β and A»G_A then G_A **type** <u>comp</u> β.
2. If A=C <u>comp</u> β and A»G_A and C»G_C then G_A=G_C <u>comp</u> β.
3. If $a \quad A$ <u>comp</u> β and $a \quad g_a$ and A»G_A then $g_a \quad G_A$ <u>comp</u> β.
4. If a=$b \quad A$ <u>comp</u> β and $a \quad g_a$, $b \quad g_b$ and A»G_A then g_a=$g_b \quad G_A$ <u>comp</u> β.

We will conclude this subsection with the analysis of the simplest rules; we establish also some direct consequences of the definition of computable judgement.

<u>Lemma 4.6</u>: (Weakening rules)

If $F[C]$ is computable then, for any context C' extending C, $F[C']$ is computable.

Proof: When considering associate judgements, if any, the claim follows by induction on the computational complexity of $F[C]$. When considering substitutions just observe that any c.c.s. that fits with C' fits also with C, with redundancies, and we yet know that the resulting substituted judgement is computable.

The next lemma on the reflexivity rule states not only that the rule preserves computability but gives us also a relation between the computational complexities of the judgements in the premise and in the conclusion of the rule. This kind of information, on the dependencies among the computational complexities of computable judgements, has a crucial role in the successive proofs when we proceed by induction on the computational complexity of a judgement. The dependencies are often easy to determine simply by looking at the computation tree of one of the considered judgements.

Lemma 4.7: (Reflexivity_on_elements)

The reflexivity_on_element rule preserves computability, i.e. if *a* *A*[*C*] comp α then *a*=*a* *A*[*C*] comp $\alpha'=\alpha+1$.

Proof: By induction on the computational complexity of the computable judgement *a* *A*[*C*].

Subcase *C* = .

The associate judgements are computable by hypothesis. To prove the computability of the parts we should analyze each possible form of the values of *a* and *A*. Let us consider only the case *a* λ (*b*) and *A*»Π(A$_1$, *A*$_2$). Fig.1a illustrates a piece of the computation tree for this case.

There is only one part judgement *b*(*x*)=*b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$]. Its computability follows, by inductive hypothesis, from the computability of the judgement *b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$]. Thus, *a*=*a* *A* is computable.

It remains to prove the stated relations on complexities.

Let α, α',β,β' be the computational complexities of the computable judgements *a* *A*, *a*=*a* *A*, *b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$], *b*(*x*)=*b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$]. The computability of *a* *A* depends on the computability of *b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$], then we have $\alpha\geq\beta+1$. By applying the inductive hypothesis we have $\beta'=\beta+1$, and hence $\alpha'=\ \ (\alpha+1, \beta'+1)=\alpha+1$.

For all the other cases the proof is analogous.

A **type** *b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$] -β

a *A* -$\alpha\geq\beta+1$ *a* *A* -α *b*(*x*)=*b*(*x*) *A*$_2$(*x*)[*x*:*A*$_1$]-$\beta'=\beta+1\leq\alpha$

a=*a* *A* - $\alpha'=\ \ (\alpha+1,\beta'+1)=\alpha+1$

Fig. 1a

Subcase *C* ≠ .

The computability of the associate judgement of *a*=*a* *A*[*C*] is given by hypothesis, while that of its substituted judgements directly follows by inductive hypothesis.

Moreover, if comp(*a*[x:=e]=*a*[x:=e] *A*[x:=e])=β and comp(*a*[x:=e] *A*[x:=e])=α_i then, by inductive hypothesis, $\beta = \alpha_i+1$, hence $\beta +1\leq \alpha +1$ since $\alpha \geq \alpha_i+1$, and $\alpha'=\ \ (\alpha+1, \beta +1) = \alpha+1$. See the Fig. 1b.

A **type**$[C]$ $\quad a[\underline{x}:=\underline{e}] \quad A[\underline{x}:=\underline{e}]$ -α_i $\quad a[\underline{x}:=\underline{e}]=a[\underline{x}:=\underline{f}] \quad A[\underline{x}:=\underline{e}]$-$\alpha_k$

$a \quad A\,[C]$-α

$a \quad A[C]$ -α $\quad a[\underline{x}:=\underline{e}]=a[\underline{x}:=\underline{e}] \quad A[\underline{x}:=\underline{e}]$ -$\beta=\alpha_i+1\leq\alpha$ $\quad a[\underline{x}:=\underline{e}]=a[\underline{x}:=\underline{f}] \quad A[\underline{x}:=\underline{e}]$ -$\alpha_k+1\leq\alpha$

$a=a \quad A[C]$ -$\alpha'=\alpha+1$

Fig.1b

Lemma 4.8: (Reflexivity_on_type)

The reflexivity_on_type rule preserves computability, i.e. if A **type**$[C]$ comp α then $A=A[C]$ comp $\alpha'=\alpha+1$.

Proof: The proof, by induction on the computational complexity of the computable judgement A **type**$[C]$, is analogous to the one of the previous lemma 4.7 except when the value of A is **I**(A_1,a,b) where the use of the reflexivity_on _elements lemma is needed.

The next corollary is a purely technical result we will use in the following lemmas to abbreviate the proofs.

Corollary 4.9:

i) Let $a \quad A$ and $c \quad A$ be computable closed judgements and g be the value both of a and c.
If $a=c \quad A$ is derivable then it is also computable.

ii) Let A **type** and C **type** be computable closed judgements and G be the value both of A and C.
If $A=C$ is derivable then it is also computable.

iii) Let $a \quad A$ be a computable closed judgement and g the value of a, then $a=g \quad A$ is computable.

iv) Let A **type** be a computable closed judgement and G the value of A, then $A=G$ is computable.

Proof:

i) The associate judgements are computable by hypothesis, then we must only prove that the parts are computable. Since $a \quad A$ is computable and $a \quad g$, if A»G then that the judgement $g \quad G$ is computable, by fact 4.5 (point 3). Hence, by the reflexivity-on-element lemma, also the judgement $g=g \quad G$ is computable and hence the parts of g and g are equal computable elements in G.

ii) The proof is analogous to point i) except for the use of point 1 of fact 4.5, instead of point 3, and the use of the reflexivity-on-type lemma, instead of the reflexivity-on-element lemma.

iii) The proof follows by point i) if we prove that $g \quad A$ is computable. By *correct evaluation*, the judgement $a=g \quad A$ is derivable and hence also its associate judgement $g \quad A$ is derivable. Its

computability then follows by reflexivity and the fact that the parts of $g \quad A$ are exactly those of $a \quad A$ which is computable by hypothesis.
iv) The proof is analogous to point iii) except for the use of point ii) instead of i).

The following lemma 4.10 does not concern one of the rules of the theory but states some properties of computable judgements which will be very often referred to in the following subsections.

<u>Lemma 4.10</u>: (Head substitution)

Let $C \equiv [x_1:A_1,\ldots, x_n:A_n]$ be a context, $J \equiv F[C]$ be a computable judgement, $a_1:\underline{A}_1,\ldots,a_i:\underline{A}_i$ and $a_1=a'_1:\underline{A}_1,\ldots, a_i=a'_i:\underline{A}_i$ (i<n)[4] be c.c.s that fit with the context $[x_1:A_1,\ldots, x_i:A_i]$. Then

i) $J[x_1:=a_1,\ldots, x_i:=a_i]$ is a computable judgement.

ii) $J[x_1{<}{-}a_1=a'_1,\ldots, x_i{<}{-}a_i=a'_i]$ is a computable judgement.

Proof: The proof is by induction on the computational complexity of J.
Let $D \equiv [x_{i+1}:A'_{i+1},\ldots, x_n:A'_n]$ where $A'_j \equiv ((x_1,\ldots, x_i)\ A_j)(a_1,\ldots, a_i)$, $i+1 \leq j \leq n$, and let $a_{i+1}:\underline{A'}_{i+1},\ldots, a_n:\underline{A'}_n$ be a c.c.s. that fits with the context D. To prove the computability of the head substituted judgements we will show that for any c.c.s. saturating $J[x_1:=a_1,\ldots,x_i:=a_i]$ or $J[x_1{<}{-}a_1=a'_1,\ldots,x_i{<}{-}a_i=a'_i]$ it is possible to find out a c.c.s. saturating J and yielding the same judgement. First of all note that for $i+1 \leq j \leq n$,

$$\begin{aligned} \underline{A}_j &\equiv ((x_1,\ldots, x_{j-1})\ A_j)(a_1,\ldots, a_{j-1}) \\ &\equiv ((x_{i+1},\ldots, x_{j-1})(((x_1,\ldots, x_i)\ A_j)(a_1,\ldots, a_i))(a_{i+1},\ldots, a_{j-1}) \\ &\equiv \underline{A'}_j. \end{aligned}$$

case i)
(*associate judgements*) The computability of the associate judgements, if any, follows by inductive hypothesis.
(*substitution* :=) For any c.c.s. $a_{i+1}:\underline{A'}_{i+1},\ldots, a_n:\underline{A'}_n$ we have that $a_1:\underline{A}_1,\ldots, a_n:\underline{A}_n$ is a c.c.s. that fits with C; hence $(J[x_1:=a_1,\ldots,x_i:=a_i])[x_{i+1}:=a_{i+1},\ldots,x_n:=a_n] \equiv J[x_1:=a_1,\ldots,x_n:=a_n]$ is computable.
(*substitution* <−) For any c.c.s. $a_{i+1}=a'_{i+1}:\underline{A'}_{i+1},\ldots, a_n=a'_n:\underline{A'}_n$ that fits with the context D, we have that $a_1=a_1:\underline{A}_1,\ldots, a_i=a_i:\underline{A}_1, a_{i+1}=a'_{i+1}:\underline{A}_{i+1},\ldots, a_n=a'_n:\underline{A}_n$ is a c.c.s. that fits with C. (Note that the reflexivity_on_elements lemma is used).
Hence

$$(J[x_1:=a_1,\ldots,x_i:=a_i])[x_{i+1}{<}{-}a_{i+1}=a'_{i+1},\ldots,x_n{<}{-}a_n=a'_n]$$
$$\equiv J[x_1{<}{-}a_1=a_1,\ldots,x_i{<}{-}a_i=a_i,x_{i+1}{<}{-}a_{i+1}=a'_{i+1},\ldots, x_n{<}{-}a_n=a'_n]$$

is computable.
case ii)
(*associate judgements*) The computability of the associate judgements follows from case i) since if $a_1=a'_1:\underline{A}_1,\ldots,a_i=a'_i:\underline{A}_i$ (i<n) are computable then also $a_1:\underline{A}_1,\ldots,a_i:\underline{A}_i$ are computable and $J[x_1:=a_1,\ldots,x_i:=a_i]$ is the associate judgement of $J[x_1{<}{-}a_1=a'_1,\ldots, x_i{<}{-}a_i=a'_i]$ whose computability is required.

[4] Note that for i=n the claim is true by definition of computable judgement.

(*substitution* :=) For any c.c.s. $a_{i+1}:\underline{A'}_{i+1},\ldots,a_n:\underline{A'}_n$ that fits with the context D, we have that $a_1=a'_1:\underline{A}_1,\ldots,a_i=a'_i:\underline{A}_1, a_{i+1}=a_{i+1}:\underline{A}_{i+1},\ldots,a_n=a_n:\underline{A}_n$ is a c.c.s. that fits with C; hence also

$(J[x_1<-a_1=a'_1,\ldots, x_i<-a_i=a'_i])[x_{i+1}:=a_{i+1},\ldots,x_n:=a_n]$

$\equiv J[x_1<-a_1=a'_1,\ldots,x_i<-a_i=a'_i, x_{i+1}<-a_{i+1}=a_{i+1},\ldots,x_n<-a_n=a_n]$

is computable.

(*substitution* <−) For any c.c.s. $a_{i+1}=a'_{i+1}:\underline{A'}_{i+1},\ldots,a_n=a'_n:\underline{A'}_n$ that fits with the context D, we have that $a_1=a'_1:\underline{A}_1,\ldots, a_i=a'_i:\underline{A}_1, a_{i+1}=a'_{i+1}:\underline{A}_{i+1},\ldots, a_n=a'_n:\underline{A}_n$ is a c.c.s. that fits with C; hence

$(J[x_1<-a_1=a'_1,\ldots, x_i<-a_i=a'_i])[x_{i+1}<-a_{i+1}=a'_{i+1},\ldots, x_n<-a_n=a'_n]$

$\equiv J[x_1<-a_1=a'_1,\ldots, x_i<-a_i=a'_i, x_{i+1}<-a_{i+1}=a'_{i+1},\ldots, x_n<-a_n=a'_n]$

is computable.

Remark 4.11:

Let $C \equiv [x_1:A_1,\ldots, x_n:A_n]$ be a context, $a_1:\underline{A}_1,\ldots,a_n:\underline{A}_n$ and $a_1=a'_1:\underline{A}_1,\ldots, a_n=a'_n:\underline{A}_n$ be c.c.s.s that fit with C, and $B\equiv (s_1:S_1,\ldots, s_m:S_m)A(s_1,\ldots, s_m)$, then from the head substitution lemma we have that

if $:B[C]$ is computable then	$:B[C]\ [x_1:=a_1,\ldots, x_n:=a_n]$ and
	$:B[C]\ [x_1<-a_1=a'_1,\ldots, x_n<-a_n=a'_n]$ are computable;
if $b:B[C]$ is computable then	$b:B[C]\ [x_1:=a_1,\ldots, x_n:=a_n]$ and
	$b:B[C]\ [x_1<-a_1=a'_1,\ldots, x_n<-a_n=a'_n]$ are computable;
if $b=b':B[C]$ is computable then	$b=b':B[C]\ [x_1:=a_1,\ldots, x_n:=a_n]$ and
	$b=b':B[C]\ [x_1<-a_1=a'_1,\ldots,x_n<-a_n=a'_n]$ are computable.

We continue by proving that each rule listed in the appendix *preserves* computability, i.e. any judgement in the conclusion of that rule is computable whenever all the judgements in the premises are computable. The ordering of the lemmas has been suitably chosen to allow us to deal separately with each rule thus mastering the complexity of the computability proof.

4.2.1. The substitution rules

The definition of computable judgement directly states that the substitution rules preserve computability in the special case of saturating substitutions. In the next lemma we will prove that computability is preserved by substitution rules also in the general case of tail substitution.

Since the different forms of judgement of the six substitution rules are not essential to prove the result we will compact the sentence as much as possible.

Lemma 4.12: (Substitution lemma)

Let $D \equiv[C, x_1:A_1,\ldots,x_n:A_n]$, where C is a context, $J\equiv F[D]$ be a computable judgement, $a_1:B_1[C],\ldots, a_n:B_n[C]$ and $a_1=a'_1:B_1[C],\ldots, a_n=a'_n:B_n[C]$ be substitutions that fit with the context $[x_1:A_1,\ldots,x_n:A_n]$, (i.e. $B_j \equiv ((x_1,\ldots,x_{j-1})\ A_j)\ (a_1,\ldots, a_{j-1})$, $1\leq j\leq n$). Then

i) $J[x_1:=a_1,\ldots, x_n:=a_n]$ is a computable judgement.

ii) $J[x_1<-a_1=a'_1,\ldots, x_n<-a_n=a'_n]$ is a computable judgement.

Proof: If $C =$ then the claim holds by definition. Let $C \equiv[s_1:S_1,\ldots,s_m:S_m]$, m>0. The proof is by induction on the computational complexity of J.

case i)

(*associate judgements*) The computability of the associate judgements follows by inductive hypothesis.

(*substitution* :=) For any c.c.s. $c_1{:}S_1,\ldots,c_m{:}S_m$ fitting with the context C, we define

$d_i \equiv ((s_1,\ldots,s_m)\ a_i)(c_1,\ldots,c_m)$, $1 \le i \le n$.

By remark 4.11

$d_i : ((s_1,\ldots,s_m)\ B_i)(c_1,\ldots,c_m) \equiv a_i{:}B_i[C]\ [s_1{:=}c_1,\ldots,s_m{:=}c_m]$

is computable.

Moreover we have

$$\begin{aligned}((s_1,\ldots,s_m)\ B_i)(c_1,\ldots,c_m) &\equiv ((s_1,\ldots,s_m)((x_1,\ldots,x_{i-1})\ A_i)(a_1,\ldots,a_{i-1}))(c_1,\ldots,c_m)\\ &\equiv ((s_1,\ldots,s_m,x_1,\ldots,x_{i-1})\ A_i)(c_1,\ldots,c_m,d_1,\ldots,d_{i-1})\\ &\equiv \underline{\mathsf{A}}_i\end{aligned}$$

Therefore $c_1{:}\underline{S}_1,\ldots,c_m{:}\underline{S}_m, d_1{:}\underline{A}_1,\ldots,d_n{:}\underline{A}_n$ is a c.c.s. fitting with the context D. Hence

$(J[x_1{:=}a_1,\ldots,x_n{:=}a_n])[s_1{:=}c_1,\ldots,s_m{:=}c_m] \equiv J[s_1{:=}c_1,\ldots,s_m{:=}c_m,x_1{:=}d_1,\ldots,x_n{:=}d_n]$

is computable.

