University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

A canonical normal form theorem for the type theory of regular categories

Supervisor
Prof. Maria Emilia Maietti
University of Padova

Master Candidate
Candidate Riccardo Borsetto

Student ID
2026842

Academic Year
2022-2023

Abstract

In this thesis we prove the canonical-form theorem for the internal language of the regular categories by means of a dependent type theory à la Martin-Löf. This theorem allows to obtain most of the standard consequence of a normalization theorem and it provides a computation model that can be used to build a proof assistant for the theory.

We will consider the calculus of regular categories which contains the terminal type, the indexed sum, the extensional equality and quotients on the terminal type.

The result that we will prove states that any closed typed term, whose derivation has no open assumption, can be reduced by a sequence of reductions into an equivalent one in canonical form, that is, a sort of external normal form, and that the proof of any provable judgement can be transformed into an equivalent one in introductory form.

The present work has been partially formalised in the proof assistant Coq, where the preservation of the computability for the logical rules of the type theory of regular categories has been proved, in the case of empty context.

Contents

1 Introduction

Category theory has become a standard tool in theoretical computer science with applications such as

- Correspondence between lambda calculus and cartesian closed categories.
- Notion of monad that can be used as part of an explanation of side effects in programming languages, for example the ”Monad typeclass” in Haskell.
- F-coalgebra, where F is a functor, have been used as a general framework to model systems (streams, automata, transition systems, probabilistic systems).
- Application of monoidal categories to quantum computation

In this thesis we can see that regular categories enjoy an internal language formulated in terms of a dependent type theory in the style of Martin-Löf's extensional type theory in [1]. In more detail, universal categorical properties correspond to already known type constructors of the extensional type theory as follows

- The terminal type, indexed sum types and the extensional equality types describe the type theory of finitely complete categories and they form the basic module of an extensional dependent type theory internal to a category.
- Stable quotients of kernel pairs correspond to extensional quotient types on the terminal type.

- The parametrised natural numbers object corresponds to the natural numbers type.

However we only have categorical semantics for its extensional version, while only the intensional version can be really thought of a functional programming language because it is strongly normalizing and its definitional equality between terms is decidable. Instead, the extensional version can not be considered a functional programming language since it does not enjoy such properties as its definitional equality between terms is undecidable. Then, the intensional version of type theory, not even enjoying extensionality of functions, is not suitable to develop mathematics as it is. So, having dependent typed calculi available as internal languages of categorical structures, lets us analyze their computational contents, such as investigating the validity of canonical normal form theorems, which help to reduce the gap between extensional mathematical language and normal proof assistants based on intensional calculus.

Our main goal is to prove the canonical form theorem for the type theory of regular categories. To achieve our goal, in chapter 2, we first provide a general introduction to regular categories and their properties. In chapter 3 we therefore define the internal language of regular categories using the theory of dependent types and show how it can be used to reason about these structures in a precise and systematic way. Furthermore, in chapter 4, we propose an extension of the theory with the type of natural numbers. In chapter 5 we present a complete proof of the canonical form theorem and in chapter 6 we prove the theorem for the extended theory of arithmetic regular categories. Finally, in chapter 7, the formalization in the proof assistant Coq is discussed.

This thesis is intended for readers with some background in category theory and type theory, but we strive to make our presentation as self-contained as possible. We hope that this work will contribute to a deeper understanding of regular categories and their role in computer science.

The work reported here is mostly based on the article by Maria Emilia Maietti in [2] for the internal type theory and on the PhD-thesis of Silvio Valentini in [3] for the proof of the computability theorem.

2

Regular categories

Category theory is a special area that studies the fundamental structures used within mathematics. It is based on the very simple notion of an arrow between objects. Category theory is sometimes described as abstract nonsense, but it is often useful because it provides an abstract framework in which similarities between seemingly different notions become apparent. It has become a standard tool in theoretical computer science, especially in the semantics of programming languages. In particular, the categorical description of fixed points, both of recursive functions and of recursive types, captures the relevant "universal" properties that are used in programming and reasoning with these constructs [4].

Definition 2.0.1 (Category). *A category is a mathematical structure consisting of objects with arrows between them, that can be composed.*

More formally, a category $\mathbb{C}$ *consists of a collection* $\mathsf{Obj}(\mathbb{C})$ *of objects and a collection* $\mathsf{Arr}(\mathbb{C})$ *of arrows (also called maps, or morphisms). Usually we write* $X \in \mathbb{C}$ *for* $X \in \mathsf{Obj}(\mathbb{C})$. *Each arrow in* $\mathbb{C}$, *written as* $X \xrightarrow{f} Y$ *or as* $f : X \to Y$, *has a domain object* $X \in \mathbb{C}$ *and a codomain object* $Y \in \mathbb{C}$. *These objects and arrows carry a composition structure.*

1. *For each pair of maps* $f : X \to Y$ *and* $g : Y \to Z$ *there is a composition map* $g \circ f : X \to Z$. *This composition operation* $\circ$ *is associative: if* $h : Z \to W$, *then* $h \circ (g \circ f) = (h \circ g) \circ f$

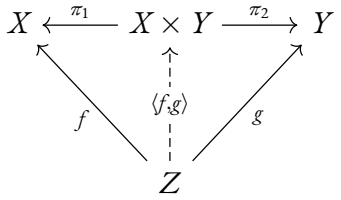
2. *For each object* $X \in \mathbb{C}$ *there is an identity map* $\mathrm{id}_X : X \to X$, *such that* id *is the neutral element for composition* $\circ$: *for* $f : X \to Y$ *one has* $f \circ \mathrm{id}_X = f = \mathrm{id}_Y \circ f$.

Ordinary sets with functions between them form an obvious example of a category, for which we shall write **Set**. Although **Set** is a standard example, it is important to realise that a category may be a very different structure. In particular, an arrow in a category need not be a function. Another example is **Grp**, the category of groups with group homomorphisms (preserving composition and unit, and thereby also inverses).

Definition 2.0.2 (Product). *Let $\mathbb{C}$ be a category. The product of two objects $X, Y \in \mathbb{C}$ is a new object $X \times Y \in \mathbb{C}$ with two projection morphisms*

$$X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$$

which are universal: for each pair of maps $f : Z \to X$ and $g : Z \to Y$ in $\mathbb{C}$ there is a unique tuple morphism $\langle f, g \rangle : Z \to X \times Y$ in $\mathbb{C}$, making the following diagram commute.



Products need not exist in a category, but if they exist they are determined up-to isomorphism. If there is another object with projections

$$X \xleftarrow{p_1} X \otimes Y \xrightarrow{p_2} Y$$

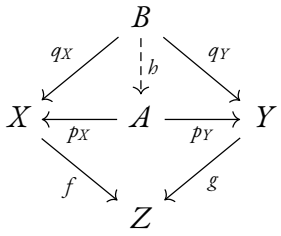
satisfying the above universal property, then there is a unique isomorphism $X \times Y \xrightarrow{\cong} X \otimes Y$ commuting with the projections. What we have described is the product $X \times Y$ of two objects X, Y. For a given X, we shall write $X^n = X \times \cdots \times X$ for the n-fold product (also known as power). The special case where $n = 0$ involves the empty product X^0, called terminal object.

Definition 2.0.3 (Terminal object). *A terminal object in a category $\mathbb{C}$ is an object, usually written as $\top \in \mathbb{C}$, such that for each object $X \in \mathbb{C}$ there is a unique morphism $\star_X : X \to \top$ in $\mathbb{C}$.*

Not every category needs to have a final object, but Sets does. Any singleton set is final. We choose one, and write it as $\top = \{*\}$. Notice then that elements of a set X can be identified with functions $\top \to X$. Hence we could forget about membership $\in$ and talk only about arrows.

When a category has binary products $\times$ and a final object $\top$, one says that the category has finite products: for each finite list $X_1, \ldots, X_n$ of objects one can form the product $X_1 \times \cdots \times X_n$. The precise bracketing in this expression is not relevant, because products are associative (up-to-isomorphism).

Definition 2.0.4 (Pullback). *Let $\mathbb{C}$ be a category, with $f : X \to Z$ and $g : Y \to Z$ arrows. A pullback of f and g consists of an object A together with arrows $p_X : A \to X$ and $p_Y : A \to Y$ such that, for any other object B with arrows $q_X : B \to X$ and $q_Y : B \to Y$, there exists a unique arrow $h : B \to A$ such that the diagram*



commutes.

We recall that for a category having a terminal object and pullbacks is equivalent to being finitely complete.

A regular category is a finitely complete category which admits a good notion of image factorization; they are considered since they have a decently behaved calculus of relations and they provide a natural semantic environment to interpret a particularly well behaved positive fragment of first order logic having connectives $\top$, $\wedge$, $\exists$.

Definition 2.0.5 (Regular category). *A regular category is a finitely complete category with pullback-stable images. [5]*

Examples of regular categories are **Set**, **Grp** and, in general, any abelian category, while **Cat**, **Pos**, and **Top** are not regular.

Since the aim of this thesis is to describe the internal dependent type theory of the regular categories, we list the necessary conditions that a category $\mathbb{C}$ has to satisfy in order to enjoy a dependent typed internal language:

1. $\mathbb{C}$ has to be finitely complete. This is because we want to interpret substitution of terms by means of pullbacks.

2. The structure of $\mathbb{C}$ necessary to interpret the type constructors on closed types has to be local, i.e. for every object $A \in \mathsf{Ob}(\mathbb{C})$ the slice category $\mathbb{C}/A$ must enjoy the same structure of $\mathbb{C}$ (for example, if $\mathbb{C}$ is a regular category then $\mathbb{C}/A$ should be a regular category for every $A \in \mathsf{Ob}(\mathbb{C})$). This is because a dependent type is interpreted in a suitable slice category and hence any slice category has to be equipped with all the structure to interpret the type constructors under a certain dependency. The slice category is defined as

   Definition 2.0.6 (Slice category)**.** *The slice category $\mathbb{C}/A$ of a category $\mathbb{C}$ over an object $A \in \mathsf{Ob}(\mathbb{C})$ has*

   - *objects that are all arrows $f \in \mathsf{Arr}(\mathbb{C})$ such that $cod(f) = A$*
   - *morphisms $h : X \to Y \in \mathsf{Arr}(\mathbb{C})$ from $f : X \to A$ to $g : Y \to A$ such that $g \circ h = f$.*

3. The structure of $\mathbb{C}$ has to be preserved by the pullback functor $F^* : \mathbb{C}/A \to \mathbb{C}/B$ for every morphism $f : B \to A$ of $\mathbb{C}$. This is because, if we interpret substitution via pullback, then the structure needs to be preserved under pullbacks to make the interpretation of a type constructor closed under substitution.