(*substitution* $\leftarrow$) For any c.c.s. $c_1{=}c'_1{:}\underline{S}_1,\ldots,c_m{=}c'_m{:}\underline{S}_m$ that fits with the context C, we define

$d_i \equiv ((s_1,\ldots,s_m)\ a_i)(c_1,\ldots,c_m)$

and

$d'_i \equiv ((s_1,\ldots,s_m)\ a_i)(c'_1,\ldots,c'_m)$, $1 \le i \le n$.

By remark 4.11

$d_i{=}d'_i : ((s_1,\ldots,s_m)\ B_i)(c_1,\ldots,c_m) \equiv a_i{:}B_i\ [C]\ [s_1{<-}c_1{=}c'_1,\ldots,s_m{<-}c_m{=}c'_m]$

is computable.

Moreover we have, see previous point, $((s_1,\ldots,s_m)\ B_i)(c_1,\ldots,c_m) \equiv \underline{\mathsf{A}}_i$. Then, $c_1{=}c'_1{:}\underline{S}_1,\ldots,c_m{=}c'_m{:}\underline{S}_m$, $d_1{=}d'_1{:}\underline{A}_1,\ldots,d_n{=}d'_n{:}\underline{A}_n$ is a c.c.s. that fits with the context D. Hence

$(J[x_1{:=}a_1,\ldots,x_n{:=}a_n])[s_1{<-}c_1{=}c'_1,\ldots,s_m{<-}c_m{=}c'_m]$

$\equiv J[s_1{<-}c_1{=}c'_1,\ldots,s_m{<-}c_m{=}c'_m,x_1{<-}d_1{=}d'_1,\ldots,x_n{<-}d_n{=}d'_n]$

is computable.

case ii)

(*associate judgements*) The computability of the associate judgements follows from case (i).

(*substitution* :=) For any c.c.s. $c_1{:}\underline{S}_1,\ldots,c_m{:}\underline{S}_m$ fitting with the context C , we define

$d_i \equiv ((s_1,\ldots,s_m)\ a_i)(c_1,\ldots,c_m)$

and

$d'_i \equiv ((s_1,\ldots,s_m)\ a'_i)(c_1,\ldots,c_m)$.

By remark 4.11

$d_i{=}d'_i : ((s_1,\ldots,s_m)\ B_i)(c_1,\ldots,c_m) \equiv a_i{=}a'_i{:}B_i\ [C]\ [s_1{:=}c_1,\ldots,s_m{:=}c_m]$

is computable.

Then, since $((s_1,\ldots,s_m)\ B_i)(c_1,\ldots,c_m) \equiv \underline{\mathsf{A}}_i$, $c_1{=}c_1{:}\underline{S}_1,\ldots,c_m{=}c_m{:}\underline{S}_m, d_1{=}d'_1{:}\underline{A}_1,\ldots,d_n{=}d'_n{:}\underline{A}_n$ is a c.c.s. fitting with the context D. Hence

$(J[x_1{<-}a_1{=}a'_1,\ldots,x_n{<-}a_n{=}a'_n])[s_1{:=}c_1,\ldots,s_m{:=}c_m]$

$\equiv J[s_1{<-}c_1{=}c_1,\ldots,s_m{<-}c_m{=}c_m,x_1{<-}d_1{=}d'_1,\ldots,x_n{<-}d_n{=}d'_n]$

is computable.
(*substitution* <−) For any c.c.s. $c_1=c'_1{:}\underline{S}_1,\ldots, c_m=c'_m{:}\underline{S}_m$ fitting with the context C , we define

$$d_i \equiv ((s_1,\ldots, s_m)\ a_i)(c_1,\ldots, c_m)$$

and

$$d'_i \equiv ((s_1,\ldots, s_m)\ a_i)(c'_1,\ldots, c'_m).$$

By remark 4.11

$$d_i=d'_i : ((s_1,\ldots, s_m)\ B_i)(c_1,\ldots, c_m) \equiv a_i=a'_i{:}B_i\ [C]\ [s_1{<-}c_1=c'_1,\ldots, s_m{<-}c_m=c'_m]$$

is computable.
Then, since $((s_1,\ldots, s_m)\ B_i)(c_1,\ldots, c_m) \equiv \underline{A}_i$, $c_1=c'_1{:}\underline{S}_1,\ldots,c_m=c'_m{:}\underline{S}_m, d_1=d'_1{:}\underline{A}_1,\ldots,d_n=d'_n{:}\underline{A}_n$ is a c.c.s. fitting with the context D. Hence

$$(J[x_1{<-}a_1=a'_1,\ldots, x_n{<-}a_n=a'_n]\ )[s_1{<-}c_1=c'_1,\ldots, s_m{<-}c_m=c'_m]$$
$$\equiv J[s_1{<-}c_1=c'_1,\ldots, s_m{<-}c_m=c'_m, x_1{<-}d_1=d'_1,\ldots, x_n{<-}d_n=d'_n]$$

is computable.

4.2.2. U-elimination rules

Lemma 4.13 deals with **U**-elimination rules. We need to know that they preserve both computability and computational complexity to establish the next lemma 4.14 about computability of the remaining structural rules. For this reason their analysis precede so much that of all the other logical rules.

<u>Lemma 4.13</u>: (**U**-elimination)
The **U**-elimination rules preserve computability and do not increase computational complexity, i.e.

1) If a **U**[C] <u>comp</u> β then <a> **type**[C] <u>comp</u> β'≤β
2) If $a=b$ **U**[C] <u>comp</u> β then <a> = <b>[C] <u>comp</u> β'≤β

Proof: by induction on the computational complexity β.
<u>Case 1</u>:
Subcase ***C*** =   .
(*evaluation*) $<a>\gg G_{<a>}$ immediately follows from the computability of the judgement a **U** by using the suitable computation rule.
(*correct evaluation*) the required derivation can be obtained by applying the suitable **U**-equality rule to premises whose existence is guarantied by the computability of the judgement a **U**. For instance, if a π (a',a''):

$$\dfrac{\dfrac{a=\pi(a',a'')\ \ \mathbf{U}}{<a>=<\pi(a',a'')>} \qquad \dfrac{a'\ \ \mathbf{U} \qquad a''(x)\ \ \mathbf{U}[x{:}<a'>]}{<\pi(a',a'')>=\prod(<a'>,(x)<a''(x)>)}}{<a> = \prod(<a'>,(x)<a''(x)>)}$$

(*parts*) the parts of $G_{<a>}$ are computable type(s) with computational complexity less or equal of that of the corresponding parts of the computable judgement a **U**. For instance if $G_{<a>} \equiv \prod(<a'>,(x)<a''(x)>)$, by ind. hyp., we obtain that a' **U** <u>comp</u> β' implies <a'> **type** <u>comp</u> γ'≤β' and $a''(x)$ **U**[x:<a'>] <u>comp</u> β'' implies <$a''(x)$> **type**[x:<a'>] <u>comp</u> γ''≤β''. The other cases are completely similar.
Subcase ***C*** ≠   .

(*substitution* :=) immediately follows by ind. hyp. (1)
(*substitution* <−) immediately follows by ind. hyp. (2)
Case 2.
Subcase ***C*** = .
(*associate*) from $a=b$ **U** comp β we obtain a **U** comp $\beta_1<\beta$ and b **U** comp $\beta_2<\beta$. Hence by ind. hyp., point 1, $<a>$ **type** comp $\beta_1'\leq\beta_1$ and $<b>$ **type** comp $\beta_2'\leq\beta_2$.
(*parts*) if $<a>\,»\,G_{<a>}$ and $<b>\,»\,G_{<b>}$ then the parts of $G_{<a>}$ and $G_{<b>}$ are equal computable types with the same computational complexity of the corresponding parts of the computable judgement $a=b$ **U**. Let us analyze the case $G_{<a>} \equiv \prod(<a'>,(x)<a''(x)>)$:

$G_{<a>} \equiv \prod(<a'>,(x)<a''(x)>)$ iff a $\pi(a',a'')$
iff b $\pi(b',b'')$
iff $G_{<b>} \equiv \prod(<b'>,(x)<b''(x)>)$

and, by ind. hyp., we obtain that $a'=b'$ **U** comp β_3 implies $<a'>=<b'>$ comp $\beta_3'\leq\beta_3$ and $a''(x)=b''(x)$ **U**$[x{:}<a'>]$ comp β_4 implies $<a''(x)>=<b''(x)>$ $[x{:}<a'>]$ comp $\beta_4'\leq\beta$.
Hence $\beta'=$ $(\beta_1'+1, \beta_2'+1, \beta_3'+1, \beta_4'+1)\leq$ $(\beta_1+1, \beta_2+1, \beta_3+1, \beta_4+1)=\beta$.
The other cases are completely similar.
Subcase ***C*** ≠ .

(*associate*)	immediately follows by ind. hyp., point 1.
(*substitution* :=)	immediately follows by ind. hyp., point 2.
(*substitution* <−)	immediately follows by ind. hyp., point 2.

Note that the computational complexity of the judgements in the premise and in the conclusion are usually equal, but we may also have different complexities. For instance, the complexity of a basic judgement, like **N type**, is 0 while the complexity of the corresponding judgement, **<n>** **U**, is 1 due to the requirement that the associate judgement **U type** is computable.

4.2.3. The structural rules

All the other structural rules, i.e. the transitivity-on-element, the transitivity-on-type, the symmetry-on-element, the symmetry-on-type, the equal element and the equal type rules, are considered in the next lemma 4.14 . This lemma is a key point in the proof of computability since it establishes, besides the fact that structural rules preserve computability, other basic and important relationships among the computational complexities of related judgements. As we already pointed out, these information are essential in the subsequent proof since they guarantee the applicability of the inductive hypothesis when we proceed by induction on the computational complexity of a judgement.

Lemma 4.14: (Structural rules)
Let β be a computational complexity.
1. If $a=c$ $A[C]$ comp $\alpha_1<\beta$, $b=d$ $A[C]$ comp $\alpha_2<\beta$, $a=b$ $A[C]$ comp α
   then (i) $\alpha_1=\alpha_2=\alpha$ and (ii) $c=d$ $A[C]$ comp α
   1.1 (transitivity on elements)
   If $a=b$ $A[C]$ comp $\alpha_1<\beta$, $b=c$ $A[C]$ comp $\alpha_2<\beta$ then $a=c$ $A[C]$ comp $\alpha=\alpha_1=\alpha_2$

2. If $A=C[C]$ $\underline{\text{comp}}\ \alpha_1 < \beta$, $B=D[C]$ $\underline{\text{comp}}\ \alpha_2 < \beta$, $A=B[C]$ $\underline{\text{comp}}\ \alpha$
then (i) $\alpha_1 = \alpha_2 = \alpha$ and (ii) $C=D[C]$ $\underline{\text{comp}}\ \alpha$

2.1 (transitivity on types)
If $A=B[C]$ $\underline{\text{comp}}\ \alpha_1 < \beta$, $B=C[C]$ $\underline{\text{comp}}\ \alpha_2 < \beta$ then $A=C[C]$ $\underline{\text{comp}}\ \alpha=\alpha_1=\alpha_2$

3. If $A=C[C]$ $\underline{\text{comp}}\ \beta$ then

i. (associate judgements)
A **type**$[C]$ $\underline{\text{comp}}\ \alpha$ iff C **type**$[C]$ $\underline{\text{comp}}\ \alpha$

ii. (symmetry on types)
$C=A\ [C]$ $\underline{\text{comp}}\ \beta$

iii. (element in equal types and equal elements in equal types)
iii.a $a\ \ A[C]$ $\underline{\text{comp}}\ \alpha$ iff $a\ \ C[C]$ $\underline{\text{comp}}\ \alpha$
iii.b $a=c\ \ A[C]$ $\underline{\text{comp}}\ \alpha$ iff $a=c\ \ C[C]$ $\underline{\text{comp}}\ \alpha$

iv. (assumption in equal types)
$J[C, x{:}A]$ $\underline{\text{comp}}\ \alpha$ iff $J[C, x{:}C]$ $\underline{\text{comp}}\ \alpha$

4. If $a=c\ \ A[C]$ $\underline{\text{comp}}\ \beta$ then

i. (associate judgements)
$a\ \ A[C]$ $\underline{\text{comp}}\ \alpha$ iff $c\ \ A[C]$ $\underline{\text{comp}}\ \alpha$

ii. (symmetry on elements)
$c=a\ \ A[C]$ $\underline{\text{comp}}\ \beta$

Proof: By principal induction on β. The base cases are obvious.

Point 1. The proof follows by subordinate induction on α. As regards Point 1.ii, a derivation for the judgement $c=d\ \ A[C]$ can easily be found by using symmetry and transitivity rules and the derivations for $a=c\ \ A[C]$, $b=d\ \ A[C]$, $a=b\ \ A[C]$.

Point 1. Subcase $C=\emptyset$

Let G_A be the canonical value of A. The proof varies according to the outermost constant of G_A, but there is a common pattern. First we prove that the three considered judgements have corresponding part judgements with the same computational complexity. Then, a similar result follows also for the corresponding associate judgements. Here we analyse only three main cases; the other cases are similar.

• $G_A \equiv \prod(A_1, A_2)$ and $a\ \ \lambda(a')$, $b\ \ \lambda(b')$, $c\ \ \lambda(c')$, $d\ \ \lambda(d')$

A **type** $\quad a'(x)\quad A_2(x)[x{:}A_1]$

$a\quad A$ -γ_1 $\qquad c\quad A$ -γ'_1 $\qquad a'(x)=c'(x)\quad A_2(x)[x{:}A_1]$ -α_1'

$a=c\quad A$-$\alpha_1<\beta$

A **type** $\quad b'(x)\quad A_2(x)[x{:}A_1]$

$b\quad A$ -γ_2 $\qquad d\quad A$ -γ'_2 $\qquad b'(x)=d'(x)\quad A_2(x)[x{:}A_1]$-$\alpha_2'$

$b=d\quad A$-$\alpha_2<\beta$

$a\quad A$ -γ_1 $\qquad b\quad A$ -γ_2 $\qquad a'(x)=b'(x)\quad A_2(x)[x{:}A_1]$ -$\alpha'<\beta$

$a=b\quad A$-α

Fig. 2. Computation trees (Point 1.i. Π-case)

(Point 1.i Π-case) We know that (see fig. 2)

$a'(x)=c'(x)\quad A_2(x)[x{:}A_1]$ <u>comp</u> $\alpha_1'<\alpha_1<\beta$ and

$b'(x)=d'(x)\quad A_2(x)[x{:}A_1]$ <u>comp</u> $\alpha_2'<\alpha_2<\beta$ and

$a'(x)=b'(x)\quad A_2(x)[x{:}A_1]$ <u>comp</u> $\alpha'<\alpha$

hence, by subordinate ind. hyp., $\alpha_1'=\alpha_2'=\alpha'$. Hence the *parts* have the same computational complexity. Note that $\alpha'<\beta$.

As regards the *associate judgements*, we obtain by ind. hyp. 4.i:

$\gamma_1=$<u>comp</u>$(a\quad A)=$<u>comp</u>$(c\quad A)=\gamma'_1$ from $a=c\quad A$ <u>comp</u>-$\alpha_1<\beta$

$\gamma_2=$<u>comp</u>$(b\quad A)=$<u>comp</u>$(d\quad A)=\gamma'_2$ from $b=d\quad A$ <u>comp</u>-$\alpha_2<\beta$

<u>comp</u>$(a'(x)\quad A_2(x)[x{:}A_1])=$comp$(b'(x)\quad A_2(x)[x{:}A_1])$

from $a'(x)=b'(x)\quad A_2(x)[x{:}A_1]$ <u>comp</u> $\alpha'<\beta$

which guarantees $\gamma_1=$<u>comp</u>$(a\quad A)=$<u>comp</u>$(b\quad A)=\gamma_2$. Hence $\gamma'_1=\gamma_1=\gamma_2=\gamma'_2$.

(Point 1.ii Π-case) We yet proved that <u>comp</u>$(c\quad A)=$<u>comp</u>$(d\quad A)$, hence $c=d\quad A$ <u>comp</u> α follows by applying ind. hyp. (point 1.ii) to the part judgements of the four considered judgements.

- $G_A\equiv\Sigma(A_1,A_2)$ and $a\quad$ **&**(a',a''), $b\quad$ **&**(b',b''), $c\quad$ **&**(c',c''), $d\quad$ **&**(d',d'')

(Point 1.i Σ-case) We know that (see fig. 3)

$a'=c' \quad A_1\ \underline{\text{comp}}\ \alpha_1'<\alpha_1<\beta$ and

$b'=d' \quad A_1\ \underline{\text{comp}}\ \alpha_2'<\alpha_2<\beta$ and

(*) $a'=b' \quad A_1\ \underline{\text{comp}}\ \alpha'<\alpha$

hence, by subordinate ind. hyp., $\alpha_1'=\alpha_2'=\alpha'<\beta$

Moreover

$a''=c'' \quad A_2(a')\ \underline{\text{comp}}\ \alpha_1''<\alpha_1<\beta$ and

$b''=d'' \quad A_2(b')\ \underline{\text{comp}}\ \alpha_2''<\alpha_2<\beta$ and

$a''=b'' \quad A_2(a')\ \underline{\text{comp}}\ \alpha''<\alpha$✎

By using (*), we obtain

$\underline{\text{comp}}(A_2(a')=A_2(b'))<\underline{\text{comp}}(A_2(x)\ \textbf{type}[x{:}A_1])<\text{comp}(A\ \textbf{type})<\text{comp}(a \quad A)<\text{comp}(a=c \quad A)<\beta$

then, by ind. hyp. (point 3.iii.b), we have: $\text{comp}(b''=d'' \quad A_2(a'))=\alpha_2''<\beta$ and hence, by subordinate ind. hyp., $\alpha_1''=\alpha_2''=\alpha''<\beta$.