Regular categories satisfy the necessary conditions to enjoy an internal dependent type theory.

3

The internal dependent type theory of regular categories $\mathcal{T}_{reg}$

3.1 Extensional dependent type theories

For the categorical structure of the regular categories described in the previous section we give here the description of the corresponding typed calculus meant to provide its internal language in the style of Martin-Löf's extensional dependent type theory [1].

Any typed system is equipped with types, which should be thought of as sets or data types, and with typed terms which represent proofs (or elements) of the types to which they belong. In order to describe them in the style of Martin-Löf's type theory, we have four kinds of judgements [6]:

$$A\ type\ [\Gamma] \quad A = B\ [\Gamma] \quad a \in A\ [\Gamma] \quad a = b \in A\ [\Gamma]$$

that is the judgements about type formation and their terms, the equality between types and the equality between terms of the same type (called definitional equality of terms in contrast to the propositional equality of terms that is a type).

The contexts of these judgements are telescopic [7], since types are allowed to depend on variables of other types. The contexts are generated by the following rules

$$\frac{}{\emptyset\ cont}\ \text{1C} \qquad \frac{\Gamma\ cont \qquad A\ type\ [\Gamma]}{\Gamma, x \in A\ cont}\ \text{2C}\ (x \in A \notin \Gamma)$$

When the context is empty we write only F instead of $F[]$, and call J a "closed" judgement as opposed to "hypothetical" judgement, that is with non-empty context.

Then, we need to add all the inference rules that express reflexivity, symmetry and transitivity of the equality between types and terms together with the type equality rules and the assumption of variables

Reflexivity

$$\frac{a \in A\,[\Gamma]}{a = a \in A\,[\Gamma]}\ \text{refl-tm} \qquad \frac{A\ \mathit{type}\,[\Gamma]}{A = A\,[\Gamma]}\ \text{refl-ty}$$

Symmetry

$$\frac{a = b \in A\,[\Gamma]}{b = a \in A\,[\Gamma]}\ \text{sym-tm} \qquad \frac{A = B\,[\Gamma]}{B = A\,[\Gamma]}\ \text{sym-ty}$$

Transitivity

$$\frac{a = b \in A\,[\Gamma] \qquad b = c \in A\,[\Gamma]}{a = c \in A\,[\Gamma]}\ \text{trans-tm} \qquad \frac{A = B\,[\Gamma] \qquad B = C\,[\Gamma]}{A = C\,[\Gamma]}\ \text{trans-ty}$$

Type equality

$$\frac{a \in A\,[\Gamma] \qquad A = B\,[\Gamma]}{a \in B\,[\Gamma]}\ \text{conv} \qquad \frac{a = b \in A\,[\Gamma] \qquad A = B\,[\Gamma]}{a = b \in B\,[\Gamma]}\ \text{conv-eq}$$

Assumption

$$\frac{\Gamma, x \in A, \Delta\ \mathit{cont}}{x \in A\,[\Gamma, x \in A, \Delta]}\ \text{var}$$

Generally, when formulating a theory of types, it is convenient to use the minimum number possible of structural rules and formation of types and terms in a way such, however, that the theory is closed on some inalienable rules such as the weakening rules the substitution rules and suitable exchange rules. Thanks to the form of the rules we will provide for building types and terms it will not be necessary explicitly to add those which we list below because their are admissible. We recall the following definition of admissible rule:

Definition 3.1.1 (Admissible rule)**.** *Let's say that a rule formulated with the judgments of $\mathcal{T}_{reg}$ theory is admissible in $\mathcal{T}_{reg}$ if and only if in the case its premise judgements are derivable in $\mathcal{T}_{reg}$ then also the conclusion judgement is also derivable in T.*

Weakening

$$\frac{\mathcal{A}\ \mathit{type}\ [\Gamma] \qquad \Gamma, \Delta\ \mathit{cont}}{\mathcal{A}\ \mathit{type}\ [\Gamma, \Delta]}\ \text{ind-ty} \qquad \frac{\mathcal{A} = B\ [\Gamma] \qquad \Gamma, \Delta\ \mathit{cont}}{\mathcal{A} = B\ [\Gamma, \Delta]}\ \text{ind-ty-eq}$$

$$\frac{a \in \mathcal{A}\ [\Gamma] \qquad \Gamma, \Delta\ \mathit{cont}}{a \in \mathcal{A}\ [\Gamma, \Delta]}\ \text{ind-tm} \qquad \frac{a = b \in \mathcal{A}\ [\Gamma] \qquad \Gamma, \Delta\ \mathit{cont}}{a = b \in \mathcal{A}\ [\Gamma, \Delta]}\ \text{ind-tm-eq}$$

Substitution

In a calculus of dependent types we must necessarily substitute terms in types, terms and their equality when these are dependent on certain other types. Therefore the calculation must be closed on the following substitution rules:

$$\frac{a_1 \in A_1\ [\Gamma] \ldots a_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad C\,(x_1, \ldots, x_n)\ \mathit{type}\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_{n-1})]}{C\,(a_1, \ldots, a_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-ty}$$

$$\frac{a_1 = b_1 \in A_1\ [\Gamma] \ldots a_n = b_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad C\,(x_1, \ldots, x_n)\ \mathit{type}\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_{n-1})]}{C\,(a_1, \ldots, a_n) = C\,(b_1, \ldots, b_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-ty-eq}$$

$$\frac{a_1 \in A_1\ [\Gamma] \ldots a_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad C\,(x_1, \ldots, x_n) = D\,(x_1, \ldots, x_n)\ \mathit{type}\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n\,(x_1, \ldots, x_{n-1})]}{C\,(a_1, \ldots, a_n) = D\,(a_1, \ldots, a_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-Eq-ty}$$

$$\frac{a_1 = b_1 \in A_1\ [\Gamma] \ldots a_n = b_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad C\,(x_1, \ldots, x_n) = D\,(x_1, \ldots, x_n)\ \mathit{type}\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n\,(x_1, \ldots, x_{n-1})]}{C\,(a_1, \ldots, a_n) = D\,(b_1, \ldots, b_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-Eq-ty-eq}$$

$$\frac{a_1 \in A_1\ [\Gamma] \ldots a_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad c\,(x_1, \ldots, x_n) \in C\,(x_1, \ldots, x_n)\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_{n-1})]}{c\,(a_1, \ldots, a_n) \in C\,(a_1, \ldots, a_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-tm}$$

$$\frac{a_1 = b_1 \in A_1\ [\Gamma] \ldots a_n = b_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad c\,(x_1, \ldots, x_n) \in C\,(x_1, \ldots, x_n)\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_{n-1})]}{c\,(a_1, \ldots, a_n) = c\,(b_1, \ldots, b_n) \in C\,(a_1, \ldots, a_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-tm-eq}$$

$$\frac{a_1 \in A_1\ [\Gamma] \ldots a_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad c\,(x_1, \ldots, x_n) = d\,(x_1, \ldots, x_n) \in C\,(x_1, \ldots, x_n)\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_{n-1})]}{c\,(a_1, \ldots, a_n) = d\,(a_1, \ldots, a_n) \in C\,(a_1, \ldots, a_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-eq-tm}$$

$$\frac{a_1 = b_1 \in A_1\ [\Gamma] \ldots a_n = b_n \in A_n\,(a_1, \ldots, a_{n-1})\ [\Gamma] \qquad c\,(x_1, \ldots, x_n) = d\,(x_1, \ldots, x_n) \in C\,(x_1, \ldots, x_n)\ [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_{n-1})]}{c\,(a_1, \ldots, a_n) = d\,(b_1, \ldots, b_n) \in C\,(a_1, \ldots, a_n)\ \mathit{type}\ [\Gamma]}\ \text{sub-eq-tm}$$

Exchange

The rule of exchange of assumptions in a context in type theory dependent is not generally derivable for the dependence of the type of an assumption in a Γ context from the assumptions in Γ that precede it. However, we can demonstrate a restricted form of exchange rule such as follows

$$\frac{A\ \mathit{type}\ [\Gamma, x \in C, y \in D, \Delta] \qquad \Gamma, y \in D, x \in C, \Delta\ \mathit{cont}}{A\ \mathit{type}\ [\Gamma, y \in D, x \in C, \Delta]}\ \text{ex-ty}$$

$$\frac{A = B\ [\Gamma, x \in C, y \in D, \Delta] \qquad \Gamma, y \in D, x \in C, \Delta\ \mathit{cont}}{A = B\ [\Gamma, y \in D, x \in C, \Delta]}\ \text{ex-ty-eq}$$

$$\frac{a \in A\ [\Gamma, x \in C, y \in D, \Delta] \qquad \Gamma, y \in D, x \in C, \Delta\ \mathit{cont}}{a \in A\ [\Gamma, y \in D, x \in C, \Delta]}\ \text{ex-tm}$$

$$\frac{a = b \in A\ [\Gamma, x \in C, y \in D, \Delta] \qquad \Gamma, y \in D, x \in C, \Delta\ \mathit{cont}}{a = b \in A\ [\Gamma, y \in D, x \in C, \Delta]}\ \text{ex-tm-eq}$$

We also adopt the usual definitions of bound and free occurrences of variables and we identify two terms under α-conversion.

3.2 The calculus of regular categories $\mathcal{T}_{reg}$

Now, we give the formation rule for types specific to the calculus of regular categories with the introduction, elimination and conversion rules of their terms.

3.2.1 Terminal type

Formation

$$\frac{\Gamma\ \mathit{cont}}{\top\ \mathit{type}\ [\Gamma]}\ \text{Tr}$$

Introduction

$$\frac{\Gamma\ \mathit{cont}}{\star \in \top\ [\Gamma]}\ \text{I-Tr}$$

Equality

$$\frac{t \in \top\ [\Gamma]}{t = \star \in \top\ [\Gamma]}\ \text{C-Tr}$$

Computation

$$\top \Rrightarrow \top$$

$$\star \Rightarrow \star$$

3.2.2 Indexed Sum type

Formation

$$\frac{C(x)\ \mathit{type}\ [\Gamma, x \in B]}{\Sigma_{x\in B} C(x)\ \mathit{type}\ [\Gamma]}\ \Sigma \qquad \frac{C(x) = E(x)\ [\Gamma, x \in B]}{\Sigma_{x\in B} C(x) = \Sigma_{x\in D} E(x)\ [\Gamma]}\ \Sigma\text{-eq}$$

Introduction

$$\frac{b \in B\ [\Gamma] \qquad c \in C(b)\ [\Gamma] \qquad \Sigma_{x\in B} C(x)\ \mathit{type}\ [\Gamma]}{\langle b, c\rangle \in \Sigma_{x\in B} C(x)\ [\Gamma]}\ \text{I-}\Sigma$$

$$\frac{b = d \in B\ [\Gamma] \qquad c = e \in C(b)\ [\Gamma] \qquad \Sigma_{x\in B} C(x)\ \mathit{type}\ [\Gamma]}{\langle b, c\rangle = \langle d, e\rangle \in \Sigma_{x\in B} C(x)\ [\Gamma]}\ \text{I-}\Sigma\text{-eq}$$

Elimination

$$\frac{M(z)\ \mathit{type}\ [\Gamma, z \in \Sigma_{x\in B} C(x)] \qquad d \in \Sigma_{x\in B} C(x)\ [\Gamma] \qquad m(x,y) \in M(\langle x, y\rangle)\ [\Gamma, x \in B, y \in C(x)]}{El_\Sigma(d, m) \in M(d)\ [\Gamma]}\ \text{E-}\Sigma$$

$$\frac{M(z)\ \mathit{type}\ [\Gamma, z \in \Sigma_{x\in B} C(x)] \qquad d = d' \in \Sigma_{x\in B} C(x)\ [\Gamma] \qquad m(x,y) = m'(x,y) \in M(\langle x, y\rangle)\ [\Gamma, x \in B, y \in C(x)]}{El_\Sigma(d, m) = El_\Sigma(d', m') \in M(d)\ [\Gamma]}\ \text{E-}\Sigma\text{-eq}$$

Equality

$$\frac{M(z)\ \mathit{type}\ [\Gamma, z \in \Sigma_{x\in B} C(x)] \qquad b \in B\ [\Gamma] \qquad c \in C(b)\ [\Gamma] \qquad m(x,y) \in M(\langle x, y\rangle)\ [\Gamma, x \in B, y \in C(x)]}{El_\Sigma(\langle b, c\rangle, m) = m(b, c) \in M(\langle b, c\rangle)\ [\Gamma]}\ \text{C-}\Sigma$$

Computation

$$\Sigma_{x\in B}C(x) \Rrightarrow \Sigma_{x\in B}C(x)$$

$$\langle b, c\rangle \Rightarrow \langle b, c\rangle \qquad \frac{d \Rightarrow \langle b, c\rangle \qquad m(b,c) \Rightarrow g}{El_{\Sigma}(d, m) \Rightarrow g}$$

3.2.3 Extensional Equality type

Formation

$$\frac{C\ type\ [\Gamma] \qquad c \in C\,[\Gamma] \qquad d \in C\,[\Gamma]}{\mathsf{Eq}(C, c, d)\ type\ [\Gamma]}\ \text{Eq}$$

$$\frac{C = E\,[\Gamma] \qquad c = e \in C\,[\Gamma] \qquad d = f \in C\,[\Gamma]}{\mathsf{Eq}(C, c, d) = \mathsf{Eq}(E, e, f)\ [\Gamma]}\ \text{Eq-eq}$$

Introduction

$$\frac{c \in C\,[\Gamma]}{\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)\ [\Gamma]}\ \text{I-Eq} \qquad \frac{c = d \in C\,[\Gamma]}{\mathsf{eq}_C(c) = \mathsf{eq}_C(d) \in \mathsf{Eq}(C, c, c)\ [\Gamma]}\ \text{I-Eq-eq}$$

Elimination

$$\frac{p \in \mathsf{Eq}(C, c, d)\ [\Gamma]}{c = d \in C\,[\Gamma]}\ \text{E-Eq}$$

Equality

$$\frac{p \in \mathsf{Eq}(C, c, d)\ [\Gamma]}{p = \mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, d)\ [\Gamma]}\ \text{C-Eq}$$

Computation

$$\mathsf{Eq}(C, c, d) \Rrightarrow \mathsf{Eq}(C, c, d)$$

$$\mathsf{eq}_C(c) \Rightarrow \mathsf{eq}_C(c)$$

3.2.4 Quotient types on the terminal type

Formation

$$\frac{A\ type\ [\Gamma]}{A/\top\ type\ [\Gamma]}\ \text{Qtr} \qquad \frac{A = B\ [\Gamma]}{A/\top = B/\top\ [\Gamma]}\ \text{Qtr-eq}$$

Introduction

$$\frac{a \in A\ [\Gamma]}{[a] \in A/\top\ [\Gamma]}\ \text{I-Qtr} \qquad \frac{a \in A\ [\Gamma] \qquad b \in A\ [\Gamma]}{[a] = [b] \in A/\top\ [\Gamma]}\ \text{eq-Qtr}$$

Elimination

$$\frac{L(z)\ type\ [\Gamma, z \in A/\top] \qquad p \in A/\top\ [\Gamma] \qquad l(x) \in L([x])\ [\Gamma, x \in A] \qquad l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A]}{El_Q(l,p) \in L(p)\ [\Gamma]}\ \text{E-Qtr}$$

$$\frac{L(z)\ type\ [\Gamma, z \in A/\top] \qquad p = p' \in A/\top\ [\Gamma] \qquad l(x) = l'(x) \in L([x])\ [\Gamma, x \in A] \qquad l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A]}{El_Q(l,p) = El_Q(l',p') \in L(p)\ [\Gamma]}\ \text{E-Qtr-eq}$$

Equality

$$\frac{L(z)\ type\ [\Gamma, z \in A/\top] \qquad a \in A\ [\Gamma] \qquad l(x) \in L([x])\ [\Gamma, x \in A] \qquad l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A]}{El_Q(l,[a]) = l(a) \in L([a])\ [\Gamma]}\ \text{C-Qtr}$$

Computation

$$A/\top \Rrightarrow A/\top$$

$$[a] \Rightarrow [a] \qquad \frac{p \Rightarrow [a] \qquad l(a) \Rightarrow g}{El_Q(l,p) \Rightarrow g}$$

We'll refer to the calculus of regular categories with the notation $\mathcal{T}_{reg}$

3.2.5 Properties

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

Definition 3.2.1 (Proof). *A proof is a tree of judgements built up in the usual way using instances of the rules of inference.*

Lemma 3.2.1 (Sanitary checks rules on typing correctness). *In a type theory they must also hold the following properties of sanitary check to ensure correctness of the typing of terms:*

1. *If* $[\Gamma, \Delta]$ *cont is derivable then* $[\Gamma]$ *cont is also derivable;*
2. *If* $[\Gamma, x \in A]$ *cont is derivable then* $[\Gamma]$ *cont and* A *type* $[\Gamma]$ *are also derivable;*
3. *If* A *type* $[\Gamma]$ *is derivable then* $[\Gamma]$ *cont is also derivable;*
4. *If* $a \in A$ $[\Gamma]$ *is derivable then* A *type* $[\Gamma]$ *is also derivable;*
5. *If* $A = B$ *type* $[\Gamma]$ *is derivable then so be it* A *type* $[\Gamma]$ *and* B *type* $[\Gamma]$ *are derivable;*
6. *If* $a = b \in A$ $[\Gamma]$ *is derivable then so be it* $a \in A$ $[\Gamma]$ *and* $b \in A$ $[\Gamma]$ *are derivable.*

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 3.2.2 (Fitting substitution). *The sequences of judgements*

$$a_1 \in \underline{A_1}[\Gamma], \ldots, a_n \in \underline{A_n}\ [\Gamma]$$

and

$$a_1 = a_1' \in \underline{A_1}[\Gamma], \ldots, a_n = a_n' \in \underline{A_n}\ [\Gamma]$$

where

$$\underline{A_i} \equiv A_i(a_1, \ldots, a_{i-1})$$

are substitutions that fit with any context $[\Gamma, x_1 \in A_1, \ldots, x_n \in A_n(x_1, \ldots, x_n)]$.

We introduce the notion of associate judgements.

Definition 3.2.3 (Associate judgements). *The associate judgement(s) of*

1. $a \in A\,[\Gamma]$ *is* A *type* $[\Gamma]$
2. $A = B\,[\Gamma]$ *are* A *type* $[\Gamma]$ *and* B *type* $[\Gamma]$
3. $a = b \in A\,[\Gamma]$ *are* $a \in A\,[\Gamma]$ *and* $b \in A\,[\Gamma]$

Theorem 3.2.1 (Associate judgements derivability)**.** *Let J be a derivable judgement. Then the associate judgements of J are derivable.*

Proof. The three cases should be proved simultaneously. 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. □

Substituted judgements

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

Definition 3.2.4 (Tail substituted judgements)**.** *Let $\Delta \equiv [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n]$ be a context, $a_1 \in \underline{A}_1\,[\Gamma], \ldots, a_n \in \underline{A}_n\,[\Gamma]$ and $a_1 = a'_1 \in \underline{A}_1\,[\Gamma], \ldots, a_n = a'_n \in \underline{A}_n\,[\Gamma]$ be substitutions that fit with the last n assumption in the context Δ, and $J \equiv F\,[\Delta]$ be any judgement. Then*

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

   (a) *If $F \equiv A$ type*
   $$A(a_1, \ldots, a_n) \text{ type } [\Gamma]$$

   (b) *If $F \equiv A = B$*
   $$A(a_1, \ldots, a_n) = B(a_1, \ldots, a_n)\,[\Gamma]$$

   (c) *If $F \equiv a \in A$*
   $$a(a_1, \ldots, a_n) \in A(a_1, \ldots, a_n)\,[\Gamma]$$