$\ldots\ldots \quad A_2(a')=A_2(b')\text{-}\delta \qquad \ldots \quad A_2(a')=A_2(c')\text{-}\gamma$

$A_1\ \textbf{type} \qquad A_2(x)\ \textbf{type}[x{:}A_1]$

$A\ \textbf{type} \qquad \ldots$

$a \quad A\ \text{-}\gamma_1 \qquad c \quad A\ \text{-}\gamma'_1 \qquad a'=c' \quad A_1\ \text{-}\alpha_1'<\beta \qquad a''=c'' \quad A_2(a')\ \text{-}\alpha_1''<\beta$

$a=c \quad A\ \text{-}\alpha_1<\beta$

$b \quad A\ \text{-}\gamma_2 \qquad d \quad A\ \text{-}\gamma'_2 \qquad b'=d' \quad A_1\ \text{-}\alpha_1'<\beta \qquad b''=d'' \quad A_2(b')\ \text{-}\alpha_2''<\beta$

$b=d \quad A\ \text{-}\alpha_2<\beta$

$a \quad A\ \text{-}\gamma_1 \qquad b \quad A\ \text{-}\gamma_2 \qquad a'=b' \quad A_1\ \text{-}\alpha'<\beta \qquad a''=b'' \quad A_2(a')\ \text{-}\alpha''<\alpha$

$a=b \quad A\ \text{-}\alpha$

Fig. 3. Computation trees (Point 1.i. Σ-case)

The proof proceeds analogously to the Π−case. Simply note that to prove that the three considered judgements have corresponding associate judgements with the same computational complexity we need the ind. hyp. (point 3.iii.a) to obtain that $\text{comp}(b'' \quad A_2(a')) = \text{comp}(b'' \quad A_2(b'))$.

(Point 1.ii Σ-case) Since $A_2(a')=A_2(b')\ \underline{\text{comp}}\ \delta<\beta$, from $b''=d''\quad A_2(b')\ \underline{\text{comp}}\ \alpha_2''$, by ind. hyp. points 3.ii and 3.iii.b we obtain $b''=d''\quad A_2(a')\ \underline{\text{comp}}\ \alpha_2''$. Then from $a''=c''\quad A_2(a')\ \underline{\text{comp}}\ \alpha_1''$, $b''=d''\quad A_2(a')\ \underline{\text{comp}}\ \alpha_2''$ and $a''=b''\quad A_2(a')\ \underline{\text{comp}}\ \alpha''$, by ind. hyp. point 1.ii, we obtain:

$c''=d''\quad A_2(a')\ \underline{\text{comp}}\ \alpha''$.

From this, since $A_2(a')=A_2(c')\ \underline{\text{comp}}\ \gamma<\beta$, by ind. hyp. point 3.iii.b, we obtain:

$c''=d''\quad A_2(c')\ \underline{\text{comp}}\ \alpha''$.

We can easily prove also that $c\quad A\ \underline{\text{comp}}\ \gamma_1$, $d\quad A\ \underline{\text{comp}}\ \gamma_2$, $c'=d'\quad A\ \underline{\text{comp}}\ \alpha'$, and hence that $c=d\quad A\ \underline{\text{comp}}\ \alpha$.

- $G_A \equiv \mathbf{U}$.

We develop in a detailed way only the case $a\quad \pi(a',a''), b\quad \pi(b',b''), c\quad \pi(c',c''), d\quad \pi(d',d'')$.

(Point 1.i **U**-case) We know that

$a'=c'\quad \mathbf{U}\ \underline{\text{comp}}\ \alpha_1'<\alpha_1<\beta$ and

$b'=d'\quad \mathbf{U}\ \underline{\text{comp}}\ \alpha_2'<\alpha_2<\beta$ and

$a'=b'\quad \mathbf{U}\ \underline{\text{comp}}\ \alpha'<\alpha$

hence, by subordinate ind. hyp., $\alpha_1'=\alpha_2'=\alpha'<\beta$.

Moreover we know that

$a''(x)=c''(x)\quad \mathbf{U}\ [x{:}<a'>]\ \underline{\text{comp}}\ \alpha_1''<\alpha_1<\beta$ and

$b''(x)=d''(x)\quad \mathbf{U}\ [x{:}<b'>]\ \underline{\text{comp}}\ \alpha_2''<\alpha_2<\beta$ and

$a''(x)=b''(x)\quad \mathbf{U}\ [x{:}<a'>]\ \underline{\text{comp}}\ \alpha''<\alpha$.

Since $a'=b'\quad \mathbf{U}\ \underline{\text{comp}}\ \alpha'<\beta$, by lemma 4.13, we obtain $<a'>=<b'>\ \underline{\text{comp}}\ \gamma'\leq\alpha'<\beta$ and then, by ind. hyp. (point 3.iv), $b''(x)=d''(x)\quad \mathbf{U}[x{:}<a'>]\ \underline{\text{comp}}\ \alpha_2''$, hence, by subordinate ind. hyp., $\alpha_1''=\alpha_2''=\alpha''<\beta$.

Analogously to the previous cases, it is now easy to prove that the three considered judgements have corresponding associate judgements with the same computational complexity .

(Point 1.ii **U**-case) The proof proceeds analogously to the previous cases. It is worth to describe only the proof that $c''(x)=d''(x)\quad \mathbf{U}\ [x{:}<c'>]\ \underline{\text{comp}}\ \alpha''$. By subordinate inductive hypothesis we know that $c''(x)=d''(x)\quad \mathbf{U}\ [x{:}<a'>]\ \underline{\text{comp}}\ \alpha''$. Since $a'=c'\quad \mathbf{U}\ \underline{\text{comp}}\ \alpha_1'<\beta$, by lemma 4.13, we obtain $<a'>=<c'>\ \underline{\text{comp}}\ \gamma_1'\leq\alpha_1'<\beta$ and then, by ind. hyp. (point 3.iv), we obtain:

$c''(x)=d''(x)\quad \mathbf{U}\ [x{:}<c'>]\ \underline{\text{comp}}\ \alpha''$.

Point 1. Subcase $C \neq \mathbf{\emptyset}$

(Point 1.i) We prove that the three considered judgements have the corresponding associate judgements and substituted judgements of the same computational complexity.

(*substitution* :=) Immediate by subordinate ind. hyp., (see fig. 4).

… $a[\underline{x}:=\underline{e}]=a[\underline{x}:=\underline{e}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha_1*$

$a\quad A[C]\text{-}\alpha_1'$ $\qquad a[\underline{x}:=\underline{e}]=c[\underline{x}:=\underline{e}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha_1''$ $a[\underline{x}:=\underline{e}]=c[\underline{x}:=\underline{f}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha_1'''$

$a=c\quad A[C]\text{-}\alpha_1<\beta$

… $b[\underline{x}:=\underline{e}]=b[\underline{x}:=\underline{f}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha_2*$

$b\quad A[C]\text{-}\alpha_2'$ $\qquad b[\underline{x}:=\underline{e}]=d[\underline{x}:=\underline{e}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha_2''$ $\qquad b[\underline{x}:=\underline{e}]=d[\underline{x}:=\underline{f}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha_2'''$

$b=d\quad A[C]\text{-}\alpha_2<\beta$

$a\quad A[C]\text{-}\alpha'=\alpha_1'$ $a[\underline{x}:=\underline{e}]=b[\underline{x}:=\underline{e}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha''$ $a[\underline{x}:=\underline{e}]=b[\underline{x}:=\underline{f}]\quad A[\underline{x}:=\underline{e}]\text{-}\alpha'''$

$a=b\quad A[C]\text{-}\alpha<\beta$

Fig. 4. Point 1.i (*substitution* :=)

(*substitution* <−) First observe that, by subordinate ind. hyp. (1.i) $\alpha'' = \alpha_1''' = \alpha_2'''$. Moreover, if $\underline{e}=\underline{f}$: is a c.c.s. fitting with C this holds also for $\underline{e}$:; hence $\alpha_1*<\beta$ and $\alpha_2*<\beta$ and, by subordinate ind. hyp. (points 1.i and 1.ii), $\alpha'' = \alpha_1* = \alpha_2* = \alpha'''$. Hence $\alpha''' = \alpha_1''' = \alpha_2'''$.

(*associate judgements*) Since $\alpha'=\alpha_1'$, $\alpha''=\alpha_1''$ and $\alpha'''=\alpha_1'''$ then $\alpha=\alpha_1<\beta$ and, by ind. hyp. 4.i, comp($b\quad A[C]$)$=\alpha_2'=$comp($a\quad A[C]$)$=\alpha_1'$.

(Point 1.ii) By ind. hyp. (point 4.i) comp($c\quad A[C]$)=comp($a\quad A[C]$)$=\alpha'$, by subordinate ind. hyp. on substituted judgements, for any c.c.s. $\underline{e}$:, comp($c[\underline{x}:=\underline{e}]=d[\underline{x}:=\underline{e}]\quad A[\underline{x}:=\underline{e}]$)$=\alpha''$ holds and for any c.c.s. $\underline{e}=\underline{f}$:, comp($c[\underline{x}:=\underline{e}]=d[\underline{x}:=\underline{f}]\quad A[\underline{x}:=\underline{e}]$)$=\alpha''$ holds hence comp($c=d\quad A[C]$)$=\alpha$.

<u>Point 1.1</u>

By ind. hyp. (point 4.ii) from comp($a=b\quad A[C]$)$=\alpha_1<\beta$ we obtain comp($b=a\quad$ A[C])$=\alpha_1<\beta$. Hence $b\quad A[C]$ is computable and, by the reflexivity lemma, $b=b\quad A[C]$ is computable. Then the result follows from the previous point 1.ii.

<u>Point 2.</u>

The proof follows by subordinate induction on α.

The proof of this case is analogous to the one of point 1. Just substitute the judgement 'equal elements in a type' with the judgement 'equal types' and pay attention in analyzing the case $G_A \equiv \mathbf{I}(A_1,e,f)$ where an obvious application of point 1. is required.

<u>Point 2.1</u>

By ind. hyp. point 3.ii, from comp(A=B[C])=$\alpha_1<\beta$ we obtain comp(B=A[C])=$\alpha_1<\beta$. Hence B **type**[C] is computable and, by the reflexivity lemma, B=B[C] is computable. Then the result follows from the previous point 2.ii.

<u>Point 3.i</u>

Point 3.i Subcase C **=** **Ø**.

Let A»G_A and C»G_C we must prove that the parts of G_A and G_C have the same computational complexity. The proof varies according to the outermost constant of G_A. We analyze only two significant cases; the other cases are similar.

- $G_A \equiv \prod(A_1,A_2)$ hence, since A=C is a computable judgement, $G_C \equiv \prod(C_1,C_2)$ and we know that

(*) comp(A_1=C_1) < β,

hence, by ind. hyp. point 3.i, comp(A_1 **type**)=comp(C_1 **type**);

(**) comp($A_2(x)$=$C_2(x)$ [x**:**A_1]) < β,

hence, by ind. hyp. point 3.i, comp($A_2(x)$ **type** [x**:**A_1]) = comp($C_2(x)$ **type**[x**:**A_1]) and finally

comp($A_2(x)$ **type** [x**:**A_1]) = comp($C_2(x)$ **type** [x:C$_1$]),

by using ind. hyp. point 3.iv.

- $G_A \equiv \mathbf{I}(A_1,a',a'')$ and $G_C \equiv \mathbf{I}(C_1,c',c'')$ and we know that

(*) comp(A_1=C_1) < β,

then, by ind. hyp. point 3.i, comp(A_1 **type**) = comp(C_1 **type**)

(**) comp(a'=c'   A$_1$) < β,

hence, by ind. hyp. point 4.i, comp(a' A_1) = comp(c' A_1) and then, by ind. hyp. point 3.iii.a, comp(a' A_1) = comp(c' C_1);

(***) comp(a''=c'' A_1) < β,

hence, by ind. hyp. point 4.i, comp(a'' A_1) = comp(c'' A_1) and then, by ind. hyp. point 3.iii.a, comp(a'' A_1) = comp(c'' C_1).

Point 3.i Subcase C ≠ **Ø**

For any c.c.s. $\underline{e}$:, since comp($A[\underline{x}:=\underline{e}]$=$C[\underline{x}:=\underline{e}]$)<$\beta$, by ind. hyp. 3.i, it immediately follows: comp($A[\underline{x}:=\underline{e}]$ **type**)=comp($C[\underline{x}:=\underline{e}]$ **type**).

For any c.c.s. $\underline{e}$=$\underline{f}$: we know that $\underline{e}$: and, by reflexivity, $\underline{e}$=$\underline{e}$: are c.c.s fitting with C. Hence,

comp($A[\underline{x}:=\underline{e}]$=$A[\underline{x}:=\underline{e}]$)=comp($A[\underline{x}:=\underline{e}]$=$A[\underline{x}:=\underline{f}]$),

by ind. hyp. 2.i, and

comp($A[\underline{x}:=\underline{e}]$=$A[\underline{x}:=\underline{e}]$)=comp($C[\underline{x}:=\underline{e}]$=$C[\underline{x}:=\underline{f}]$),

by ind. hyp. 2.ii. Hence

comp($A[\underline{x}:=\underline{e}]$=$A[\underline{x}:=\underline{f}]$)=comp($C[\underline{x}:=\underline{e}]$=$C[\underline{x}:=\underline{f}]$)

and thus comp(A **type**[C]) = comp(C **type**[C])

Point 3.ii

(*associate judgements*) We must prove that the associate judgements of the derivable judgement $C=A[C]$ are computable and their computational complexities are equal to the computational complexities of the corresponding associate of the judgement $A=C[C]$. This result is obvious by the previous point 3.i.

Point 3.ii Subcase $C = \emptyset$.

We must prove that the parts of $C=A$ have the same computational complexity of the parts of $A=C$. Let us consider two cases

- $G_A \equiv \prod(A_1, A_2)$ and hence $G_C \equiv \prod(C_1, C_2)$.

The parts of $A=C$ are $A_1=C_1$ $\underline{\text{comp}}$ α_1 and $A_2(x)=C_2(x)[x{:}A_1]$ $\underline{\text{comp}}$ α_2 and the parts of $C=A$ are $C_1=A_1$ $\underline{\text{comp}}$ α_1' and $C_2(x)=A_2(x)[x{:}C_1]$ $\underline{\text{comp}}$ α_2'

Now $\alpha_1=\alpha_1'$, by inductive hypothesis point 3.ii, because $\alpha_1<\beta$ and, since, by ind. hyp. point 3.v, $\text{comp}(C_2(x)=A_2(x)[x{:}A_1])=\alpha_2'$, it follows that $\alpha_2=\alpha_2'$, again by using the ind. hyp. point 3.ii, because $\alpha_2<\beta$.

- $G_A \equiv \mathbf{I}(A_1,a',a'')$ and hence $G_C \equiv \mathbf{I}(C_1,c',c'')$.

The parts of $A=C$ are $A_1=C_1$ $\underline{\text{comp}}$ α_1, $a'=c'$ A_1 $\underline{\text{comp}}$ α_2 and $a''=c''$ A_1 $\underline{\text{comp}}$ α_3 and the parts of $C=A$ are $C_1=A_1$ $\underline{\text{comp}}$ α_1', $c'=a'$ C_1 $\underline{\text{comp}}$ α_2' and $c''=a''$ C_1 $\underline{\text{comp}}$ α_3'. Now $\alpha_1=\alpha_1'$ by ind. hyp. point 3.ii because $\alpha_1<\beta$ and, since, by ind. hyp. point 4.ii, $\text{comp}(c'=a'\ \ A_1)=\alpha_2$ and $\text{comp}(c''=a''\ \ A_1)=\alpha_3$, we obtain $\text{comp}(c'=a'\ \ C_1)=\alpha_2$ and $\text{comp}(c''=a''\ \ C_1)=\alpha_3$ by using ind. hyp. point 3.iv.

Point 3.ii Subcase $C \neq \emptyset$.

We have to prove that the two considered judgements have substituted judgements with the same computational complexity.

(*substitution* :=) Immediate by ind. hyp. 3.ii.