   (d) *If $F \equiv a = b \in A$*
   $$a(a_1, \ldots, a_n) = b(a_1, \ldots, a_n) \in A(a_1, \ldots, a_n)\,[\Gamma]$$

2. *$J[x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$ is an abbreviation for the tail substituted judgement of J which is the following:*

(a) *If* $F \equiv A$ *type*

$$A(a_1, \ldots, a_n) = A(a'_1, \ldots, a'_n)\ [\Gamma]$$

(b) *If* $F \equiv A = B$

$$A(a_1, \ldots, a_n) = B(a'_1, \ldots, a'_n)\ [\Gamma]$$

(c) *If* $F \equiv a \in A$

$$a(a_1, \ldots, a_n) = a(a'_1, \ldots, a'_n) \in A(a_1, \ldots, a_n)\ [\Gamma]$$

(d) *If* $F \equiv a = b \in A$

$$a(a_1, \ldots, a_n) = b(a'_1, \ldots, a'_n) \in A(a_1, \ldots, a_n)\ [\Gamma]$$

The substitutions rules are sufficient to prove the following theorem

Lemma 3.2.2. *The tail substituted judgements of a derivable judgement are derivable.*

Proof. Just apply the suitable substitution rule. □

Definition 3.2.5 (Head substituted judgements)**.** *Let* $\Delta \equiv [x_1 : A_1, \ldots, x_n : A_n]$ *be a context,* $a_1 \in A_1, \ldots, a_i \in A_i$ *and* $a_1 = a'_1 \in A_1, \ldots, a_i = a'_i \in A_i$ *for some* $i \leq n$*, be substitutions that fit with the first* i *assumptions of* Δ*,* $J \equiv F\,[\Delta]$ *be any judgement. Moreover let* $A'_j \equiv A_j(a_1, \ldots, a_i)$ *for any* $i + 1 \leq j \leq n$*. Then*

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

   (a) *If* $F \equiv A$ *type*

   $$A(a_1, \ldots, a_i)\ \textit{type}\ [x_{i+1} \in A'_{i+1}, \ldots, x_n \in A'_n]$$

   (b) *If* $F \equiv A = B$

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

   (c) *If* $F \equiv a \in A$

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

(d) If $F \equiv a = b \in A$

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

2. $J[x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$ *is an abbreviation for the tail substituted judgement of* J *which is the following:*

   (a) If $F \equiv A$ *type*

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

   (b) If $F \equiv A = B$

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

   (c) If $F \equiv a \in A$

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

   (d) If $F \equiv a = b \in A$

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

Lemma 3.2.3. *The head substituted judgements of a derivable judgement are derivable.*

Proof. 1. Note that if $a_1 \in A_1, \ldots, a_i \in A_i$ is a substitution that fits with $[x_1 \in A_1, \ldots, x_i \in A_i]$ then $a_1 \in A_1, \ldots, a_i \in A_i, x_{i+1} \in A_{i+1}, \ldots, x_n \in A_n$ is a substitution that fits with $[x_1 \in A_1, \ldots, x_n \in A_n]$. Hence the result follows by using the suitable substitution rule.

2. If $a_1 = a'_1 \in A_1, \ldots, a_i = a'_i \in A_i$ is a substitution that fits with $[x_1 \in A_1, \ldots, x_i \in A_i]$ then $a_1 = a'_1 \in A_1, \ldots, a_i = a'_i \in A_i, x_{i+1} = x_{i+1} \in A_{i+1}, \ldots, x_n = x_n \in A_n$ is a substitution that fits with $[x_1 \in A_1, \ldots, x_n \in A_n]$. Hence the result follows by using directly the suitable substitution rule.

□

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.

3.2.6 Alternative version

In order to prove the canonical normal form for $\mathcal{T}_{reg}$, we'll consider an equivalent version of the theory, named $\mathcal{T}^*_{reg}$, where the Σ-formation rules and the var rule are replaced by the following rules:

$$\frac{B\ type\ [\Gamma] \qquad C(x)\ type\ [\Gamma, x \in B]}{\Sigma_{x\in B}C(x)\ type\ [\Gamma]}\ \Sigma^* \qquad \frac{B = D\ [\Gamma] \qquad C(x) = E(x)\ [\Gamma, x \in B]}{\Sigma_{x\in B}C(x) = \Sigma_{x\in D}E(x)\ [\Gamma]}\ \Sigma^*\text{-eq}$$

$$\frac{p \in \mathsf{Eq}(C, c, d)\ [\Gamma] \qquad C\ type\ [\Gamma] \qquad c \in C\ [\Gamma] \qquad d \in C\ [\Gamma]}{c = d \in C\ [\Gamma]}\ \text{E-Eq}^*$$

$$\frac{\Gamma, x \in A, \Delta\ cont \qquad A\ type\ [\Gamma]}{x \in A\ [\Gamma, x \in A, \Delta]}\ \text{var}^*$$

The equivalence is guaranteed by the following lemma.

Lemma 3.2.4. *$\mathcal{T}_{reg}$ and $\mathcal{T}^*_{reg}$ are equivalent, i.e.*

1. *$\mathcal{T}_{reg}$ derives the Σ^*-formation and the var* rules*
2. *$\mathcal{T}^*_{reg}$ derives the Σ-formation and the var rules*

Proof. 1. Σ^* can be derived using the Σ rule, Σ^*-eq can be derived using the Σ-eq rule, E-Eq* can be derived using the E-Eq rule and var* can be derived using the var rule.

2. To derive the Σ rule, consider the derivable judgement $C(x)\ type\ [\Gamma, x \in B]$, then $[\Gamma, x \in B]\ cont$ is derivable and so is $B\ type\ [\Gamma]$. The result follows by applying the Σ^* rule. Similarly for the Σ-eq rule.

   To derive the E-Eq rule, consider the derivable judgement $p \in \mathsf{Eq}(C, c, d)\ [\Gamma]$, then $\mathsf{Eq}(C, c, d)\ type\ [\Gamma]$ is derivable and so are $C\ type\ [\Gamma]$, $c \in C\ [\Gamma]$ and $d \in C\ [\Gamma]$. The result follows by applying the E-Eq* rule.

   To derive the var rule, consider the derivable judgement $\Gamma, x \in A, \Delta\ cont$, then, by point 1 of lemma 3.2.1, $\Gamma, x \in A\ cont$ is derivable, and so, by point 2 of the same lemma, also $A\ type\ [\Gamma]$. Hence, the result follows by applying the var* rule.

□

4

The internal dependent type theory of arithmetic regular categories $\mathcal{T}_{areg}$

In this chapter, we extend the theory of regular categories by adding natural numbers. This allows us to reason about recursive functions and their properties in a more precise and systematic way. Here, we show the added rules of the natural numbers type.

Natural numbers type

Formation

$$\frac{\Gamma\ cont}{N\ type\ [\Gamma]}\ \text{nat}$$

Introduction

$$\frac{\Gamma\ cont}{0 \in N\,[\Gamma]}\ \text{I}_1\text{-nat} \qquad \frac{n \in N\,[\Gamma]}{\mathsf{s}(n) \in N\,[\Gamma]}\ \text{I}_2\text{-nat} \qquad \frac{m = n \in N\,[\Gamma]}{\mathsf{s}(m) = \mathsf{s}(n) \in N\,[\Gamma]}\ \text{I}_2\text{-nat-eq}$$

Elimination

$$\frac{L(z)\ type\ [\Gamma, z \in N] \qquad n \in N\,[\Gamma] \qquad a \in L(0)\ [\Gamma] \qquad l(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]}{El_N(a,l,n) \in L(n)\ [\Gamma]}\ \text{E-nat}$$

$$\frac{L(z)\ \mathit{type}\ [\Gamma, z \in N] \qquad n = n' \in N\,[\Gamma] \qquad a = a' \in L(0)\ [\Gamma] \qquad l(x,y) = l'(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]}{El_N(a,l,n) = El_N(a',l',n') \in L(n)\ [\Gamma]}\ \text{E-nat-eq}$$

Equality

$$\frac{L(z)\ \mathit{type}\ [\Gamma, z \in N] \qquad a \in L(0)\ [\Gamma] \qquad l(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]}{El_N(a,l,0) = a \in L(0)\ [\Gamma]}\ \text{C}_1\text{-nat}$$

$$\frac{L(z)\ \mathit{type}\ [\Gamma, z \in N] \qquad n \in N\,[\Gamma] \qquad a \in L(0)\ [\Gamma] \qquad l(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]}{El_N(a,l,\mathsf{s}(n)) = l(n, El_N(a,l,n)) \in L(\mathsf{s}(n))\ [\Gamma]}\ \text{C}_2\text{-nat}$$

Computation

$$N \Rrightarrow N$$

$$0 \Rightarrow 0 \qquad \frac{n \Rightarrow 0 \qquad a \Rightarrow g}{El_N(a,l,n) \Rightarrow g}$$

$$\mathsf{s}(m) \Rightarrow \mathsf{s}(m) \qquad \frac{n \Rightarrow \mathsf{s}(m) \qquad l(m, El_N(a,l,m)) \Rightarrow g}{El_N(a,l,n) \Rightarrow g}$$

5

The canonical form theorem for $\mathcal{T}_{reg}$

5.1 The evaluation tree

A set of computation rules is associated to each defined type. They specify a process for evaluating expressions denoting elements or types. They apply to variable-free and saturated expressions, that is, expressions of arity 0 in which no variable occurs free. The "normal form" theorem for expressions [8], assures us that a variable-free, saturated expression is always definitionally equivalent to an expression of the form $c\,(a_1, \ldots, a_n)$ where 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 to continue the process recursively. 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 theorem.

Theorem 5.1.1. *Let c and C be variable-free and saturated expressions. Then*

1. *If $c \Rightarrow g$ then g is a canonical expression for an element, i.e. exactly one of the following holds: $g \equiv \star, g \equiv \langle a, b\rangle, g \equiv \mathsf{eq}_A(a), g \equiv [a]$.*

2. *If $C \Rightarrow G$ then G is a canonical expression for a type, i.e. exactly one of the following holds:* $G \equiv \top, G \equiv \Sigma_{x \in A} B(x), G \equiv \mathsf{Eq}(A, a, b), G \equiv A/\top$.

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 canonical form theorem will certify this whenever we consider judgements derived within the theory. More precisely, if the judgement $a \in A$ (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 $a \Rightarrow g$ (respectively $A \Rightarrow G$), then no variable appears in g (respectively G).

5.2 Computability

In this section we introduce the main notion of the chapter: the definition of computable judgement.

To prove a canonical-form theorem for the system we are considering we will follow a proof style similar to the one used by MartinLöf in [9] based on the method of Tait [10] 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 [1], but the needs of formalization make it a very long definition.

Definition 5.2.1 (Computable judgement)**.** *The judgement $J \equiv F\,[\Gamma]$ is computable if it is derivable and*

1. *There is no assumption, i.e. the context Γ is empty.*

- *Subcase 1.1:* $F \equiv A$ *type. Then*
  - *(evaluation)* $A \Rrightarrow G_A$
  - *(correct evaluation) the judgement* $A = G_A$ *is derivable*
  - *(parts) the parts of* G_A *are computable type(s), i.e.*
    - *if* $G_A \equiv \top$ *then no condition*
    - *if* $G_A \equiv \Sigma_{x \in C_1} C_2(x)$ *then the judgements* C_1 *type and* $C_2(x)$ *type* $[x \in C_1]$ *are computable*
    - *if* $G_A \equiv \mathsf{Eq}\,(C, c_1, c_2)$ *then the judgements* C *type,* $c_1 \in C$ *and* $c_2 \in C$ *are computable*
    - *if* $G_A \equiv C/\top$ *then the judgement* C *type is computable*

- *Subcase 1.2:* $F \equiv A = B$ *then*
  - *(associate judgements) the associate judgements* A *type and* B *type are computable, and hence* $A \Rrightarrow G_A$ *and* $B \Rrightarrow G_B$.
  - *(parts)* G_A *and* G_B *are equal computable types, i.e.*
    - $G_A \equiv \top$ *iff* $G_B \equiv \top$
    - $G_A \equiv \Sigma_{x \in C_1} C_2(x)$ *iff* $G_B \equiv \Sigma_{x \in D_1} D_2(x)$ *and the judgements* $C_1 = D_1$ *and* $C_2(x) = D_2(x)$ $[x \in C_1]$ *are computable*
    - $G_A \equiv \mathsf{Eq}\,(C, c_1, c_2)$ *iff* $G_B \equiv \mathsf{Eq}\,(D, d_1, d_2)$ *and the judgements* $C = D$, $c_1 = d_1 \in C$ *and* $c_2 = d_2 \in C$ *are computable*
    - $G_A \equiv C/\top$ *iff* $G_B \equiv D/\top$ *and the judgement* $C = D$ *is computable*

- *Subcase 1.3:* $F \equiv a \in A$ *then*
  - *(associate judgements) The associate judgement* A *type is computable, and hence* $A \Rrightarrow G_A$
  - *(evaluation)* $a \Rightarrow g$
  - *(correct evaluation)* $a = g \in A$ *is provable*
  - *(parts) the parts of* g *are computable element(s) in* G_A*, i.e.*
    - $G_A \equiv \top$ *iff* $g \equiv \star$
    - $G_A \equiv \Sigma_{x \in C} D(x)$ *iff* $g \equiv \langle c, d \rangle$ *and the judgements* $c \in C$ *and* $d \in D(c)$ *are computable*
    - $G_A \equiv \mathsf{Eq}\,(C, c_1, c_2)$ *iff* $g \equiv \mathsf{eq}_C(c_3)$ *and the judgement* $c_1 = c_2 \in C$ *is computable*

* $G_A \equiv C/\top$ *iff* $g \equiv [c]$ *and the judgement* $c \in C$ *is computable*

- *Subcase 1.4:* $F \equiv a = b \in A$ *then*
  - *(associate judgements) the associate judgements* $a \in A$ *and* $b \in A$ *are computable, and hence* $a \Rightarrow g_a$, $b \Rightarrow g_b$ *and* $A \Rrightarrow G_A$.
  - *(parts) the following holds*
    * $G_A \equiv \top$ *iff* $g_a \equiv \star$ *and* $g_b \equiv \star$
    * $G_A \equiv \Sigma_{x \in C} D(x)$ *iff* $g_a \equiv \langle c_1, d_1 \rangle$ *and* $g_b \equiv \langle c_2, d_2 \rangle$ *and the judgements* $c_1 = c_2 \in C$ *and* $d_1 = d_2 \in D(c_1)$ *are computable*
    * $G_A \equiv \mathsf{Eq}(C, c_1, c_2)$ *iff* $g_a \equiv \mathsf{eq}_C(c_3)$ *and* $g_b \equiv \mathsf{eq}_C(c_4)$ *and the judgement* $c_1 = c_2 \in C$ *is computable*
    * $G_A \equiv C/\top$ *iff* $g_a \equiv [c_1]$ *and* $g_b \equiv [c_2]$ *and the judgements* $c_1 \in C$ *and* $c_2 \in C$ *are computable*

2. *There are assumptions, i.e.* $\Gamma \equiv x_1 \in A_1, \ldots, x_n \in A_n$, *for some* $n > 0$. *The judgement* J *is computable if for any computable closed substitution (c.c.s.)* $a_1 \in A_1, \ldots, a_n \in A_n$ *(i.e.* $a_i \in A_i$, *for* $1 \leq i \leq n$, *are computable judgements), and for any computable closed substitution (c.c.s.)* $a_1 = c_1 \in A_1, \ldots, a_n = c_n \in A_n$ *(i.e.* $a_i = c_i \in A_i$, *for* $1 \leq i \leq n$, *are computable judgements) that fit with* Γ

   - *Subcase 2.1:* $F \equiv B(x_1, \ldots, x_n)$ *type*
     - *(substitution* $:=$*) the judgement* $B(a_1, \ldots, a_n)$ *type is computable*
     - *(substitution* $\leftarrow$*) the judgement* $B(a_1, \ldots, a_n) = B(c_1, \ldots, c_n)$ *is computable*

   - *Subcase 2.2:* $F \equiv B(x_1, \ldots, x_n) = D(x_1, \ldots, x_n)$ *then*
     - *(associate) the judgement* $B(x_1, \ldots, x_n)$ *type* $[\Gamma]$ *is computable*
     - *(substitution* $:=$*) the judgement* $B(a_1, \ldots, a_n) = D(a_1, \ldots, a_n)$ *is computable*
     - *(substitution* $\leftarrow$*) the judgement* $B(a_1, \ldots, a_n) = D(c_1, \ldots, c_n)$ *is computable*

   - *Subcase 2.3:* $F \equiv b(x_1, \ldots, x_n) \in B(x_1, \ldots, x_n)$ *then*
     - *(associate) the judgement* $B(x_1, \ldots, x_n)$ *type* $[\Gamma]$ *is computable*

- *(substitution :=) the judgement* $b\,(a_1, \ldots, a_n) \in B\,(a_1, \ldots, a_n)$ *is computable*
- *(substitution* $\leftarrow$*) the judgement* $b\,(a_1, \ldots, a_n) = b\,(c_1, \ldots, c_n) \in B\,(a_1, \ldots, a_n)$ *is computable*

- *Subcase 2.4:* $F \equiv b\,(x_1, \ldots, x_n) = d\,(x_1, \ldots, x_n) \in B\,(x_1, \ldots, x_n)$ *then*
  - *(associate) the judgement* $b\,(x_1, \ldots, x_n) \in B\,(x_1, \ldots, x_n)\,[\Gamma]$ *is computable*
  - *(substitution :=) the judgement* $b\,(a_1, \ldots, a_n) = d\,(a_1, \ldots, a_n) \in B\,(a_1, \ldots, a_n)$ *is computable*
  - *(substitution* $\leftarrow$*) the judgement* $b\,(a_1, \ldots, a_n) = d\,(c_1, \ldots, c_n) \in B\,(a_1, \ldots, a_n)$ *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 next 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.

5.3 Computability of the rules

We are now going to prove that any judgement derivable in the theory is computable. The proof will consist in proving that each rule preserves computability, that is, if the judgements in the premises 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. 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.

5.3.1 The structural rules

Lemma 5.3.1 (Weakening rules)**.** *If $F[\Gamma]$ is computable then $F[\Gamma, \Delta]$ is computable.*

Proof. Let $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$ and $\Delta \equiv [y_1 \in B_1, \ldots, y_m \in B_m]$.

- (substitution $:=$) For any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n, b_1 \in B_1, \ldots, b_m \in B_m$ fitting with $[\Gamma, \Delta]$, we have that $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. that fits with Γ; hence

  $$F[\Gamma]\,[x_1 := a_1, \ldots, x_n := a_n] \equiv F[\Gamma, \Delta]\,[x_1 := a_1, \ldots, x_n := a_n, y_1 := b_1, \ldots, y_m := b_m]$$

  is computable by assumption.

- (substitution $\leftarrow$) For any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n, b_1 = b_1' \in B_1, \ldots, b_m = b_m' \in B_m$ fitting with $[\Gamma, \Delta]$, we have that $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ is a c.c.s. that fits with Γ; hence

  $$F[\Gamma]\,[x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n'] \equiv F[\Gamma, \Delta]\,[x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n', y_1 \leftarrow b_1 = b_1', \ldots, y_m \leftarrow b_m = b_m']$$

  is computable by assumption.

- (associate) Consider any possible form of the judgement J.

  - If $F \equiv C\ type$, then there is no condition.
  - If $F \equiv C = D$, then the associate judgement $C\ type\ [\Gamma, \Delta]$ is computable by the previous case.
  - If $F \equiv c \in C$, then the associate judgement $C\ type\ [\Gamma, \Delta]$ is computable by the first case.
  - If $F \equiv c = d \in C$, then the associate judgement $c \in C\ [\Gamma, \Delta]$ is computable by the previous case.

□

The next lemma on the reflexivity rule states that the rule preserves computability.

Reflexivity rules

Lemma 5.3.2 (Reflexivity on elements)**.** *The reflexivity on elements rule preserves computability, that is, if*

$$a \in A\ [\Gamma]$$

is computable, then

$$a = a \in A\ [\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

– (associate judgements) The associate judgements are computable by hypothesis.