(*substitution* <−) For any c.c.s. $\underline{e}=\underline{f}$: fitting with C also $\underline{e}$: is a c.c.s. fitting with C, hence by ind. hyp. (2.i and 2.ii)

$$\begin{aligned}\text{comp}(A[\underline{x}:=\underline{e}]=C[\underline{x}:=\underline{e}]) &= \text{comp}(A[\underline{x}:=\underline{e}]=A[\underline{x}:=\underline{f}])\\ &= \text{comp}(A[\underline{x}:=\underline{e}]=A[\underline{x}:=\underline{e}])\\ &= \text{comp}(C[\underline{x}:=\underline{e}]=A[\underline{x}:=\underline{f}])\end{aligned}$$

$$\begin{aligned}\text{comp}(A[\underline{x}:=\underline{e}]=C[\underline{x}:=\underline{e}]) &= \text{comp}(A[\underline{x}:=\underline{e}]=C[\underline{x}:=\underline{f}])\\ &= \text{comp}(A[\underline{x}:=\underline{e}]=A[\underline{x}:=\underline{e}])\\ &= \text{comp}(C[\underline{x}:=\underline{e}]=C[\underline{x}:=\underline{f}]).\end{aligned}$$

Hence $\text{comp}(C[\underline{x}:=\underline{e}]=A[\underline{x}:=\underline{f}]) = \text{comp}(A[\underline{x}:=\underline{e}]=C[\underline{x}:=\underline{f}])$.

Point 3.iii.a.

Let us prove the if-part (the proof of the only-if part is completely similar) by subordinate induction on the complexity α.

Note that, by point 3.i, the associate judgements of a $A[C]$ and a $C[C]$ are computable judgements with the same complexity.

Point 3.iii.a Subcase C =**Ø**.

(*evaluation*) $a \quad g_a$, by hypothesis

(*correct evaluation*) $a=g_a$ C is provable, immediate

(*parts*) Let us analyze three cases according to the form of G_C.

- $G_C \equiv \Sigma(C_1,C_2)$ and hence $G_A \equiv \Sigma(A_1,A_2)$ and $g_a \equiv \mathbf{\&}(a', a'')$ and we know that

  $A_1=C_1$ <u>comp</u> $\alpha_1<\beta$,

  $A_2(x)=C_2(x)$ $[x{:}A_1]$ <u>comp</u> $\alpha_2<\beta$,

  $a' \quad A_1$ <u>comp</u> $\alpha'<\alpha$ and $a'' \quad A_2(a')$ <u>comp</u> $\alpha''<\alpha$

and hence comp($a' \quad C_1$)= α', by ind. hyp. point 3.iii.a, and, since

  comp($A_2(a')=C_2(a')$)<comp($A_2(x)=C_2(x)[x{:}A_1]$)<comp($A=C$)<β,

we obtain $a'' \quad C_2(a')$ <u>comp</u> α'', by using ind. hyp. point 3.iii.a.

- $G_C \equiv \mathbf{I}(C_1,c',c'')$ and hence $G_A \equiv \mathbf{I}(A_1,a',a'')$ and $g_a \equiv \mathbf{r}$ and we know that

  $a'=c' \quad A_1$ <u>comp</u> $\alpha_2<\beta$,

  $a''=c'' \quad A_1$ <u>comp</u> $\alpha_3<\beta$

  $a'=a'' \quad A_1$ <u>comp</u> $\alpha'<\alpha$

and, by point 1., we obtain $c'=c'' \quad A_1$ <u>comp</u> α'

then, since $A_1=C_1$ <u>comp</u> $\alpha_1<\beta$, by using ind. hyp. point 3.iii.b, comp($c'=c'' \quad C_1$)= α'.

- $G_C \equiv \mathbf{W}(C_1,C_2)$ and hence $G_A \equiv \mathbf{W}(A_1,A_2)$ and $g_a \equiv \mathbf{sup}(a',a'')$ and we know that

  $A_1=C_1$ <u>comp</u> $\alpha_1<\beta$,

  $a' \quad A_1$ <u>comp</u> $\alpha'<\alpha$ and

then, by ind. hyp. 3.iii.a, $a' \quad C_1$ <u>comp</u> α'.

Moreover we know that

  $a''(y) \quad \mathbf{W}(A_1,A_2)[y{:}A_2(a')]$ <u>comp</u> $\alpha''<\alpha$

  $A_2(x)=C_2(x)$ $[x{:}\mathrm{A}_1]$ <u>comp</u> $\alpha_2<\beta$

and, since comp($A_2(a')=C_2(a')$)<comp($A_2(x)=C_2(x)[x{:}A_1]$)<β, by ind. hyp. point 3.iv,

  $a''(y) \quad \mathbf{W}(A_1,A_2)[y{:}C_2(a')]$ <u>comp</u> $\alpha''<\alpha$.

Now to prove $a''(y) \quad \mathbf{W}(C_1,C_2)[y{:}C_2(a')]$ <u>comp</u> α'' we must consider

(*associate judgements*) Since

  comp($\mathbf{W}(A_1,A_2)$ **type**)=comp($\mathbf{W}(C_1,C_2)$ **type**)

and

  comp($\mathbf{W}(A_1,A_2)=\mathbf{W}(A_1,A_2)$)=comp($\mathbf{W}(C_1,C_2)=\mathbf{W}(C_1,C_2)$)

then

  comp($\mathbf{W}(A_1,A_2)$ **type**$[y{:}C_2(a')]$)=comp($\mathbf{W}(C_1,C_2)$ **type**$[y{:}C_2(a')]$)

(*substitutions*) By subordinate ind. hyp. 3.iii.a and 3.iii.b, for any c.c.s. $e \quad C_2(a')$ and $e=f \quad C_2(a')$ we have

  comp($a''(e) \quad \mathbf{W}(A_1,A_2)$)=comp($a''(e) \quad \mathbf{W}(C_1,C_2)$)

  comp($a''(e)=a''(f) \quad \mathbf{W}(A_1,A_2)$)=comp($a''(e)=a''(f) \quad \mathbf{W}(C_1,C_2)$)

Point 3.iii.a Subcase $C \neq$ **Ø**.

(*substitution* :=) the result immediately follows by using the ind. hyp. point 3.iii.a.

(*substitution* **<−**) the result immediately follows by using the ind. hyp. point 3.iii.b.

<u>Point 3.iii.b</u>.

The proof of this case is similar to the previous point 3.iii.a. Just note that, by point 3.iii.a, the associate judgements of $a=c \quad \mathsf{A}[C]$ and $c=a \quad C[C]$ are computable with the same complexity.

<u>Point 3.iv</u>.

The proof is by subordinate induction on the computational complexity α. Let us prove the if-part (the proof of the only if-part is completely similar) .
(*associate judgements*) By subordinate ind. hyp. the associate judgements of $J[C, x{:}\mathrm{C}]$ are computable with the same complexity of those of $J[C, x{:}\mathrm{A}]$.
(*substitutions*) For analyzing the substituted judgements, let C be $[y_1{:}B_1,\ldots,y_n{:}B_n]$. Since $A=C[C]$ <u>comp</u> β then for any c.c.s. $a_1{:}\underline{B}_1,\ldots,a_n{:}\underline{B}_n, e{:}\underline{C}$ (or $a_1=a'_1{:}\underline{B}_1,\ldots,a_n=a'_n{:}\underline{B}_n, e=e'{:}\underline{C}$) fitting with $[C, x{:}C]$, we have $\mathrm{comp}(A[\underline{y}:=\underline{a}]=C[\underline{y}:=\underline{a}])<\beta$ hence, by point 3.iii.a (or 3.iii.b), $e{:}\underline{A}$ (or $e=e'{:}\underline{A}$) is a computable judgement and thus the same substitution fits also with $[C, x{:}A]$. Hence the substituted judgements are computable with the same complexity.

<u>Point 4.i</u>.

Let us prove the if-part (the proof of the only if-part is completely similar).
(*associate judgements*) The associate judgement of both judgements is A **type**.
Point 4.i Subcase $C = \emptyset$.

We must prove only that the parts of $c \quad A$ have the same computational complexity of the corresponding parts of $a \quad A$. According to the values of A and a we consider here only three cases:

- A»$\Sigma(A_1,A_2)$ and $a \quad \&(a',a'')$. Then we know that
  $c \quad \&(c',c'')$, and that
  $\mathrm{comp}(a'=c' \quad A_1) < \beta$ and
  $\mathrm{comp}(a''=c'' \quad A_2(a')) < \beta$;

therefore:
  $\mathrm{comp}(a' \quad A_1)= \mathrm{comp}(c' \quad A_1)$, by ind. hyp. 4.i; and, since
  $\mathrm{comp}(A_2(a')=A_2(c')) \quad < \mathrm{comp}(A_2(x)\ \textbf{type}[x{:}A_1])$
  $< \mathrm{comp}(A\ \textbf{type})<\mathrm{comp}(a \quad A)$
  $< \mathrm{comp}(a=c \quad A)=\beta$,

we obtain $\mathrm{comp}(a'' \quad A_2(a'))= \mathrm{comp}(c'' \quad A_2(c'))$, by using ind. hyp. point 4.i and point 3.iii.a.

- A»$\mathbf{W}(A_1,A_2)$ and $a \quad \mathbf{sup}(a',a'')$. Then we know that
  $c \quad \mathbf{sup}(c',c'')$, and that
  $\mathrm{comp}(a'=c' \quad A_1)<\beta$

therefore
  $\mathrm{comp}(a' \quad A_1)=\mathrm{comp}(c' \quad A_1)$, by using ind. hyp. point 4.i.

We know also that
  $\mathrm{comp}(a''(x)=c''(x) \quad \mathbf{W}(A_1,A_2)[x{:}A_2(a')])<\beta$

and

$\mathrm{comp}(A_2(a')=A_2(c'))$ $\quad <\mathrm{comp}(A_2(x)=A_2(x)[x{:}A_1])$

$\quad <\mathrm{comp}(A_2(x)\mathbf{type}[x{:}A_1]<\beta$,

by ind. hyp. point 4.i and point 3.iv, then we obtain

$\mathrm{comp}(a''(x) \quad \mathbf{W}(A_1,A_2)[x{:}A_2(a')])$

$= \mathrm{comp}(c''(x) \quad \mathbf{W}(A_1,A_2)[x{:}A_2(a')])$

$= \mathrm{comp}(c''(x) \quad \mathbf{W}(A_1,A_2)[x{:}A_2(c')])$

- A»$\mathbf{U}$ and $a \quad \pi\,(a',a'')$, and therefore $c \quad \pi\,(c',c'')$. We know that

  $\mathrm{comp}(a'=c' \quad \mathbf{U})<\beta$ and

  $\mathrm{comp}(a''(x)=c''(x) \quad \mathbf{U}\,[x{:}<a'>])<\beta$,

therefore, by ind. hyp. (4.i),

$\mathrm{comp}(a' \quad \mathbf{U})= \mathrm{comp}(c' \quad \mathbf{U})$ and

$\mathrm{comp}(a''(x) \quad \mathbf{U}\,[x{:}<a'>]) = \mathrm{comp}(c''(x) \quad \mathbf{U}\,[x{:}<a'>])$.

Moreover,

$\mathrm{comp}(a'=c' \quad \mathbf{U})<\beta$

then we know, by lemma 4.13, that

$\mathrm{comp}(<a'>=<c'>)<\beta$,

and we obtain, by using ind. hyp. (3.iv),

$\mathrm{comp}(c''(x) \quad \mathbf{U}\,[x{:}<a'>])=\mathrm{comp}(c''(x) \quad \mathbf{U}\,[x{:}<c'>])$.

Point 4.i Subcase $C \neq \emptyset$.

(*substitution* :=) the result immediately follows by ind. hyp.4.i.

(*substitution* <−) the result follows by ind. hyp. point 1 using the fact that if $\underline{e}=\underline{f}$: is a c.c.s fitting with C so is $\underline{e}$: .

Point 4.ii.

Point 4.ii Subcase $C = \emptyset$.

Since the associate judgement of $a=c \quad A$ are exactly those of $c=a \quad A$, we must prove only that the parts of $c=a \quad A$ have the same computational complexity of the corresponding parts of $a=c \quad A$. The proof is similar to that of the previous point 4.i; let us analyze just one case:

- A»$\Sigma(A_1,A_2)$ and $a \quad \mathbf{\&}(a',a'')$. Then we know that

  $c \quad \mathbf{\&}(c',c'')$,

and that

$\mathrm{comp}(a'=c' \quad A_1)< \beta$ and

$\mathrm{comp}(a''=c'' \quad A_2(a')\,)< \beta$;

therefore:

$\mathrm{comp}(a'=c' \quad A_1)= \mathrm{comp}(c'=a' \quad A_1)$,

by ind. hyp. 4.ii; and, since

$\mathrm{comp}(A_2(a')=A_2(c'))$ $\quad < \mathrm{comp}(A_2(x)\ \mathbf{type}[x{:}A_1])$

$\quad < \mathrm{comp}(A\ \mathbf{type})$

$\quad < \mathrm{comp}(a \quad A)$

$\quad < \mathrm{comp}(a=c \quad A) = \beta$, we obtain comp($a''=c''\ \ A_2(a')$)=comp($c''=a''\ \ A_2(c')$), by using ind. hyp. point 4.ii and point 3.iii.b.

Point 4.ii Subcase $C \neq$ **Ø**.

As for the previous case the result follows by ind. hyp. (4.i. and 4.ii) by using ind. hyp. (point 1.1) and the fact that if $\underline{e}=\underline{f}$: is a c.c.s fitting with C so is $\underline{e}$:.

It is worth noting that, the computability of the associate judgements which were left out from the definition 4.1 of computable judgement (points 2.2.1 and 2.4.1) is now established by the symmetry rules.

4.2.4. The assumption rules

In section 2 we presented the assumption rules of our system; they introduce variables of any arity. The new assumption appears in the context of the conclusion of the rule and for this reason only the case $C \neq$ **Ø** has to be considered in order to prove that assumption rules preserve computability.

Lemma 4.15: (First assumption_rule)

Let a_i**:**$\underline{A}_i[C_i]$ ($1\leq i\leq n$) and **:**$B[C] \equiv A(x_1,\ldots,x_n)$ **type**$[C,\ x_1{:}A_1,\ldots,x_n{:}A_n]$, be computable judgements then the judgement $y(a_1,\ldots,a_n)\ \ A(a_1,\ldots,a_n)[C']$, where C' is the merge without duplication of the contexts $C_1,\ldots,C_n, C, [y{:}B]$, is computable;

Proof: Let $C' \equiv s_1{:}S_1,\ldots, s_k{:}S_k, y{:}B, z_1{:}C_1,\ldots, z_m{:}C_m$, $k\geq 0$, $m\geq 0$, where $z_1{:}C_1,\ldots, z_m{:}C_m$ strictly depend on y: B. First note that, since the context C' extends the contexts C and C_i, $1\leq i\leq n$, by the weakening lemma, we have that $A(x_1,\ldots,x_n)$ **type**$[C', x_1{:}A_1,\ldots,x_n{:}A_n]$ and a_i**:**$\underline{A}_i[C']$ ($1\leq i\leq n$) are computable judgements.

(*associate judgement*) The computability of the judgement $A(a_1,\ldots,a_n)$ **type**$[C']$ follows by the substitution lemma.

(*substitution* :=) Consider any c.c.s. fitting with C': $d_1{:}\underline{S}_1,\ldots, d_k{:}\underline{S}_k, b{:}\underline{B}, c_1{:}\underline{C}_1,\ldots,c_m{:}\underline{C}_m$. Note that, if $A'_i\equiv((s_1,\ldots,s_k)\ A_i)(d_1,\ldots,d_k)$ then

$b{:}\underline{B}$

abbreviates:

$b{:}B[s_1{:=}d_1,\ldots,s_k{:=}d_k] \equiv b(x_1,\ldots,x_n)\ \ ((s_1,\ldots,s_k)\ A)(d_1,\ldots,d_k)\ [x_1{:}A'_1,\ldots,x_n{:}A'_n]$.

Moreover a_i**:**$\underline{A}_i[C']$ ($1\leq i\leq n$) are computable judgements then, by the head substitution lemma,

a_i**:**$\underline{A}_i[C']\ [s_1{:=}d_1,\ldots, s_k{:=}d_k, y{:=}b, z_1{:=}c_1,\ldots, z_m{:=}c_m]$ ($1\leq i\leq n$)

are also computable, and

$((s_1,\ldots,s_k,y,z_1,\ldots,z_m)A_i)(d_1,\ldots,d_k,b,c_1,\ldots,c_m) \equiv ((s_1,\ldots,s_k)A_i)(d_1,\ldots,d_k) \equiv A'_i$, ($1\leq i\leq n$).

Let

$e_i \equiv ((s_1,\ldots,s_k,y,z_1{:},\ldots,z_m)\ a_i)(d_1,\ldots,d_k,b,c_1,\ldots,c_m)$, ($1\leq i\leq n$),

then also the judgement

$b{:}\underline{B}[x_1{:=}e_1,\ldots, x_n{:=}e_n]$ is computable.

But $y{:}B, z_1{:}C_1,\ldots, z_m{:}C_m$, cannot appear in $[C, x_1{:}A_1,\ldots,x_n{:}A_n]$ then

$((s_1,\ldots,s_k,y,z_1,\ldots,z_m)\ A)(d_1,\ldots,d_k,b,c_1,\ldots,c_m) \equiv ((s_1,\ldots,s_k)\ A)(d_1,\ldots,d_k)$

and therefore

$b{:}\underline{B}[x_1{:=}e_1,\ldots,x_n{:=}e_n] \equiv y(a_1,\ldots,a_n) \quad A(a_1,\ldots,a_n)[C'][s_1{:=}d_1,\ldots,s_k{:=}d_k, y{:=}b, z_1{:=}c_1,\ldots, z_m{:=}c_m]$

(*substitution* <−) Consider any c.c.s. fitting with C': $d_1=d'_1{:}\underline{S}_1,\ldots,d_k=d'_k{:}\underline{S}_k, b=b'{:}\underline{B}$, $c_1=c'_1{:}\underline{C}_1,\ldots, c_m=c'_m{:}\underline{C}_m$. The proof proceeds as before by noting that the judgements

$a_i{:}\underline{A}_i[C'][s_1{<-}d_1=d'_1,\ldots,s_k{<-}d_k=d'_k, y{<-}b=b', z_1{<-}c_1=c'_1,\ldots,z_m{<-}c_m=c'_m]$ (1≤i≤n)

are computable and can be substituted for x_i in $b=b'{:}\underline{B}$ obtaining a computable judgement which is exactly

$y(a_1,\ldots,a_n) \quad A(a_1,\ldots,a_n)[C'][s_1{<-}d_1=d'_1,\ldots,s_k{<-}d_k=d'_k, y{<-}b=b', z_1{<-}c_1=c'_1,\ldots, z_m{<-}c_m=c'_m]$

Lemma 4.16 (Second assumption_rule)

Let $a_i=a'_i{:}\underline{A}_i[C_i]$ (1≤i≤n) and ${:}B[C] \equiv A(x_1,\ldots,x_n)$ **type**$[C, x_1{:}A_1,\ldots,x_n{:}A_n]$, be computable judgements then the judgement $y(a_1,\ldots,a_n)=y(a'_1,\ldots,a'_n) \quad A(a_1,\ldots,a_n)[C']$, where C' is the merge without duplication of the contexts $C_1,\ldots,C_n, C, [y{:}B]$, is computable.

Proof: Let $C' \equiv s_1{:}S_1,\ldots, s_k{:}S_k, y{:}B, z_1{:}C_1,\ldots, z_m{:}C_m$,k≥ 0, m≥0, where $z_1{:}C_1,\ldots, z_m{:}C_m$ strictly depend on $y{:}B$.

(*associate judgement*) By hypothesis, $a_i=a'_i{:}\underline{A}_i[C_i]$ (1≤i≤n) are computable judgements then also their associate judgements $a_i{:}\underline{A}_i[C_i]$ (1≤i≤n) are computable. Hence, by the previous lemma, the judgement $y(a_1,\ldots,a_n) \quad A(a_1,\ldots,a_n)[C']$ is computable.

(*substitution* :=) Consider any c.c.s. fitting with C': $d_1{:}\underline{S}_1,\ldots, d_k{:}\underline{S}_k, b{:}\underline{B}, c_1{:}\underline{C}_1,\ldots, c_m{:}\underline{C}_m$. The proof proceeds as before by noting that $b=b{:}\underline{B}$ is computable (reflexivity lemma) and that

$a_i=a'_i{:}\underline{A}_i[C'][s_1{:=}d_1,\ldots,s_k{:=}d_k, y{:=}b, z_1{:=}c_1,\ldots, z_m{:=}c_m]$, (1≤i≤n)

are computable judgements which can be substituted for x_i in $b=b{:}\underline{B}$ obtaining a computable judgement which is exactly

$y(a_1,\ldots,a_n)=y(a'_1,\ldots,a'_n) \quad A(a_1,\ldots,a_n)[C'][s_1{:=}d_1,\ldots,s_k{:=}d_k, y{:=}b, z_1{:=}c_1,\ldots, z_m{:=}c_m]$

(*substitution* <−) Consider any c.c.s. fitting with C': $d_1=d'_1{:}\underline{S}_1,\ldots,d_k=d'_k{:}\underline{S}_k, b=b'{:}\underline{B}$, $c_1=c'_1{:}\underline{C}_1,\ldots, c_m=c'_m{:}\underline{C}_m$. The proof proceeds as before by noting that

$a_i=a'_i{:}\underline{A}_i[C'][s_1{<-}d_1=d'_1,\ldots, s_k{<-}d_k=d'_k, y{<-}b=b', z_1{<-}c_1=c'_1,\ldots, z_m{<-}c_m=c'_m]$ (1≤i≤n)

are computable judgements which can be substituted for x_i in $b=b'{:}\underline{B}$ obtaining a computable judgement which is exactly

$y(a_1,\ldots,a_n)=y(a'_1,\ldots,a'_n) \quad A(a_1,\ldots,a_n)[C'][s_1{<-}d_1=d'_1,\ldots,s_k{<-}d_k=d'_k, y{<-}b=b', z_1{<-}c_1=c'_1,\ldots,z_m{<-}c_m=c'_m]$

4.2.5. The logical rules

We have now to analyse the rules that we call "logical" since they can be used to interpret a logical intuitionistic first order calculus or a logical theory of natural numbers. An informal discussion on the computability of these rules is usually depicted in many of the descriptions of ITT and we follow the same ideas. Nevertheless, we should like to note that in our experience a complete formal proof of computability for these rules cannot be carried on without a substantial use of lemma 4.14 on structural rules.

Lemma 4.17: (Π-formation rules)

The Π-formation rules preserve computability. That is:

1. Let $J_1 \equiv A$ **type**$[C]$ and $J_2 \equiv B(x)$ **type**$[C, x{:}A]$ be computable judgements then the judgement $\Pi(A,B)$ **type**$[C]$ is computable
2. Let $J_1 \equiv A{=}C[C]$ and $J_2 \equiv B(x){=}D(x)[C, x{:}A]$ be computable judgements then the judgement $\Pi(A,B){=}\Pi(C,D)[C]$ is computable.

Proof: By induction on the computational complexity α of J_1.

Case 1 (Π-formation rules)

Subcase $C = \emptyset$.

(*evaluation*) $\Pi(A,B) \gg \Pi(A,B)$

(*correct evaluation*) $\Pi(A,B) = \Pi(A,B)$ is derivable (use formation rule and reflexivity).

(*parts*) They are J_1 and J_2.

Subcase $C \neq \emptyset$.

(*substitution* :=) Consider any c.c.s. $a_1{:}\underline{A}_1,\ldots, a_n{:}\underline{A}_n$ fitting with $C \equiv [x_1{:}A_1,\ldots,x_n{:}A_n]$, then

A **type**$[C][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable with complexity lower then α;

$B(x)$ **type**$[C, x{:}A][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable, by head substitution lemma, hence the result follows by inductive hypothesis (case 1).

(*substitution* $<-$) Consider any c.c.s. $a_1{=}a'_1{:}\underline{A}_1,\ldots, a_n{=}a'_n{:}\underline{A}_n$ fitting with $C \equiv [x_1{:}A_1,\ldots,x_n{:}A_n]$, then

A **type**$[C][x_1 <- a_1{=}a'_1,\ldots, x_i <- a_n{=}a'_n]$

is computable with complexity lower then α;

$B(x)$ **type**$[C, x{:}A][x_1 <- a_1{=}a'_1,\ldots, x_i <- a_n{=}a'_n]$

is computable, by head substitution lemma, hence the result follows by inductive hypothesis (case 2).

Case 2 (Π-formation rules)

Subcase $C = \emptyset$.

(*associate judgements*) The judgements A **type** and $B(x)$ **type**$[x{:}A]$, associate of J_1 and J_2, are computable by definition, and, by lemma 4.14 (3.i) also C **type** and $D(x)$ **type**$[x{:}A]$ are computable with the same computational complexity. By lemma 4.14 (case 3.iv), we know that also $D(x)$ **type**$[x{:}C]$ is computable and hence the result follows by inductive hypothesis (case 1).

(*parts*) They are J_1 and J_2.

Subcase $C \neq \emptyset$.

(*associate judgement*) The associate judgements A **type**$[C]$ of J_1 and $B(x)$ **type**$[C, x{:}A]$ of J_2 are computable judgements. Hence the result follows by ind. hyp. (case 1) since the computational complexity of the judgement A **type**$[C]$ is lower than that of J_1.

(*substitution* :=) and (*substitution* $<-$) Similar to the previous case 1 by using ind. hyp. (case 2).

Lemma 4.18: (Π-introduction rules)

The Π-introduction rules preserve computability. That is:

1. Let $J_1 \equiv b(x)$ $B(x)[C, x{:}A]$ and $J_2 \equiv A$ **type**$[C]$ and $J_3 \equiv B(x)$ **type**$[C, x{:}A]$ be computable

judgements then $\lambda(b)\quad \prod(A,B)[C]$ is computable.

2. Let $J_1 \equiv b(x)=d(x) \quad B(x)\ [C, x{:}A]$ and $J_2 \equiv A$ **type**$[C]$ and $J_3 \equiv B(x)$ **type**$[C, x{:}A]$ be computable judgements then the judgement $\lambda(b)=\lambda(d) \quad \prod(A,B)[C]$ is computable.

Proof: By induction on the computational complexity α of J_2.

<u>Case 1</u> ($\prod$-introduction rules)

(*associate judgement*) Immediate by the previous lemma on $\prod$-formation rules.

Subcase $C = \emptyset$.

(*evaluation*) $\lambda(b) \quad \lambda(b)$

(*correct evaluation*) $\lambda(b)=\lambda(b) \quad \prod(A,B)$ is derivable (use introduction rule and reflexivity).

(*parts*) It is J_1

Subcase $C \neq \emptyset$.

(*substitution* :=) Consider any c.c.s. $a_1{:}\underline{A}_1,\ldots, a_n{:}\underline{A}_n$ fitting with $C \equiv [x_1{:}A_1,\ldots,x_n{:}A_n]$, then

A **type**$[C][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable with complexity lower then α;

$B(x)$ **type**$[C, x{:}A][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable, by head substitution lemma,

$b(x) \quad B(x)\ [C, x{:}A][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable, by head substitution lemma, hence the result follows by inductive hypothesis (case 1).

(*substitution* <–) Consider any c.c.s. $a_1=a'_1{:}\underline{A}_1,\ldots, a_n=a'_n{:}\underline{A}_n$ fitting with $C \equiv [x_1{:}A_1,\ldots,x_n{:}A_n]$, then also $a_1{:}\underline{A}_1,\ldots, a_n{:}\underline{A}_n$ is a c.c.s. fitting with C, and so

A **type**$[C][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable with complexity lower then α;

$B(x)$ **type**$[C, x{:}A][x_1{:=}a_1,\ldots, x_n{:=}a_n]$

is computable, by head substitution lemma,

$b(x) \quad B(x)\ [C, x{:}A][x_1{<}{-}a_1=a'_1,\ldots, x_i{<}{-}a_n=a'_n]$

is computable, by head substitution lemma, hence the result follows by inductive hypothesis (case 2).

<u>Case 2</u> ($\prod$-introduction rules)

Subcase $C = \emptyset$.

(*associate judgements*) The judgement $b(x) \quad B(x)[x{:}A]$, associate of J_1 is computable by definition and, by lemma 4.14 (4.i) also $d(x) \quad B(x)[x{:}A]$ is computable. Hence the result follows by case 1.

(*parts*) It is J_1.

Subcase $C \neq \emptyset$.

(*associate judgements*) The judgement $b(x) \quad B(x)[x{:}A]$, associate of J_1 is computable by definition, hence the result follows by case 1.

(*substitutions*) Similar to the previous case 1 using inductive hypothesis (case 2).

<u>Lemma 4.19</u>: ($\prod$-elimination rules)

The $\prod$-elimination rules preserve computability. That is:

1. Let $J_1 \equiv c \quad \prod(A,B)\ [C], J_2 \equiv d(y) \quad C(\lambda(y))\ [C, y{:}(x{:}A)B(x)]$,

$J_3 \equiv C(t)$ **type** $[C, t{:}\prod(A,B)]$ be computable judgements then the judgement
$\mathbf{F}(c,d) \quad C(c)[C]$ is computable.

2. Let $J_1 \equiv c=c' \quad \prod(A,B)\ [C]$, $J_2 \equiv d(y)=d'(y) \quad C(\lambda(y))\ [C, y{:}(x{:}A)B(x)]$,
$J_3 \equiv C(t)$ **type** $[C, t{:}\prod(A,B)]$ be computable judgements then the judgement
$\mathbf{F}(c,d)=\mathbf{F}(c',d') \quad C(c)[C]$ is computable.

Proof: By induction on the computational complexity α of J_1.

<u>Case 1</u> ($\prod$-elimination rules)

(associate judgements) The computability of the associated judgement $C(c)$ **type** $[C]$ follows by substitution lemma.

Subcase $C = \emptyset$.

(*evaluation*) J_1 is computable and $\prod(A,B) \gg \prod(A,B)$, then $c \quad \lambda(b)$ and that the judgement $b{:}(x{:}A)B(x)$ is computable and it is a c.c.s. fitting with $y{:}(x{:}A)B(x)$; therefore $J_2[y{:=}b]$, which is $d(b) \quad C(\lambda(b))$, is a computable judgement. Hence $d(b) \quad g$ and the result follows by using the computation rule.

(*correct evaluation*) Since J_1 is computable, we know that there exists a derivation of the judgement $\prod(A,B)$ **type** hence $x \quad \prod(A,B)$ is a correct assumption, and $c=\lambda(b) \quad \prod(A,B)$ is derivable. Let Π_1 be the following derivation

$$\dfrac{\dfrac{c=\lambda(b) \quad \prod(A,B) \qquad \dfrac{x \quad \prod(A,B)[x \quad \prod(A,B)] \quad J_2 \quad J_3}{\mathrm{F}(x,d) \quad C(x)[x \quad \prod(A,B)]}}{\mathrm{F}(c,d)=\mathrm{F}(\lambda(b),d) \quad C(c)} \qquad \dfrac{c=\lambda(b) \quad \prod(A,B) \qquad J_3}{C(c)=C(\lambda(b))}}{\mathbf{F}(c,d)=\mathbf{F}(\lambda(b),d) \quad C(\lambda(b))}$$

Since J_1 is computable, so is $b(x) \quad B(x)[x{:}A]$ and then $J_2[y{:=}b]$ is computable. Thus the judgements $d(b)=g \quad C(\lambda(b))$ and $b(x) \quad B(x)[x{:}A]$ are derivable. Let Π_2 be the following derivation

$$\dfrac{\begin{array}{c}\Pi_1\\ \mathbf{F}(c,d)=\mathbf{F}(\lambda(b),d) \quad C(\lambda(b))\end{array} \qquad \dfrac{\dfrac{b(x) \quad B(x)[x{:}A] \quad \prod(A,B)\ \mathbf{type} \quad J_2 \quad J_3}{\mathbf{F}(\lambda(b),d)=d(b) \quad C(\lambda(b))} \qquad d(b)=g \quad C(\lambda(b))}{\mathbf{F}(\lambda(b),d)=g \quad C(\lambda(b))}}{\mathbf{F}(c,d)=g \quad C(\lambda(b))}$$

Hence

$$\dfrac{\begin{array}{c}\Pi_2\\ \mathbf{F}(c,d)=g \quad C(\lambda(b))\end{array} \qquad \dfrac{C(c)=C(\lambda(b))}{C(\lambda(b))=C(c)}}{\mathbf{F}(c,d)=g \quad C(c)}$$

(*parts*) Since J_1 is computable we know that $c \quad \lambda(b)$ and, by fact 4.5 (point 3), that the judgement $\lambda(b) \quad \prod(A,B)$ is computable. Hence, by lemma 4.9 (point i), we can deduce that the judgement $c=\lambda(b) \quad \prod(A,B)$ is computable. Therefore, since J_3 is computable, we obtain that the judgement $C(\lambda(b))=C(c)$ is a computable. Then, since $d(b) \quad C(\lambda(b))$ is a computable judgement, so is $d(b) \quad C(c)$, by lemma 4.14 point 3.iii. Hence, since $d(b) \quad g$, the parts of g, which is also the value of $\mathbf{F}(c,d)$, are computable element(s) in the value of $C(c)$.

Subcase $C \neq \emptyset$

(*substitution* :=) immediate by ind.hyp. (case 1).

(*substitution* <−) immediate by ind. hyp. (case 2)

<u>Case 2</u> ($\prod$-elimination rules)

(*associate judgements*) The computability of the associate judgement $\mathbf{F}(c,d)\ \ C(c)[C]$ follows by case 1. If C is empty, also the computability of $\mathbf{F}(c',d')\ \ C(c')$ follows by case 1. since from the fact that J_1 and J_2 are computable by lemma 4.14 point 4.i we obtain that $c'\ \ \Pi(A,B)$ and $d'(y)\ \ C(\lambda(y))[y:(x:A)B(x)]$ are computable judgements. Then, since the judgement $C(c)=C(c')$ is computable, by lemma 4.14 point 3.iii.a, $\mathbf{F}(c',d')\ \ C(c')$ is computable.

Subcase $C = \emptyset$.

(*parts*) J_1 is computable, then $c\ \ \lambda(b)$, $c'\ \ \lambda(b')$ and the judgement $b(x)=b'(x)\ \ B(x)[x:A]$ is computable. Moreover $b=b':(x:A)B(x)$ is a c.c.s. for $y:(x:A)B(x)$ in J_2, and then $J_2[y\leftarrow b=b']$, that is the judgement $d(b)=d'(b')\ \ C(\lambda(b))$ is computable. Then, by lemma 4.14 point 3.iii.b, $d(b)=d'(b')\ \ C(c)$ is a computable judgement, since, as in the previous point, we can prove that $C(\lambda(b))=C(c)$ is a computable judgement. So, if $d(b)\ \ g_d$ and $d'(b')\ \ g_{d'}$, the parts of g_d and $g_{d'}$ are computable equal elements in the value of $C(c)$.