– (parts) To prove the computability of the parts we proceed by structural induction on each possible form of the values of a and A.

* If $a \Rightarrow \star$ and $A \Rrightarrow \top$, then there is no condition to show.
* If $a \Rightarrow \langle c, d \rangle$ and $A \Rrightarrow \Sigma_{x \in C} D(x)$, then $c \in C$ and $d \in D(c)$ are computable and by induction hypothesis also $c = c \in C$ and $d = d \in D(c)$.
* If $a \Rightarrow \mathsf{eq}_C(c_3)$ and $A \Rrightarrow \mathsf{Eq}\,(C, c_1, c_2)$, then $c_1 = c_2 \in C$ is computable by assumption.
* If $a \Rightarrow [c]$ and $A \Rrightarrow C/\top$, then $c \in C$ is computable by assumption.

• Subcase $\Gamma \neq \emptyset$.

– (associate) The computability of the associate judgement of $a = a \in A\ [\Gamma]$ is given by hypothesis.

– (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable;

$$a = a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$a \in A\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n] \equiv \\ a = a \in A\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable by assumption.

□

Lemma 5.3.3 (Reflexivity on types)**.** *The reflexivity on types rule preserves computability, that is, if*

$$A \textit{ type } [\Gamma]$$

is computable, then

$$A = A\,[\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

– (associate judgements) The associate judgements are computable by hypothesis.

– (parts) To prove the computability of the parts we proceed by structural induction on each possible form of the values of A.

* If $A \Rrightarrow \top$, then there is no condition to show.
* If $A \Rrightarrow \Sigma_{x \in C} D(x)$, then C *type* and $D(x)$ *type* $[x \in C]$ are computable, and by induction hypothesis also $C = C$ and $D(x) = D(x)\ [x \in C]$.
* If $A \Rrightarrow \mathsf{Eq}\,(C, c_1, c_2)$, then C *type*, $c_1 \in C$ and $c_2 \in C$ are computable; by induction hypothesis we have that $C = C$ is computable and by the previous lemma also $c_1 = c_1 \in C$ and $c_2 = c_2 \in C$.
* If $A \Rrightarrow C/\top$, then C *type* is computable and by induction hypothesis also $C = C$.

• Subcase $\Gamma \neq \emptyset$.

– (associate) The computability of the associate judgement of $A = A \in [\Gamma]$ is given by hypothesis.

– (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

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

is computable;

$$A = A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$A \textit{ type } [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n] \equiv A = A\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable by assumption.

□

The substitution lemmas

The following lemma 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.

Lemma 5.3.4 (Head substitution). *Let $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$ be a context, $J \equiv F[\Gamma]$ be a computable judgement, $a_1 \in A_1, \ldots, a_i \in A_i$ and $a_1 = a'_1 \in A_1, \ldots, a_i = a'_i \in A_i$, for $i < n$, be c.c.s that fit with the context $[x_1 \in A_1, \ldots, x_i \in A_i]$. Then*

1. *$J[x_1 := a_1, \ldots, x_i := a_i]$ is a computable judgement.*
2. *$J[x_1 \leftarrow a_1 = a'_1, \ldots, x_i \leftarrow a_i = a'_i]$ is a computable judgement.*

Proof. Let $\Delta \equiv \left[x_{i+1} \in A'_{i+1}, \ldots, x_n \in A'_n\right]$, where $A'_j = A_j(a_1, \ldots a_i)$, for $i + 1 \leq j \leq n$, and let $a_{i+1} \in A'_{i+1}, \ldots, a_n \in A'_n$ be a c.c.s. that fits with the context Δ. 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 $J[x_1 \leftarrow a_1 = a'_1, \ldots, x_i \leftarrow a_i = a'_i]$ it is possible to find out a c.c.s. saturating J and yielding the same judgement.

1. • (substitution $:=$) For any c.c.s. $a_{i+1} \in A'_{i+1}, \ldots, a_n \in A'_n$ we have that $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. that fits with Γ; 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 $\leftarrow$) For any c.c.s. $a_{i+1} = a'_{i+1} \in A'_{i+1}, \ldots, a_n = a'_n \in A'_n$ that fits with the context Δ, we have that $a_1 = a_1 : A_1, \ldots, a_i = a_i : A_i, a_{i+1} = a'_{i+1} : A_{i+1}, \ldots, a_n = a'_n : A_n$ is a c.c.s. that fits with Γ. Note that the reflexivity-on-elements lemma 5.3.2 must be used here. Hence

$$\begin{aligned}&(J[x_1 := a_1, \ldots, x_i := a_i])\,\left[x_{i+1} \leftarrow a_{i+1} = a'_{i+1}, \ldots, x_n \leftarrow a_n = a'_n\right]\\ &\equiv J\left[x_1 \leftarrow a_1 = a_1, \ldots, x_i \leftarrow a_i = a_i, x_{i+1} \leftarrow a_{i+1} = a'_{i+1}, \ldots, x_n \leftarrow a_n = a'_n\right]\end{aligned}$$

is computable.

• (associate) Consider any possible form of the judgement J.

– If $F \equiv C\ type$, then there is no condition.

– If $F \equiv C = D$, then the associate judgement $C\ type\ [\Gamma]\ [x_1 := a_1, \ldots, x_i := a_i]$ is computable by the previous case.

– If $F \equiv c \in C$, then the associate judgement $C\ type\ [\Gamma]\ [x_1 := a_1, \ldots, x_i := a_i]$ is computable by the first case.

– If $F \equiv c = d \in C$, then the associate judgement $c \in C\ [\Gamma]\ [x_1 := a_1, \ldots, x_i := a_i]$ is computable by the previous case.

2. • (associate judgements) The computability of the associate judgements follows from case 1 since if $a_1 = a'_1 : A_1, \ldots, a_i = a'_i : A_i$, for $i < n$, are computable then also $a_1 : A_1, \ldots, a_i : A_i$ are computable and

$$J[x_1 := a_1, \ldots, x_i := a_i]$$

is the associate judgement of

$$J[x_1 \leftarrow a_1 = a'_1, \ldots, x_i \leftarrow a_i = a'_i]$$

whose computability is required.

• (substitution $:=$) For any c.c.s. $a_{i+1} \in A'_{i+1}, \ldots, a_n \in A'_n$ that fits with the context Δ, we have that $a_1 = a'_1 : A_1, \ldots, a_i = a'_i : A_i, a_{i+1} = a_{i+1} : A_{i+1}, \ldots, a_n =$

$a_n : A_n$ is a c.c.s. that fits with Γ; hence also

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

is computable.

- (substitution $\leftarrow$) For any c.c.s. $a_{i+1} = a'_{i+1} : A'_{i+1}, \ldots, a_n = a'_n : A'_n$ that fits with the context Δ, we have that $a_1 = a'_1 : A_1, \ldots, a_i = a'_i : A_i, a_{i+1} = a'_{i+1} : A_{i+1}, \ldots, a_n = a'_n : A_n$ is a c.c.s. that fits with Γ; hence

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

is computable.

□

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 substitution rules are not essential to prove the result we will compact the sentence as much as possible.

Lemma 5.3.5 (Tail substitution). *Let Γ be a context, $\Delta \equiv [\Gamma, x_1 \in A_1, \ldots, x_n \in A_n]$ be a context, $J \equiv F[\Delta]$ be a computable judgement, $a_1 \in \underline{A_1}\,[\Gamma], \ldots, a_n \in \underline{A_n}\,[\Gamma]$ and $a_1 = a'_1 \in \underline{A_1}\,[\Gamma]$, $\ldots, a_n = a'_n \in \underline{A_n}\,[\Gamma]$ be two lists of substitutions that fit with the context $[x_1 \in A_1, \ldots, x_n \in A_n]$. Then*

1. *$J[x_1 := a_1, \ldots, x_n := a_n]$ is a computable judgement.*
2. *$J[x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$ is a computable judgement.*

Proof. If the context Γ is empty then the claim holds by definition. Thus, let us suppose that $\Gamma \equiv [s_1 \in S_1, \ldots, s_m \in S_m]$, for some $m > 0$.

1. - (substitution :=) For any c.c.s. $c_1 \in S_1, \ldots, c_m \in S_m$ fitting with the context Γ, we define
$$d_i \equiv a_i(c_1, \ldots c_m) \qquad 1 \leq i \leq n$$

Then

$$d_i \in \underline{A_i}(c_1, \ldots, c_m)$$

are computable. Moreover we have

$$\begin{aligned}\underline{A_i}(c_1, \ldots, c_m) &\equiv (A_i(a_1, \ldots, a_{i-1}))(c_1, \ldots, c_m)\\ &\equiv A_i(c_1, \ldots, c_m, d_1, \ldots, d_{i-1})\\ &\equiv A_i\end{aligned}$$

Therefore $c_1 \in S_1, \ldots, c_m \in S_m, d_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with the context Δ. Hence

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

is computable.

- (substitution $\leftarrow$) For any c.c.s. $c_1 = c'_1 \in S_1, \ldots, c_m = c'_m \in S_m$ that fits with the context Γ, we define

$$d_i \equiv a_i(c_1, \ldots c_m) \qquad 1 \leq i \leq n$$

and

$$d'_i \equiv a_i(c'_1, \ldots c'_m) \qquad 1 \leq i \leq n$$

Then

$$d_i = d'_i \in \underline{A_i}(c_1, \ldots, c_m)$$

are computable. Moreover from the previous point we have

$$A_i \equiv \underline{A_i}(c_1, \ldots, c_m)$$

Then $c_1 = c'_1 \in S_1, \ldots, c_m = c'_m \in S_m, d_1 = d'_1 \in A_1, \ldots, d_n = d'_n \in A_n$ is a c.c.s. that fits with the context Δ. Hence

$$\begin{aligned}&(J[x_1 := a_1, \ldots, x_n := a_n])\,[s_1 \leftarrow c_1 = c'_1, \ldots, s_m \leftarrow c_m = c'_m] \equiv\\ &J[s_1 \leftarrow c_1 = c'_1, \ldots, s_m \leftarrow c_m = c'_m, x_1 \leftarrow d_1 = d'_1, \ldots, x_n \leftarrow d_n = d'_n]\end{aligned}$$

is computable.