Subcase $C \neq \emptyset$.

(*substitution* :=) and (*substitution* $\leftarrow$) Immediately follows by ind. hyp. (case 2.).

Lemma 4.20: ($\prod$-equality rule)

The $\prod$-equality rule preserves computability. That is, let

$J_1 \equiv b(x)\ \ B(x)\ [C, x:A]$,

$J_2 \equiv \prod(A,B)$ **type** $[C]$,

$J_3 \equiv d(y)\ \ C(\lambda(y))\ [C, y:(x:A)B(x)]$,

$J_4 \equiv C(t)$ **type** $[C, t:\prod(A,B)]$

be computable judgements then the judgement $\mathbf{F}(\lambda(b),d)=d(b)\ \ C(\lambda(b))\ [C]$ is computable.

Proof: by induction on the computational complexity α of J_2.

(*associated judgements*) J_1 and J_2 are computable, thus, by the $\prod$-introduction lemma, we obtain that $\lambda(b)\ \ \prod(A,B)[C]$ is a computable judgement, and hence $\mathbf{F}(\lambda(b),d)\ \ C(\lambda(b))\ [C]$ is computable, by the previous lemma on Π-elimination rules. Moreover, if C is empty, since J_1 and J_3 are computable, we obtain that $J_3[y:=b]$, i.e. the second associate judgement $d(b)\ \ C(\lambda(b))$, is computable.

Subcase $C = \emptyset$.

Since $\mathbf{F}(\lambda(b),d)$ and $d(b)$ evaluate into the same canonical element, the computability of $\mathbf{F}(\lambda(b),d)=d(b)\ \ C(\lambda(b))$ follows from the computability of the associated judgements by lemma 4.8.

Subcase $C \neq \emptyset$.

(*substitution* :=) It immediately follows by inductive hypothesis.

(*substitution* $\leftarrow$) Consider any c.c.s. $a_1=a'_1:\underline{A}_1,\ldots, a_n=a'_n:\underline{A}_n$ fitting with $C \equiv[x_1:A_1,\ldots,x_n:A_n]$, then also $a_1:\underline{A}_1,\ldots,a_n:\underline{A}_n$ is a c.c.s. fitting with C, and, by ind. hyp. we obtain that

$\mathbf{F}(\lambda(b),d)=d(b)\ \ C(\lambda(b))[C][x_1:=a_1,\ldots,x_n:=a_n]$

is computable. Moreover, since $J_3[y:=b]$ is computable so is $J_3[y:=b][x_1\leftarrow a_1=a'_1,\ldots,x_i\leftarrow a_n=a'_n]$ which is $d(b)\ \ C(\lambda(b))[C][x_1\leftarrow a_1=a'_1,\ldots,x_i\leftarrow a_n=a'_n]$ and then the result follows by transitivity.

For all the other cases, with a few exceptions, the proof goes on analogously to the Π case. In the following we will stress the essential points.

- We proceed always by induction on the computational complexity of the first premise such that none of its assumptions is discharged.
- For each type we must consider the rules in the following association and ordering:
  - the two formation rules
  - the two introduction rules
  - the two elimination rules (with the exception of **I** and **U**)
  - the equality rule.

The ordering is important since, in some cases, to carry on the proof we need to apply a rule which precedes the considered one in the given ordering and therefore we must have already proved that such a rule preserves computability. For instance, when the first introduction rule is considered, the computability of the associate judgement follows by applying the formation rule to some suitable judgements which are among the premises.

The association is important since, when the first of the two associated rules is considered, to prove the computability of the substituted judgements (*substitution* <−) we apply the second rule while, when the second rule is considered, the computability of the associate judgements follows by applying the first rule.

<u>(*associate judgements*)</u>

- For the first introduction rule, the computability of the *associate judgements* follows, as already noted, by applying the formation rule to suitable judgements which are among the premises.
- For the first elimination rule, the computability of the *associate judgements* follows by applying a suitable substitution to one of the premise. **U**-elimination rules are unlike and had been treated in Lemma 4.13.
- For the second formation, introduction or elimination rule, the computability of the *associate judgements* follows, by inductive hypothesis, by applying the first rule to the associate of the premises or to their variants whose computability is assured by definition or by lemma 4.14 and also 4.13 when U-introductions are considered. These lemmas are needed in order to prove the computability of the second associate judgement or to allow switching the assumptions from one type to a computationally equal one . For instance, in the Π case from the computability of the judgements $B(x)=D(x)$ $[x{:}A]$ and $A=C$, we deduced, by lemma 4.14 (3.i and 3.iv) the computability of $D(x)$ **type**$[x{:}C]$ which is a variant of the computationally equal judgement $B(x)$ **type** $[x{:}A]$. Only for the elimination rules, in the case $C \neq \varnothing$, the application of the first rule does not immediately produce the wanted associate: a changing of type is required and allowed by lemma 4.14 since $C(c)=C(c')$ is a computable judgement. Clearly **I**-elimination is an exception (there is only one elimination rule). In this case, when a substitution $\underline{e}=\underline{f}$: is considered in order to prove the computability of a hypothetical judgement $a=b$ $A[C]$ derived from the computable premises c $\mathbf{I}(A,a,b)[C]$, A **type**$[C]$, a $A[C]$,b $A[C]$, the computability of the saturated judgement can be proved as follows. The substitution $\underline{e}$: is first applied to the premises in order to obtain, by inductive hypothesis, that the judgement $a=b$ $A[C][..:=\underline{e}]$ is computable;

then the substitution $\underline{e}=\underline{f}$: is applied to the judgement b $A[C]$; the result follows by transitivity (lemma 4.14 point 1.1).

- For the equality rule, the computability of the first associate is obtained by using an instance of the introduction rule and an instance of the elimination rule of the considered type. In the case $C = \emptyset$, the computability of the other associate judgement is obtained by a suitable use of the substitution rules, that is easy, even if not immediate, also in the case of the inductive types **N** and **W**. The only exception is the type **U** where suitable formation rules, that preserves computability (see Lemma 4.13), must be applied to the judgements of kind **type** that one obtain by using the first **U**-elimination rule.

<u>(*evaluation*), (*correct evaluation*), (*parts*)</u>

- When formation or introduction rules are considered, the points (*evaluation*), (*correct evaluation*), (*parts*), are always immediate.
- As regards elimination rule, the points (*evaluation*), (*correct evaluation*), (*parts*), in the case $C = \emptyset$, must be a little more detailed.

<u>Case 1: first elimination rule.</u>

Let *non-can-el* $C(c)$ be the conclusion of the rule (in the Π-case we have $\mathbf{F}(c,d)$ $C(c)$). First of all, note that there is always a premise of the form c Tp where the outermost constant of the expression Tp characterizes the type to which the elimination refers (in the Π-case we have c $\Pi(A,B)$), a hypothetical type-judgement depending on Tp (in the Π-case we have $C(t)$ **type**[t:$\Pi(A,B)$]) and one or more other minor premises (in the Π-case we have $d(y)$ $C(\lambda(y))[y{:}(x{:}A)B(x)]$). Then the proof gets on in the following way.
(*evaluation*) The canonical value g_C of c ($\lambda(b)$ in the Π-case), which exists since the major premise c Tp is computable, allows one to choose which minor premises to analyze (in the Π-case there is only one minor premise: $d(y)$ $C(\lambda(y))[y{:}(x{:}A)B(x)]$). When this is a hypothetical judgement it must be saturated and the part judgements of the major premise gives us some of the substitutions needed to saturate it (in the Π-case we obtained $d(b)$ $C(\lambda(b))$. This saturated judgement, *sat-el* $C(g_C)$ is computable and its evaluation is exactly what we are looking for. Usually the parts of the major premise together with the other premises provides all the needed substitutions; exceptions are the cases **U**, which had been considered in Lemma 4.13, **N** and **W** where an induction on the complexity of the major premise is necessary to build the suitable substitution. Let us develop these two cases in detail.

<u>**N**-elimination</u>

The premises are c **N**, d $C(\mathbf{0})$, $e(x,y)$ $C(\mathbf{s}(x))[x{:}\mathbf{N}, y{:}C(x)]$ and $C(t)$ **type**[t:**N**]. c **N** is computable thus either c $\mathbf{0}$ or c $\mathbf{s}(a)$. If c $\mathbf{0}$ then we choose d $C(\mathbf{0})$ among the minor premises and the value of d, which exists since d $C(\mathbf{0})$ is computable, is just the value of $\mathbf{R}(c,d,e)$. Otherwise, if c $\mathbf{s}(a)$, we choose $e(x,y)$ $C(\mathbf{s}(x))[x{:}\mathbf{N}, y{:}C(x)]$. a **N** is computable then a:**N** is a c.c.s. fitting with x:**N**. comp(a **N**)<comp(c **N**) thus, by ind. hyp., $\mathbf{R}(a,d,e)$ $C(a)$ is computable and a:**N**, $\mathbf{R}(a,d,e){:}C(a)$ is a c.c.s. fitting with x:**N**, $y{:}C(x)$. Hence $e(a,\mathbf{R}(a,d,e))$ $C(\mathbf{s}(a))$ is computable and the value of $e(a,\mathbf{R}(a,d,e))$ is just the value of $\mathbf{R}(c,d,e)$.

W-elimination

$c \quad \mathbf{W}(A,B)$ is computable, then $c \quad \mathbf{sup}(a,b)$, $a \quad A$ and $b(x) \quad \mathbf{W}(A,B)[x:B(a)] \equiv b:(x:B(a))\mathbf{W}(A,B)$ are computable judgements. Hence $a:A,b:(x:B(a))\mathbf{W}(A,B)$ is a c.c.s. fitting with $z:A$, $y:(x:B(z))\mathbf{W}(A,B)$ and since comp($b(x) \quad \mathbf{W}(A,B)[x:B(a)]$) < comp($c \quad \mathbf{W}(A,B)$), by applying again the same rule with $b(x) \quad \mathbf{W}(A,B)[x:B(a)]$ instead of $c \quad \mathbf{W}(A,B)$ we obtain, by inductive hypothesis, that $\mathbf{T}(b(x), d) \quad C(b(x))\ [x:B(a)] \equiv (x)\mathbf{T}(b(x),d):(x:B(a))C(b(x))$ is computable and is a c.c.s. fitting with $t:(x:B(a))C(b(x))$. Then, by substituting, we obtain that $d(a,b,(x)\mathbf{T}(b(x), d)) \quad C(\mathbf{sup}(a,b))$ is computable and the value of $d(a,b,(x)\mathbf{T}(b(x), d))$ is exactly the value of $\mathbf{T}(c,d)$ we are looking for.

(*correct evaluation*) For each canonical value of the major premise a derivation can be constructed analogously to the Π-case. It is sufficient to substitute any application of Π-elimination and Π-equality rules by the corresponding one for the considered type.

(*parts*) The computability of the major premise $c \quad Tp$ guarantees, by lemmas 4.5 and 4.9.i, the computability of the judgement $c=g_C \quad Tp$ (in the Π-case we have $c=\lambda(b) \quad \Pi(A,B)$). This allows us to obtain the computability of the type equality: $C(c)=C(g_C)$ ($C(c)=C(\lambda(b)$ in the Π-case). At this point if we consider the computable judgement built up to prove the previous *evaluation* point, *satel* $\quad C(g_C)$ (in the Π-case we have $d(b) \quad C(\lambda(b))$), by lemma 4.14 point 3.iii, we obtain that *sat-el* $\quad C(c)$ is computable (in the Π-case we have $d(b) \quad C(c)$). Hence if *sat-el* $\quad$ *can-el* and $C(c) \quad$ *can-C*, then also *non-can-el* $\quad$ *can-el* and the parts of *can-el* are computable elements in *can-C.*

Case 2: second elimination rule.

Let *non-can-el*$_1$ = *non-can-el*$_2$ $\quad C(c)$ be the conclusion of the rule.

(*parts*) First of all, note that the computability of the first associate of the major premise, $c=c' \quad Tp$, guarantees the computability of the judgement $C(c) = C(g_C)$. Then, analogously to the case 1 of the first elimination rule, we can choose the suitable minor premise and saturate it by using <− instead of :=. By the computability of the resulting judgement *sat-el*$_1$=*sat-el*$_2$ $\quad C(g_C)$ together with that of $C(c) = C(g_C)$, we will obtain the computability of *sat-el*$_1$=*sat-el*$_2$ $\quad C(c)$. From this the result is immediate.

- For the equality rule, the point (*parts*), follow easily since, by lemma 4.9.i (or 4.9.ii. for **U**), the computability of the associate judgements together with the definition of , guarantees the computability of the judgement in the conclusion.

(*substitution* :=)

- The point (*substitution* :=) always follows, by induction, by first applying the same substitution to the premises and next applying again the same rule to the resulting judgements. Note that when a rule which discharges assumptions is considered, we must apply a head substitution which preserves computability.

(*substitution* <−)

- For the first formation, introduction or elimination rule, the point (*substitution* <−) follows, by inductive hypothesis, by applying the second rule in the association to judgements obtained by properly substituting the given premises. In some cases, when a rule which discharges assumptions is considered, the computability of the suitably substituted premises is stated by the head substitution lemma.

- For the second formation, introduction or elimination rule, the proof of the point (*substitution* <−) follows by applying the same rule to judgements obtained by wisely applying the same substitution $\underline{e}$=$\underline{f}$: or its associate $\underline{e}$: to the given premises.
- For the equality rule, when the substitution $\underline{e}$=$\underline{f}$: is considered in proving the point (*substitution* <−), we will proceed as follows. On one side we apply the same rule to judgements obtained by applying the associate substitution $\underline{e}$: to the premises. On the other side, we apply the substitution $\underline{e}$=$\underline{f}$: to a judgement built up by applying a suitable head substitution to the minor premise analogously to what done for the (*evaluation*) point. The result then follows by transitivity (consider again the Π-case as a typical example). For the **U** case we must build up a first judgement by applying the same rule to judgements obtained by applying the substitution $\underline{e}$: to the given premises, and a second one by applying a formation rule to the result of applying the **U**-elimination to the premises. The result then follows by transitivity. Note that all the rules used in the construction preserve computability.

5. The computability theorem.

Now we can state our main theorem: it shows that any derivable judgement is computable and hence that all the properties we ask for a judgement to be computable hold for any derivable judgement.

Theorem 5.1: (Computability theorem)

Any derivable judgement is computable.

From a proof-theoretical point of view the main meta-theoretical result on a deductive system in natural deduction style as ours, is a normal form theorem, i.e. a theorem that states that any proof can be transformed in a new one with the same conclusion but enjoying stronger structure properties. These properties generally allow in turn to deduce important properties on the considered deduction system such as its consistency. Our computability theorem does not regard derivations but still is strongly related to normal form theorems as the following definitions will clarify.

Definition 5.2: (Canonical proof)

(1) A proof of the judgement *A* **type** or *A*=*B* is *canonical* when its last inference step is obtained by a formation rule.

(2) A proof of the judgement *a*   *A* or *a*=*b*   *A* is *canonical* when its last inference step is obtained by an introduction rule.

A canonical proof might be also called "normal at the end". Clearly not every closed judgement can be derived by a canonical proof. This holds only for the judgements which, according to the following definition, are in canonical form.

<u>Definition 5.3</u>: (Canonical form)

Let J be a closed judgement,

if $J \equiv A$ **type** and $A \gg G_A$ then the canonical form of J is G_A **type**;

if $J \equiv A{=}B$ and $A \gg G_A$ and $B \gg G_B$ then the canonical form of J is $G_A{=}G_B$;

if $J \equiv a \quad A$ and $a \quad g_a$ and $A \gg G_A$ then the canonical form of J is $g_a \quad G_A$;

if $J \equiv a{=}b \quad A$ and $a \quad g_a$ and $b \quad g_b$ and $A \gg G_A$ then the canonical form of J is $g_a{=}g_b \quad G_A$.

<u>Corollary 5.4</u>: (Canonical-form theorem)

Let J be a derivable closed judgement then there exists a canonical proof of the canonical form of J.

Proof. Since J is derivable then it is computable and hence there exist a derivation of its parts judgements since they also are computable. By putting them together with a formation or an introduction rule we obtain a canonical proof of the canonical form of J.

It is easy to see that if J is a derivable closed judgement then its computability implies that its canonical form is a judgement equivalent to J, in fact:

- if $J \equiv A$ **type** and $A \gg G_A$ then the canonical form of J is G_A **type** and the computability of J assures that $A{=}G_A$ is derivable.
- if $J \equiv A{=}B$ and $A \gg G_A$ and $B \gg G_B$ then the canonical form of J is $G_A{=}G_B$ and the computability of J assures that $A{=}G_A$ and $B{=}G_B$ are derivable judgements.
- if $J \equiv a \quad A$ and $a \quad g_a$ and $A \gg G_A$ then the canonical form of J is $g_a \quad G_A$ and the computability of J assures that $A{=}G_A$ and $a{=}g_a \quad A$ are derivable judgements.
- if $J \equiv a{=}b \quad A$ and $a \quad g_a$, $b \quad g_b$ and $A \gg G_A$ then the canonical form of J is $g_a{=}g_b \quad G_A$ and the computability of J assures that $A{=}G_A$, $a{=}g_a \quad$ A and $b{=}g_b \quad A$ are derivable judgements.