- (associate) Consider any possible form of the judgement J.
  - If $F \equiv C\ type$, then there is no condition.
  - If $F \equiv C = D$, then the associate judgement $C\ type\ [\Delta]\ [x_1 := a_1, \ldots, x_i := a_n]$ is computable by the previous case.

– If $F \equiv c \in C$, then the associate judgement $C\,type\,[\Delta]\,[x_1 := a_1, \ldots, x_i := a_n]$ is computable by the first case.
– If $F \equiv c = d \in C$, then the associate judgement $c \in C\,[\Delta]\,[x_1 := a_1, \ldots, x_i := a_n]$ is computable by the previous case.

2. • (associate judgements) The computability of the associate judgements follows from case 1.
• (substitution $:=$) For any c.c.s. $c_1 \in S_1, \ldots, c_m \in S_m$ fitting with the context Γ, we define

$$d_i \equiv a_i(c_1, \ldots c_m) \qquad 1 \leq i \leq n$$

and

$$d'_i \equiv a'_i(c_1, \ldots c_m) \qquad 1 \leq i \leq n$$

Then

$$d_i = d'_i \in \underline{A_i}(c_1, \ldots, c_m)$$

is computable. Therefore, since

$$A_i \equiv \underline{A_i}(c_1, \ldots, c_m)$$

$c_1 = c_1 \in S_1, \ldots, c_m = c_m \in S_m, d_1 = d'_1 \in A_1, \ldots, d_n = d'_n \in A_n$ is a c.c.s. fitting with the context Δ. Hence

$$(J\,[x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n])\,[s_1 := c_1, \ldots, s_m := c_m] \equiv$$
$$J\,[s_1 \leftarrow c_1 = c_1, \ldots, s_m \leftarrow c_m = c_m, x_1 \leftarrow d_1 = d'_1, \ldots, x_n \leftarrow d_n = d'_n]$$

is computable.
• (substitution $\leftarrow$) For any c.c.s. $c_1 = c'_1 : S_1, \ldots, c_m = c'_m : S_m$ fitting with the context Γ, we define

$$d_i \equiv a_i(c_1, \ldots c_m) \qquad 1 \leq i \leq n$$

and

$$d'_i \equiv a_i(c'_1, \ldots c'_m) \qquad 1 \leq i \leq n$$

Then

$$d_i = d'_i \in \underline{A_i}(c_1, \ldots, c_m)$$

are computable. Therefore, since

$$A_i \equiv \underline{A_i}(c_1, \ldots, c_m)$$

, $c_1 = c'_1 \in S_1, \ldots, c_m = c'_m \in S_m, d_1 = d'_1 \in A_1, \ldots, d_n = d'_n \in A_n$ is a c.c.s.

fitting with the context Δ. Hence

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

is computable.

□

The following lemmas are a key point in the proof of computability since they establish that structural rules preserve computability.

Lemma 5.3.6. *The following properties on closed judgements hold*

1. *If $J_1 \equiv a = c \in A$, $J_2 \equiv b = d \in A$ and $J_3 \equiv a = b \in A$ are computable, then $c = d \in A$ is computable.*
2. *The elements in equal types rule preserves computability, that is, if $J_4 \equiv A = B$ is computable, then $J_5 \equiv a \in A$ is computable if and only if $a \in B$ is computable.*
3. *The equal elements in equal types rule preserves computability, that is, if $J_6 \equiv A = B$ is computable, then $J_7 \equiv a = b \in A$ is computable, if and only if $a = b \in B$ is computable.*

Proof. 1. $c = d \in A$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J_1}{c = a \in A}\,\text{sym-tm} \qquad \dfrac{J_3 \qquad J_2}{a = d \in A}\,\text{trans-tm}}{c = d \in A}\,\text{trans-tm}$$

2. By conv rule.
3. By conv-eq rule.

By induction on the maximum length $l \geq 2$ of the derivation of the hypotheses J_1, J_2, J_3, J_4, J_5, J_6 and J_7.

- ($l = 2$) The minimum l is 2, which is the case when J_1, J_2, J_3 and J_7 are $\star = \star in \top$, J_4 and J_6 are $\top = \top$ and J_5 is $\star \in \top$ then also $c_1 = d_1 \in A$ is $\star = \star \in \top$, $a_2 \in B_2$ is $\star \in \top$ and $a_3 = b_3 \in B_3$ is $\star = \star \in \top$.

- ($l \to l+1$) Suppose that the maximum length of the derivation of the hypotheses J_1, J_2, J_3 and J_4 is $l+1$, then

  1. – (associate judgements) The associate judgements are computable by hypothesis.
     – (parts) To prove the computability of the parts we analyze each possible form of the values of a, b, c, d and A.
       * If $a \Rightarrow \star$, $b \Rightarrow \star$, $c \Rightarrow \star$, $d \Rightarrow \star$ and $A \Rrightarrow \top$, then there is no condition to show.
       * If $a \Rightarrow \langle e_1, f_1\rangle$, $b \Rightarrow \langle e_2, f_2\rangle$, $c \Rightarrow \langle e_3, f_3\rangle$, $d \Rightarrow \langle e_4, f_4\rangle$ and $A \Rrightarrow \Sigma_{x\in E}F(x)$, then $e_1 = e_3 \in E$, $e_2 = e_4 \in E$, $e_1 = e_2 \in E$, $f_1 = f_3 \in F(e_1)$, $f_2 = f_4 \in F(e_2)$ and $f_1 = f_2 \in F(e_1)$ are computable; since J_1 is computable, A *type* is computable and, by definition, $F(x)$ *type* $[x \in E]$ is computable and so is $F(e_1) = F(e_2)$. By inductive hypothesis (case 3), $f_2 = f_4 \in F(e_1)$ is computable and also $e_3 = e_4 \in E$ and $f_3 = f_4 \in F(e_1)$ are computable; hence the result follows by inductive hypothesis again (case 3).
       * If $a \Rightarrow \mathsf{eq}_E(e_3)$, $b \Rightarrow \mathsf{eq}_E(e_4)$, $c \Rightarrow \mathsf{eq}_E(e_5)$, $d \Rightarrow \mathsf{eq}_E(e_6)$ and $A \Rrightarrow \mathsf{Eq}\,(E, e_1, e_2)$, then $e_1 = e_2 \in E$ is computable by assumption.
       * If $a \Rightarrow [e_1]$, $b \Rightarrow [e_2]$, $c \Rightarrow [e_3]$, $d \Rightarrow [e_4]$ and $A \Rrightarrow E/\top$, then $e_3 \in E$ and $e_4 \in E$ are computable by assumption.

  2. Let us prove the only-if part (the proof of the if-part is completely similar).
     – (associate judgements) The associate judgement is computable by hypothesis.
     – (evaluation) $a \Rightarrow g$ by hypothesis.
     – (correct evaluation) $a = g \in B$ is derivable using the conv-eq rule.
     – (parts) To prove the computability of the parts we analyze each possible form of the values of a, A and B.
       * If $a \Rightarrow \star$, $A \Rrightarrow \top$ and $B \Rrightarrow \top$, then there is no condition to show.
       * If $a \Rightarrow \langle c_1, c_2\rangle$, $A \Rrightarrow \Sigma_{x\in C_1}C_2(x)$ and $B \Rrightarrow \Sigma_{x\in D_1}D_2(x)$, then $C_1 = D_1$, $C_2(x) = D_2(x)$ $[x \in C_1]$, $c_1 \in C_1$, $c_2 \in C_2(c_1)$ are computable; then $C_2(c_1) = D_2(c_1)$ is computable, so by induction hypothesis $c_1 \in D_1$ and $c_2 \in D_2(c_1)$ are computable.
       * If $a \Rightarrow \mathsf{eq}_C(c_3)$, $A \Rrightarrow \mathsf{Eq}\,(C, c_1, c_2)$ and $B \Rrightarrow \mathsf{Eq}\,(D, d_1, d_2)$, then $C = D$, $c_1 = c_2 \in C$, $c_1 = d_1 \in C$ and $c_2 = d_2 \in C$ are computable; then, by inductive hypothesis (case 1), $d_1 = d_2 \in C$ is computable, hence the result follows by inductive hypothesis (case 3).
       * If $a \Rightarrow [c_1]$, $A \Rrightarrow C/\top$ and $B \Rrightarrow D/\top$, then $c \in C$ and $C = D$ are computable and by inductive hypothesis (case 2) also $c \in D$.

3. Let us prove the only-if part (the proof of the if-part is completely similar).
   - (associate judgements) The associate judgements are computable by inductive hypothesis (case 2).
   - (parts) To prove the computability of the parts we proceed by structural induction on each possible form of the values of a, A and B.
     * If $a \Rightarrow \star$, $b \Rightarrow \star$, $A \Rrightarrow \top$ and $B \Rrightarrow \top$, then there is no condition to show.
     * If $a \Rightarrow \langle c_1, d_1\rangle$, $b \Rightarrow \langle c_2, d_2\rangle$, $A \Rrightarrow \Sigma_{x\in C}D(x)$ and $B \Rrightarrow \Sigma_{x\in E}F(x)$, then $C = E$, $D(x) = F(x)\ [x \in C]$, $c_1 = c_2 \in C$, $d_1 = d_2 \in D(c_1)$ are computable; then $c_1 \in C$ is computable and also $D(c_1) = F(c_1)$ is computable, so by induction hypothesis $c_1 = c_2 \in E$ and $d_1 = d_2 \in F(c_1)$ are computable.
     * If $a \Rightarrow \mathsf{eq}_C(c_3)$, $b \Rightarrow \mathsf{eq}_C(c_4)$, $A \Rrightarrow \mathsf{Eq}\,(C, c_1, c_2)$ and $B \Rrightarrow \mathsf{Eq}\,(D, d_1, d_2)$, then $C = D$, $c_1 = c_2 \in C$, $c_1 = d_1 \in C$ and $c_2 = d_2 \in C$ are computable; then, by inductive hypothesis (case 1), $d_1 = d_2 \in C$ is computable, hence the result follows by inductive hypothesis.
     * If $a \Rightarrow [c_1]$, $b \Rightarrow [c_2]$, $A \Rrightarrow C/\top$ and $B \Rrightarrow D/\top$, then $c_1 \in C$, $c_2 \in C$ and $C = D$ are computable and by inductive hypothesis (case 2) also $c_1 \in D$ and $c_1 \in D$ are computable.