Then the previous canonical form theorem is, in our system, the counterpart of a standard normal form theorem since it guarantees that if J is a closed derivable judgement then we can construct a canonical proof for a judgement equivalent to J. Moreover it allows us to deduce most of the results usually obtained by a normal form theorem such as, for instance, consistency.

<u>Corollary 5.5</u>: (Consistency of HITT)

The Higher order Intuitionistic Theory of Type is consistent.

Proof: Since the judgement $c \quad \mathbf{N_0}$ is not computable, see Fact 4.4, then it cannot be derivable.

Note that this result establishes also the consistency of the original ITT. As we could expect, the minimal properties, which are usually asked for a logical system to be considered constructive, immediately follow just by reading the definition of computable judgement.

<u>Corollary 5.6</u>: (Disjunction property)

If the judgement c $+(A,B)$ is derivable then either there exists an element a such that a A is derivable or there exists an element b such that b B is derivable.

Proof: If c $+(A,B)$ is derivable then it is computable and hence either c $\mathbf{i}(a)$ and a A is derivable or c $\mathbf{j}(b)$ and b B is derivable.

<u>Corollary 5.7</u>: (Existential property)

If the judgement c $\Sigma(A,B)$ is derivable then there exists an element b such that the judgement b $B(a)$ is derivable for some a A.

Proof: If c $\Sigma(A,B)$ is derivable then it is computable and hence c & (a, b) and a A and b $B(a)$ are derivable judgements.

Other consequences of the computability theorem can be stated when the Intuitionistic Theory of Type is viewed as a formal system to derive programs, that is when a type is interpreted as the specification of a problem and an element in this type as the program which meets this specification. In this environment an expression denoting an element in a type is thought as a program written in a functional language whose operational semantics is given by the computation rules and hence to execute a program corresponds to evaluating it. The computability theorem shows that whenever we prove that program a is partially correct with respect to its specification A, i.e. we derive the judgement a A, then we know also that it is totally correct, i.e. its evaluation terminates.

<u>Corollary 5.8</u>: (Evaluation theorem)

(1) Let A **type** be a provable judgement, then A has a canonical value.

(2) Let a A be a provable judgement, then a has a canonical value.

Thus any program whose evaluation does not terminate, such as the famous Church's non-terminating function, cannot be typed in HITT.

Related works

We should like to thank the referee for his comments and suggestions and for pointing out to us related works, in particular [All86] and [All87] where similar results are proved by a realizability-like semantics.

References

[All86] Allen, S.F. A non-type-theoretic definition of Martin-Löf’s types. *Proceedings of the 1st Annual Symposium on Logic in Computer Science*, IEEE, 1986, 215-221

[All87] Allen, S.F. *A non-type-theoretic semantics for a type-theoretic language*, PhD Thesis, Cornell University, 1987.

[Bar81] Barendreght, H. *The lambda calculus - its syntax and semantics*, North-Holland, Amsterdam, 1981.

[Bac89] Backhouse, R. Chisholm, P. Malcom, G. and Saaman, E. Do-it-Yourself Type Theory. *Formal Aspects of Computing* 1 (1989), 19-84.

[BaC85] Bates, J.L. and Constable, R. L. Proofs as Programs, *ACM Trans. on Programming Languages and Systems* 7, N.1 (January 1985), 94-117.

[Bee85] Beeson, M.J. *Foundation of Constructive Mathematics*, Springer-Verlag, 1985.

[BoV85] Bossi, A. and Valentini, S. *The Expressions with arity*, Rapporto interno (Luglio 1985).

[BoV87] Bossi, A. and Valentini, S. Assunzioni di arietà superiore nella teoria intuizionistica dei tipi, (in Italian). Atti secondo convegno nazionale sulla Programmazione Logica, Torino 1987, B. Demo (ed.), 81-92.

[Bru91] de Bruijn, N.G. Telescopic mappings in typed lambda calculus. *Information and Computation*, 91(2), April 1991, 189-204.

[Mar71] Martin-Löf, P. Hauptsatz for the intuitionistic theory of iterated inductive definitions, in: J.E. Fenstad (ed.), *Proceedings of the second Scandinavian logic symposium*, North-Holland, Amsterdam, 1971, 179-216.

[Mar75] Martin-Löf, P. An Intuitionistic Theory of Types: Predicative Part, in: H. E. Rose and J. C. Shepherdson (eds), *Logic Colloquium 1973*, North-Holland, Amsterdam, 1975, 73-118.

[Mar82] Martin-Löf, P. *Constructive Mathematics and Computer Programming*, in: Cohen, L.J. Los, J. Pfeiffer, H. and Podewski, K. P. (eds), Proceedings of the 6th International Congress for Logic, Methodology and Philosophy of Science, IV, Hannover, 1979. North-Holland, Amsterdam (1982), pp.153-175.

[Mar84] Martin-Löf, P. *Intuitionistic Type Theory*, Notes by Giovanni Sambin of a series of lectures given in Padova. Bibliopolis, Napoli, 1984.

[NoP83] Nordstrom, B. and Petersson, K, Types and Specifications, in: R.E.A. Mason ed., *Proceedings IFIP '83*, Paris, Elsevier Science Publishers (North-Holland), Amsterdam 1983.

[NoS84] Nordstrom, B. and Smith, J.M. Propositions, Types and Specifications of Programs in Martin-Lof's Type Theory, *BIT*, 24, N.3 (October 1984), 288-301.

[Nor89] Nordstrom, B., Petersson, K. and J.M. Smith, *Programming in Martin-Lof's Type Theory. An Introduction.* Oxford University Press, 1990.

[PeS86] Petersson, K. and Smith, J.M. Program Derivation in Type Theory: a Partitioning Problem, *Computer Languages*, 11. N.3/4 (1986), 161-172.

[Pra65] Prawitz, D. *Natural Deduction: A Proof-Theoretical Study*, Almqvist & Wiksell, Stockholm, 1965 .

[Tai67] Tait, W. W. Intentional Interpretations of functionals of finite type I. *J.Symbolic Logic*. 32, N.2, (June 1967), 198-212.

[Tro87] Troelstra, A. S. On the Syntax of Martin Löf's Type Theories. *Theoretical Computer Science* 51 (1987), 1-26.

Forms of judgement

$$A\ \textbf{type}\ [C] \qquad A = C\ [C] \qquad a \quad A\ [C] \qquad a{=}c \quad A\ [C]$$

Weakening

$$\frac{J\ [C]}{J\ [C']}$$

where C' is a context extending C

Assumptions

$$\frac{a_1{:}\underline{A}_1\ \ \ldots\ \ a_n{:}\underline{A}_n \qquad\qquad B(w_1,..,w_n)\ \textbf{type}\ [w_1{:}\mathrm{A}_1,..,w_n{:}A_n]}{x(a_1,..,a_n)\quad B(a_1,..,a_n)\ [x{:}(w_1{:}A_1,..,w_n{:}A_n)\ B(w_1,..,w_n)]}$$

where $a_1{:}\underline{A}_1\ \ \ldots\ \ a_n{:}\underline{A}_n$ fits with $w_1{:}A_1,..,w_n{:}A_n$

$$\frac{a_1{=}c_1\boldsymbol{+}\underline{A}_1\ \ldots\ a_n{=}c_n\boldsymbol{+}\underline{A}_n \qquad\qquad B(w_1,..,w_n)\ \textbf{type}\ [w_1{:}A_1,..,w_n{:}A_n]}{x(a_1,..,a_n){=}x(c_1,..,c_n)\quad B(a_1,..,a_n)\ [x{:}(w_1{:}A_1,..,w_n{:}A_n)\ B(w_1,..,w_n)]}$$

where $a_1{=}c_1\boldsymbol{+}\underline{A}_1\ \ldots\ a_n{=}c_n\boldsymbol{+}\underline{A}_n$ fits with $w_1{:}A_1,..,w_n{:}A_n$

Equality rules

Ref

$$\frac{a \quad A}{a{=}a \quad A} \qquad\qquad \frac{A\ \textbf{type}}{A = A}$$

Sim

$$\frac{a{=}c \quad A}{c{=}a \quad A} \qquad\qquad \frac{A = C}{C = A}$$

Tran

$$\frac{a{=}e \quad A \quad e{=}c \quad A}{a{=}c \quad A} \qquad\qquad \frac{A = E \quad E = C}{A = C}$$

Equal types rules

$$\frac{a \quad A \qquad A = C}{a \quad C} \qquad\qquad \frac{a{=}c \quad A \qquad A = C}{a{=}c \quad C}$$

Substitution rules

Let $e_1{:}\underline{E}_1 \dots e_n{:}\underline{E}_n$ and $e_1{=}f_1{:}\underline{E}_1 \dots e_n{=}f_n{:}\underline{E}_n$ fit with $y_1{:}E_1, \dots, y_n{:}E_n$.

$$\frac{e_1{:}\underline{E}_1 \dots e_n{:}\underline{E}_n \qquad A(y_1,..,y_n)\ \mathbf{type}[y_1{:}E_1, \dots ,y_n{:}E_n]}{A(e_1,..,e_n)\ \mathbf{type}}$$

$$\frac{e_1{=}f_1{:}\underline{E}_1 \dots e_n{=}f_n{:}\underline{E}_n \quad A(y_1,..,y_n)\ \mathbf{type}[y_1{:}E_1, \dots ,y_n{:}E_n]}{A(e_1,..,e_n) = A(f_1,..,f_n)}$$

$$\frac{e_1 \boldsymbol{+} \underline{E}_1 \dots e_n \boldsymbol{+} \underline{E}_n \qquad A(y_1,..,y_n){=}C(y_1,..,y_n)[y_1{:}E_1, \dots ,y_n{:}E_n]}{A(e_1,..,e_n){=}C(e_1,..,e_n)}$$

$$\frac{e_1{:}\underline{E}_1 \dots e_n{:}\underline{E}_n \quad a(y_1,..,y_n) \quad A(y_1,..,y_n)[y_1{:}E_1, \dots ,y_n{:}E_n]}{a(e_1,..,e_n) \quad A(e_1,..,e_n)}$$

$$\frac{e_1{=}f_1{:}\underline{E}_1 \dots e_n{=}f_n{:}\underline{E}_n \quad a(y_1,..,y_n) \quad A(y_1,..,y_n)[y_1{:}E_1, \dots ,y_n{:}E_n]}{a(e_1,..,e_n){=}a(f_1,..,f_n) \quad A(e_1,..,e_n)}$$

$$\frac{e_1{:}\underline{E}_1 \dots e_n{:}\underline{E}_n \quad a(y_1,..,y_n){=}c(y_1,..,y_n) \quad A(y_1,..,y_n)[y_1{:}E_1, \dots ,y_n{:}E_n]}{a(e_1,..,e_n){=}c(e_1,..,e_n) \quad A(e_1,..,e_n)}$$

$\prod$-<u>**rules**</u>

Note that in the following rules we suppose that *x*-**0** var , *y*-(**0**)**0** var, *t*-**0** var.

Formation

$$\frac{A\ \mathbf{type} \qquad B(x)\ \mathbf{type}\ [x{:}A]}{\prod(A,B)\ \mathbf{type}} \qquad\qquad \frac{A=C \qquad B(x)=D(x)[x{:}A]}{\prod(A,B) = \prod(C,D)}$$

Introduction

$$\frac{b(x) \quad B(x)[x{:}A] \quad A\ \mathbf{type} \quad B(x)\ \mathbf{type}[x{:}A]}{\lambda(b) \quad \prod(A,B)} \qquad \frac{b(x)=d(x) \quad B(x)[x{:}A] \quad A\ \mathbf{type} \quad B(x)\ \mathbf{type}[x{:}A]}{\lambda(b)=\lambda(d) \quad \prod(A,B)}$$

Elimination

$$\frac{c \quad \prod(A,B) \qquad d(y) \quad C(\lambda(y))[y{:}(x{:}A)B(x)] \qquad C(t)\ \mathbf{type}[t{:}\prod(A,B)]}{\mathbf{F}(c,d) \quad C(c)}$$

$$\frac{c=c' \quad \prod(A,B) \qquad d(y)=d'(y) \quad C(\lambda(y))[y{:}(x{:}A)B(x)] \qquad C(t)\ \mathbf{type}[t{:}\prod(A,B)]}{\mathbf{F}(c,d)=\mathbf{F}(c',d') \quad C(c)}$$

Equality

$$\frac{b(x) \quad B(x)[x{:}A] \qquad \prod(A,B)\ \mathbf{type} \qquad d(y) \quad C(\lambda(y))[y{:}(x{:}A)B(x)] \qquad C(t)\ \mathbf{type}[t{:}\prod(A,B)]}{\mathbf{F}(\lambda(b),d)=d(b) \quad C(\lambda(b))}$$

Computation

$$\prod(A,B) \text{ » } \prod(A,B)$$

$$\lambda(b) \qquad \lambda(b) \qquad\qquad \frac{c \quad \lambda(b) \quad d(b) \qquad g}{\mathbf{F}(c,d) \qquad g}$$

∑-**rules**

Note that in the following rules *x*-**0** var , *y*-**0** var, *t*-**0** var.

Formation

$$\frac{A\ \mathbf{type} \qquad B(x)\ \mathbf{type}[x{:}A]}{\Sigma(A,B)\ \mathbf{type}} \qquad \frac{A{=}C \qquad B(x){=}D(x)[x{:}A]}{\Sigma(A,B) = \Sigma(C,D)}$$

Introduction

$$\frac{a \quad A \quad b \quad B(a) \quad A\ \mathbf{type} \quad B(x)\ \mathbf{type}[x{:}A]}{\mathbf{\&}(a,b) \quad \Sigma(A,B)} \qquad \frac{a{=}c \quad A \quad b{=}d \quad B(a) \quad A\ \mathbf{type} \quad B(x)\ \mathbf{type}[x{:}A]}{\mathbf{\&}(a,b){=}\mathbf{\&}(c,d) \quad \Sigma(A,B)}$$

Elimination

$$\frac{c \quad \Sigma(A,B) \qquad d(x,y) \quad C(\mathbf{\&}(x,y))[x{:}A, y{:}B(x)] \qquad C(t)\ \mathbf{type}[t{:}\Sigma(A,B)]}{\mathbf{E}(c,d) \quad C(c)}$$

$$\frac{c{=}c' \quad \Sigma(A,B) \quad d(x,y){=}d'(x,y) \quad C(\mathbf{\&}(x,y))[x{:}A, y{:}B(x)] \quad C(t)\ \mathbf{type}[t{:}\Sigma(A,B)]}{\mathbf{E}(c,d){=}\mathbf{E}(c',d') \quad C(c)}$$

Equality

$$\frac{a \quad A \quad b \quad B(a) \quad \Sigma(A,B)\ \mathbf{type} \quad d(x,y) \quad C(\mathbf{\&}(x,y))[x{:}A, y{:}B(x)] \quad C(t)\ \mathbf{type}[t{:}\Sigma(A,B)]}{\mathbf{E}(\mathbf{\&}(a,b),d){=}d(a,b) \quad C(\mathbf{\&}(a,b))}$$

Computation

$$\Sigma(A,B) \text{ » } \Sigma(A,B)$$

$$\mathbf{\&}(a,b) \qquad \mathbf{\&}(a,b) \qquad\qquad \frac{c \quad \mathbf{\&}(a,b) \quad d(a,b) \quad g}{\mathbf{E}(c,d) \quad g}$$

+ **-rules**

Note that in the following rules *x*-**0** var , *y*-**0** var, *t*-**0** var.

Formation

$$\frac{A\ \mathbf{type}\qquad B\ \mathbf{type}}{+(A,B)\ \mathbf{type}}\qquad\qquad \frac{A=C\qquad B=D}{+(A,B)=+(C,D)}$$

Introduction

$$\frac{a\quad A\qquad A\ \mathbf{type}\quad B\ \mathbf{type}}{\mathbf{i}(a)\quad +(A,B)}\qquad\qquad \frac{b\quad B\qquad A\ \mathbf{type}\quad B\ \mathbf{type}}{\mathbf{j}(b)\quad +(A,B)}$$

$$\frac{a=c\quad A\qquad A\ \mathbf{type}\quad B\ \mathbf{type}}{\mathbf{i}(a)=\mathbf{i}(c)\quad +(A,B)}\qquad\qquad \frac{b=d\quad B\qquad A\ \mathbf{type}\quad B\ \mathbf{type}}{\mathbf{j}(b)=\mathbf{j}(d)\quad +(A,B)}$$

Elimination

$$\frac{c\quad +(A,B)\qquad d(x)\quad C(\mathbf{i}(x))[x{:}A]\qquad e(y)\quad C(\mathbf{j}(y))[y{:}B]\qquad C(t)\ \mathbf{type}[t{:}+(A,B)]}{\mathbf{D}(c,d,e)\quad C(c)}$$

$$\frac{c=c'\quad +(A,B)\quad d(x)=d'(x)\quad C(\mathbf{i}(x))[x{:}A]\ e(y)=e'(y)\quad C(\mathbf{j}(y))[y{:}B]\ C(t)\ \mathbf{type}[t{:}+(A,B)]}{\mathbf{D}(c,d,e)=\mathbf{D}(c',d',e')\quad C(c)}$$

Equality

$$\frac{a\quad A\quad +(A,B)\ \mathbf{type}\qquad d(x)\quad C(\mathbf{i}(x))[x{:}A]\quad e(y)\quad C(\mathbf{j}(y))[y{:}B]\qquad C(t)\ \mathbf{type}[t{:}+(A,B)]}{\mathbf{D}(\mathbf{i}(a),d,e)=d(a)\quad C(\mathbf{i}(a))}$$

$$\frac{b\quad B\quad +(A,B)\ \mathbf{type}\qquad d(x)\quad C(\mathbf{i}(x))[x{:}A]\quad e(y)\quad C(\mathbf{j}(y))[y{:}B]\qquad C(t)\ \mathbf{type}[t{:}+(A,B)]}{\mathbf{D}(\mathbf{j}(b),d,e)=e(b)\quad C(\mathbf{j}(b))}$$

Computation

$$+(A,B)\ »\ +(A,B)$$

$$\mathbf{i}(a)\qquad \mathbf{i}(a)\qquad\qquad \mathbf{j}(b)\qquad \mathbf{j}(b)\frac{c\qquad \mathbf{i}(a)\quad d(a)\qquad g}{\mathbf{D}(c,d,e)\qquad g}\quad \frac{c\qquad \mathbf{j}(b)\quad e(b)\qquad g}{\mathbf{D}(c,d,e)\qquad g}$$

I-<u>rules</u>

Formation

$$\frac{A\ \mathbf{type} \qquad a\quad A \qquad b\quad A}{\mathbf{I}(A,a,b)\ \mathbf{type}} \qquad\qquad \frac{A{=}C \qquad a{=}c\quad A \qquad b{=}d\quad A}{\mathbf{I}(A,a,b){=}\mathbf{I}(C,c,d)}$$

Introduction

$$\frac{a{=}b\quad A}{\mathbf{r}\quad \mathbf{I}(A,a,b)} \qquad\qquad \frac{a{=}b\quad A}{\mathbf{r}{=}\mathbf{r}\quad \mathbf{I}(A,a,b)}$$

Elimination

$$\frac{c\quad \mathbf{I}(A,a,b) \qquad A\ \mathbf{type} \qquad a\quad A \qquad b\quad A}{a{=}b\quad A}$$

Equality

$$\frac{a{=}b\quad A \qquad c\quad \mathbf{I}(A,a,b)}{c{=}\mathbf{r}\quad \mathbf{I}(A,a,b)}$$

Computation

$$\mathbf{I}(A,a,b) \gg \mathbf{I}(A,a,b)$$

$$\mathbf{r} \qquad \mathbf{r}$$

$\mathbf{N_n}$-rules

Note that in the following rules t-$\mathbf{0}$ var.

Formation

$$\mathbf{N_n}\ \mathbf{type} \qquad\qquad \mathbf{N_n} = \mathbf{N_n}$$

Introduction

$$\mathbf{0_n}\quad \mathbf{N_n},\dots,\ \mathbf{m_n}\quad \mathbf{N_n},\dots,\ \mathbf{n\text{-}1_n}\quad \mathbf{N_n} \qquad \mathbf{0_n}=\mathbf{0_n}\quad \mathbf{N_n},\dots,\ \mathbf{m_n}=\mathbf{m_n}\quad \mathbf{N_n},\dots,\ \mathbf{n\text{-}1_n}=\mathbf{n\text{-}1_n}\quad \mathbf{N_n}$$

Elimination

$$\frac{c\ \ \mathbf{N_n}\quad d_0\ \ C(\mathbf{0_n})\ \dots\ d_{n\text{-}1}\ \ C(\mathbf{n\text{-}1_n})\quad C(t)\ \mathbf{type}[t{:}\mathbf{N_n}]}{\mathbf{R_n}(c,d_0,\dots,d_{n\text{-}1})\quad C(c)}$$

note that if n=0 this is the usual -rule

$$\frac{c=c'\ \ \mathbf{N_n}\quad d_0=d'_0\ \ C(\mathbf{0_n})\ \dots\ d_{n\text{-}1}=d'_{n\text{-}1}\ \ C(\mathbf{n\text{-}1_n})\quad C(t)\ \mathbf{type}[t{:}\mathbf{N_n}]}{\mathbf{R_n}(c,d_0,..,d_{n\text{-}1}) = \mathbf{R_n}(c',d'_0,..,d'_{n\text{-}1})\quad C(c)}$$

Equality

$$\frac{d_0\ \ C(\mathbf{0_n})\ \dots\ d_{n\text{-}1}\ \ C(\mathbf{n\text{-}1_n})\quad C(t)\ \mathbf{type}[t{:}\mathbf{N_n}]}{\mathbf{R_n}(\mathbf{m_n},d_0,\dots,d_{n\text{-}1})=d_m\quad C(\mathbf{m_n})}$$

note that $\mathbf{N_n}$ has n equality-rules. $\mathbf{N_0}$ has no equality-rule.

Computation

$$\mathbf{N_n} \text{ » } \mathbf{N_n}$$

$$\mathbf{m_n}\quad \mathbf{m_n} \qquad\qquad \frac{c\quad \mathbf{m_n}\quad d_m\quad g}{\mathbf{R_n}(c,d_0,..,d_{n\text{-}1})\quad g}$$

note that $\mathbf{N_n}$ has n computation rules for canonical elements and n computation rules for non canonical elements.

$\mathbf{N_0}$ has no computation rule.

N-<u>rules</u>

Note that in the following rules x-**0** var , y-**0** var, t-**0** var.

Formation

$$\mathbf{N\ type} \qquad\qquad \mathbf{N} = \mathbf{N}$$

Introduction

$$\mathbf{0}\quad \mathbf{N} \qquad \frac{a\quad \mathbf{N}}{\mathbf{s}(a)\quad \mathbf{N}} \qquad\qquad \mathbf{0}=\mathbf{0}\quad \mathbf{N} \qquad \frac{a=b\quad \mathbf{N}}{\mathbf{s}(a)=\mathbf{s}(b)\quad \mathbf{N}}$$

Elimination

$$\frac{c\quad \mathbf{N} \qquad d\quad C(\mathbf{0}) \qquad e(x,y)\quad C(\mathbf{s}(x))[x{:}\mathbf{N}, y{:}C(x)] \qquad C(t)\ \mathbf{type}[t{:}\mathbf{N}]}{\mathbf{R}(c,d,e)\quad C(c)}$$

$$\frac{c=c'\quad \mathbf{N} \quad d=d'\quad C(\mathbf{0}) \qquad e(x,y)=e'(x,y)\quad C(\mathbf{s}(x))[x{:}\mathbf{N}, y{:}C(x)] \qquad C(t)\ \mathbf{type}[t{:}\mathbf{N}]}{\mathbf{R}(c,d,e)=\mathbf{R}(c',d',e')\quad C(c)}$$

Equality

$$\frac{d\quad C(\mathbf{0}) \qquad e(x,y)\quad C(\mathbf{s}(x))[x{:}\mathbf{N}, y{:}C(x)] \qquad C(t)\ \mathbf{type}[t{:}\mathbf{N}]}{\mathbf{R}(\mathbf{0},d,e)=d\quad C(\mathbf{0})}$$

$$\frac{c\quad \mathbf{N} \qquad d\quad C(\mathbf{0}) \qquad e(x,y)\quad C(\mathbf{s}(x))[x{:}\mathbf{N}, y{:}C(x)] \qquad C(t)\ \mathbf{type}[t{:}\mathbf{N}]}{\mathbf{R}(\mathbf{s}(c),d,e)=e(c,\mathbf{R}(c,d,e))\quad C(\mathbf{s}(c))}$$

Computation

$$\mathbf{N} \gg \mathbf{N}$$

$$\mathbf{0}\quad \mathbf{0} \qquad \mathbf{s}(a)\quad \mathbf{s}(a) \qquad \frac{c\quad \mathbf{0}\quad d\quad g}{\mathbf{R}(c,d,e)\quad g} \qquad \frac{c\quad \mathbf{s}(a)\quad e(a,\mathbf{R}(a,d,e))\quad g}{\mathbf{R}(c,d,e)\quad g}$$

W-rules

Note that in the following rules x-**0** var, u-**0**, y-(**0**)**0** var, z-(**0**)**0** var, t-**0** var.

Formation

$$\frac{A\ \mathbf{type} \qquad B(x)\ \mathbf{type}[x{:}A]}{\mathbf{W}(A,B)\ \mathbf{type}} \qquad\qquad \frac{A{=}C \qquad B(x){=}D(x)[x{:}A]}{\mathbf{W}(A,B) = \mathbf{W}(C,D)}$$

Introduction

$$\frac{a \quad A \ \ b(x) \quad \mathbf{W}(A,B)[x{:}B(a)] \quad A\ \mathbf{type} \quad B(x)\ \mathbf{type}[x{:}A]}{\mathbf{sup}(a,b) \quad \mathbf{W}(A,B)}$$

$$\frac{a{=}c \quad A \quad b(x){=}d(x) \quad \mathbf{W}(A,B)[x{:}B(a)] \quad A\ \mathbf{type} \quad B(x)\ \mathbf{type}[x{:}A]}{\mathbf{sup}(a,b){=}\mathbf{sup}(c,d) \quad \mathbf{W}(A,B)}$$

Elimination

$$\frac{\begin{array}{c} C(t)\ \mathbf{type}[t{:}\ \mathbf{W}(A,B)] \\ c \quad \mathbf{W}(A,B) \quad d(x,y,z) \quad C(\mathbf{sup}(x,y))[x{:}A, y{:}(t{:}B(x))\ \mathbf{W}(A,B), z{:}(u{:}B(x))\ C(y(u))] \end{array}}{\mathbf{T}(c,d) \quad C(c)}$$

$$\frac{\begin{array}{c} C(t)\ \mathbf{type}[t{:}\ \mathbf{W}(A,B)] \\ c{=}c' \quad \mathbf{W}(A,B) \quad d(x,y,z){=}d'(x,y,z) \quad C(\mathbf{sup}(x,y))[x{:}A, y{:}(t{:}B(x))\ \mathbf{W}(A,B), z{:}(u{:}B(x))\ C(y(u))] \end{array}}{\mathbf{T}(c,d){=}\mathbf{T}(c',d') \quad C(c)}$$

Equality

$$\frac{\begin{array}{c} \mathbf{W}(A,B)\mathbf{type} \qquad\qquad\qquad\qquad C(t)\ \mathbf{type}[t{:}\ \mathbf{W}(A,B)] \\ a \quad A\ b(x) \quad \mathbf{W}(A,B)[x{:}B(a)] \quad d(x,y,z) \quad C(\mathbf{sup}(x,y))[x{:}\mathrm{A}, y{:}(t{:}B(x))\ \mathbf{W}(A,B), z{:}(u{:}B(x))\ C(y(u))] \end{array}}{\mathbf{T}(\mathbf{sup}(a,b),d){=}d(a,b,(x)\mathbf{T}(b(x),d)) \quad C(\mathbf{sup}(a,b))}$$

Computation

$$\mathbf{W}(A,B) \text{ » } \mathbf{W}(A,B)$$

$$\mathbf{sup}(a,b) \qquad \mathbf{sup}(a,b) \ \ \frac{c \quad \mathbf{sup}(a,b) \quad d(a,b,(x)\mathbf{T}(b(x),d)) \quad g}{\mathbf{T}(c,d) \quad g}$$

U-<u>rules</u>

Note that in the following rules *x*-**0** var.

Formation

$$\mathbf{U}\ \mathbf{type} \qquad\qquad \mathbf{U}{=}\mathbf{U}$$

Introduction

$$\frac{a\ \ \mathbf{U}\qquad b(x)\ \ \mathbf{U}[x{:}{<}a{>}]}{\pi(a,b)\ \ \mathbf{U}} \qquad \frac{a{=}c\ \ \mathbf{U}\qquad b(x){=}d(x)\ \ \mathbf{U}[x{:}{<}a{>}]}{\pi(a,b){=}\pi(c,d)\ \ \mathbf{U}}$$

$$\frac{a\ \ \mathbf{U}\qquad b(x)\ \ \mathbf{U}[x{:}{<}a{>}]}{\sigma(a,b)\ \ \mathbf{U}} \qquad \frac{a{=}c\ \ \mathbf{U}\qquad b(x){=}d(x)\ \ \mathbf{U}[x{:}{<}a{>}]}{\sigma(a,b){=}\sigma(c,d)\ \ \mathbf{U}}$$

$$\frac{a\ \ \mathbf{U}\quad b\ \ \mathbf{U}}{+(a,b)\ \ \mathbf{U}} \qquad \frac{a{=}c\ \ \mathbf{U}\qquad b{=}d\ \ \mathbf{U}}{+(a,b){=}+(c,d)\ \ \mathbf{U}}$$

$$\frac{a\ \ \mathbf{U}\ \ b\ \ {<}a{>}\ \ d\ \ {<}a{>}}{\mathbf{i}(a,b,d)\ \ \mathbf{U}} \qquad \frac{a{=}c\ \ \mathbf{U}\ \ b{=}e\ \ {<}a{>}\ \ d{=}f\ \ {<}a{>}}{\mathbf{i}(a,b,d){=}\mathbf{i}(c,e,f)\ \ \mathbf{U}}$$

$$\mathbf{n}_{\mathbf{n}}\ \ \mathbf{U} \qquad\qquad \mathbf{n}_{\mathbf{n}}{=}\mathbf{n}_{\mathbf{n}}\ \ \mathbf{U}$$

$$\mathbf{n}\ \ \mathbf{U} \qquad\qquad \mathbf{n}{=}\mathbf{n}\ \ \mathbf{U}$$

$$\frac{a\ \ \mathbf{U}\qquad b(x)\ \ \mathbf{U}[x{:}{<}a{>}]}{\mathbf{w}(a,b)\ \ \mathbf{U}} \qquad \frac{a{=}c\ \ \mathbf{U}\qquad b(x){=}d(x)\ \ \mathbf{U}[x{:}{<}a{>}]}{\mathbf{w}(a,b){=}\mathbf{w}(c,d)\ \ \mathbf{U}}$$

Elimination

$$\frac{a\ \ \mathbf{U}}{{<}a{>}\ \mathbf{type}} \qquad\qquad \frac{a{=}b\ \ \mathbf{U}}{{<}a{>}{=}{<}b{>}}$$

Equality

$$\frac{a \quad \mathbf{U} \qquad\qquad b(x) \quad \mathbf{U}[x{:}<a>]}{<\pi(a,b)>=\prod(<a>,(x)<b(x)>)}$$

$$\frac{a \quad \mathbf{U} \qquad\qquad b(x) \quad \mathbf{U}[x{:}<a>]}{<\sigma(a,b)>=\Sigma(<a>,(x)<b(x)>)}$$

$$\frac{a \quad \mathbf{U} \qquad b \quad \mathbf{U}}{<+(a,b)>=+(<a>,<b>)}$$

$$\frac{a \quad \mathbf{U} \quad b \quad <a> \quad d \quad <a>}{<\mathbf{i}(a,b,d)>=\mathbf{I}(<a>,b,d)}$$

$$<\mathbf{n_n}>=\mathbf{N_n}$$

$$<\mathbf{n}>=\mathbf{N}$$

$$\frac{a \quad \mathbf{U} \qquad\qquad b(x) \quad \mathbf{U}[x{:}<a>]}{<\mathbf{w}(a,b)>= \mathbf{W}(<a>,(x)<b(x)>)}$$

Computation

$$\mathbf{U} \text{ » } \mathbf{U}$$

$\pi(a,b) \quad \pi(a,b)$ $$\frac{c \quad ❶\pi(a,b)}{<c> \text{ » } \prod(<a>,(x)<b(x)>)}$$

$\sigma(a,b) \quad \sigma(a,b)$ $$\frac{c \quad \sigma(a,b)}{<c> \text{ » } \Sigma(<a>,(x)<b(x)>)}$$

$+(a,b) \quad +(a,b)$ $$\frac{c \quad +(a,b)}{<c> \text{ » } +(<a>,<b>)}$$

$\mathbf{i}(a,b,d) \quad \mathbf{i}(a,b,d)$ $$\frac{c \quad \mathbf{i}(a,b,d)}{<c> \text{ » } \mathbf{I}(<a>,b,d)}$$

$\mathbf{n_n} \quad \mathbf{n_n}$ $$\frac{c \quad \mathbf{n_n}}{<c> \text{ » } \mathbf{N_n}}$$

$\mathbf{n} \quad \mathbf{n}$ $$\frac{c \quad \mathbf{n}}{<c> \text{ » } \mathbf{N}}$$

$\mathbf{w}(a,b) \quad \mathbf{w}(a,b)$ $$\frac{c \quad \mathbf{w}(a,b)}{<c> \text{ » } \mathbf{W}(<a>,(x)<b(x)>)}$$