□

Lemma 5.3.7 (Assumption in equal types)**.** *If $A = B\ [\Gamma]$ is computable then $J \equiv F\ [\Gamma, x \in A]$ is computable if and only if $F\ [\Gamma, x \in B]$.*

Proof. Let us prove the only if-part (the proof of the if-part is completely similar).

To prove that $F\ [\Gamma, x \in B]$ is derivable, we proceed by induction on the length l of the derivation of the judgement J. Base cases are trivial. Suppose J has a derivation of length $l + 1$, then

- If the last rule of the derivation tree is a var rule, then there are two cases
  - ($F \equiv x \in A$) Since we know that Γ *cont* and B *type* $[\Gamma]$ are derivable, we get a derivation of $x \in A\ [\Gamma, x \in B]$

$$\dfrac{\dfrac{\dfrac{\Gamma\ cont \qquad B\ type\ [\Gamma]}{\Gamma, x \in B\ cont}\ \text{2c}}{x \in B\ [\Gamma, x \in B]}\ \text{var} \qquad \dfrac{A = B\ [\Gamma]}{A = B\ [\Gamma, x \in B]}\ \text{ind-ty-eq}}{B = A\ [\Gamma, x \in B]}\ \text{refl-ty}}{x \in A\ [\Gamma, x \in B]}\ \text{conv}$$

– ($F \equiv x \in B$ or $F \equiv y \in C$) A derivation is built easily by applying the var rule.

- If the last rule of the derivation tree is $r \neq$ var, then the premises $F_1\ [\Gamma, x \in A, \Delta_1]$, ..., $F_n\ [\Gamma, x \in A, \Delta_n]$ are derivable with length of derivation at most l. By using the rules of suitable exchange and by inductive hypothesis, we get that $F_1\ [\Gamma, x \in B, \Delta_1]$, ..., $F_n\ [\Gamma, x \in B, \Delta_n]$ are derivable. Hence the result follows by applying the rule r.

Then

- (substitution :=) In order to analyse the substituted judgements, suppose Γ is the context $[x_1 \in A_1, \ldots x_n \in A_n]$. Since $A = B\ [\Gamma]$ is computable then for any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n, e \in A$ fitting with $[\Gamma, x \in A]$, we have that

  $$A = B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

  and

  $$F\,[\Gamma, x \in A][x_1 := a_1, \ldots, x_n := a_n, x := e]$$

  are computable hence, by point 2 of lemma 5.3.6, $e \in B$ is a computable judgement and thus the same substitution fits also with $[\Gamma, x \in B]$. Hence

  $$F\,[\Gamma, x \in B][x_1 := a_1, \ldots, x_n := a_n, x := e]$$

  is computable.

- (substitution $\leftarrow$) In order to analyse the substituted judgements, suppose Γ is the context $[x_1 \in A_1, \ldots x_n \in A_n]$. Since $A = B\ [\Gamma]$ is computable then for any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n, e = e' \in A$ fitting with $[\Gamma, x \in A]$, we have that

  $$A = B\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

  and

  $$F\,[\Gamma, x \in A][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n', x \leftarrow e = e']$$

  are computable hence, by point 3 of lemma 5.3.6, $e = e' \in B$ is a computable judgement and thus the same substitution fits also with $[\Gamma, x \in B]$. Hence

  $$F\,[\Gamma, x \in B][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n', x \leftarrow e = e']$$

is computable.

- (associate) Consider any possible form of the judgement J.
  - If $F \equiv C(x)\ \mathit{type}$, then there is no condition.
  - If $F \equiv C(x) = D(x)$, then the associate judgement $C\ \mathit{type}\ [\Gamma, x \in B]$ is computable by the previous case.
  - If $F \equiv c(x) \in C(x)$, then the associate judgement $C\ \mathit{type}\ [\Gamma, x \in B]$ is computable by the first case.
  - If $F \equiv c(x) = d(x) \in C(x)$, then the associate judgement $c(x) \in C(x)[\Gamma, x \in B]$ is computable by the previous case.

□

Lemma 5.3.8. *If $J \equiv a = b \in A\ [\Gamma]$ is computable, then $b \in A\ [\Gamma]$ is computable.*

Proof.
- Subcase $\Gamma = \emptyset$. This subcase is trivial, since $b \in A$ is computable by assumption.
- Subcase $\Gamma \neq \emptyset$.
  - (associate) The associate judgement of J is $a \in A\ [\Gamma]$, so $A\ \mathit{type}\ [\Gamma]$ is computable.
  - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then
    $$a = b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
    is computable;
    $$b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
    has empty context, then it is computable by definition.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ and $a_1' \in A_1, \ldots, a_n' \in A_n$ are c.c.s. fitting with Γ, and so
    $$a = b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
    $$a = b \in A\ [\Gamma][x_1 := a_1', \ldots, x_n := a_n']$$
    and
    $$a \in A\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

are computable;

$$b \in A\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable by point 1 of previous lemma 5.3.6.

□

Lemma 5.3.9. *The following hold*

1. *If $A = B\,[\Gamma]$ is computable, then B type $[\Gamma]$ is computable.*
2. *If $J_1 \equiv A = C\,[\Gamma], J_2 \equiv B = D\,[\Gamma]$ and $J_3 \equiv A = B\,[\Gamma]$ are computable, then $C = D\,[\Gamma]$ is computable.*

Proof. 1. By derivability of associate judgements.

2. $C = D\,[\Gamma]$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J_1}{C = A\,[\Gamma]}\,\text{sym-ty} \qquad \dfrac{J_3 \qquad J_2}{A = D\,[\Gamma]}\,\text{trans-ty}}{C = D\,[\Gamma]}\,\text{trans-ty}$$

- Subcase $\Gamma = \emptyset$.

  1. This subcase is trivial, since *B type* is computable by assumption.
  2. – (associate judgements) The associate judgements are computable by hypothesis.
     – (parts) To prove the computability of the parts we proceed by structural induction on each possible form of the values of a, b, c and A.
       * If $A \Rrightarrow \top$, $B \Rrightarrow \top$, $C \Rrightarrow \top$ and $D \Rrightarrow \top$, then there is no condition to show.
       * If $A \Rrightarrow \Sigma_{x \in E_1} E_2(x)$, $B \Rrightarrow \Sigma_{x \in F_1} F_2(x)$, $C \Rrightarrow \Sigma_{x \in G_1} G_2(x)$ and $D \Rrightarrow \Sigma_{x \in H_1} H_2(x)$, then $E_1 = G_1$, $F_1 = H_1$, $E_1 = F_1$, $E_2(x) = G_2(x)\,[x \in E_1]$, $F_2(x) = H_2(x)\,[x \in F_1]$ and $E_2(x) = F_2(x)\,[x \in E_1]$ are computable; by lemma 5.3.7, also $F_2(x) = H_2(x)\,[x \in E_1]$ is computable and, by induction hypothesis, $G_1 = H_1$ and $G_2(x) = H_2(x)\,[x \in E_1]$ are computable; hence the result follows by lemma 5.3.7.

* If $A \Rrightarrow \mathsf{Eq}(E, e_1, e_2)$, $B \Rrightarrow \mathsf{Eq}(F, f_1, f_2)$, $C \Rrightarrow \mathsf{Eq}(G, g_1, g_2)$ and $D \Rrightarrow \mathsf{Eq}(H, h_1, h_2)$, then $E = G$, $F = H$, $E = F$, $e_1 = g_1 \in E$, $f_1 = h_1 \in F$, $e_1 = f_1 \in E$, $e_2 = g_2 \in E$, $f_2 = h_2 \in F$, $e_2 = f_2 \in E$ are computable; by point 3 of lemma 5.3.6, $f_1 = h_1 \in E$ and $f_2 = h_2 \in E$ are computable, then, by point 1 of lemma 5.3.6, $g_1 = h_1 \in E$ and $g_2 = h_2 \in E$ are computable, by inductive hypothesis $G = H$ is computable and thus the result follows by point 3 of lemma 5.3.6.
* If $A \Rrightarrow E/\top$, $B \Rrightarrow F/\top$, $C \Rrightarrow G/\top$, $D \Rrightarrow H/\top$, then $E = G$, $F = H$ and $E = F$ are computable; hence the result follows by inductive hypothesis.

- Subcase $\Gamma \neq \emptyset$.

1. – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$A = B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable;

$$B\ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by definition.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ and $a'_1 \in A_1, \ldots, a'_n \in A_n$ are c.c.s. fitting with Γ, and so

$$A = B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$A = B\,[\Gamma][x_1 := a'_1, \ldots, x_n := a'_n]$$

and

$$A\ \mathit{type}\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

are computable;

$$B\ \mathit{type}\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

2. – (associate) The computability of the associate judgement of $C = D\ [\Gamma]$ follows by previous point.

– (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$A = C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$B = D\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$A = B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$C = D\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$A = C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$B = D\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

and

$$A = B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$C = D\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

□

Lemma 5.3.10. *The following hold*

1. *If $J_1 \equiv a = c \in A\,[\Gamma]$, $J_2 \equiv b = d \in A\,[\Gamma]$ and $J_3 \equiv a = b \in A\,[\Gamma]$ are computable, then $c = d \in A\,[\Gamma]$ is computable.*

2. *The elements in equal types rule preserves computability, that is, if $A = B\,[\Gamma]$ is computable, then $a \in A\,[\Gamma]$ is computable if and only if $a \in B\,[\Gamma]$ is computable.*

3. *The equal elements in equal types rule preserves computability, that is, if $A = B\,[\Gamma]$ is computable, then $a = b \in A\,[\Gamma]$ is computable, if and only if $a = b \in B\,[\Gamma]$ is computable.*

Proof. 1. $c = d \in A\,[\Gamma]$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J_1}{c = a \in A\,[\Gamma]}\,\text{sym-tm} \qquad \dfrac{J_3 \quad J_2}{a = d \in A\,[\Gamma]}\,\text{trans-tm}}{c = d \in A\,[\Gamma]}\,\text{trans-tm}$$

2. By conv rule.

3. By conv-eq rule.

4. By derivability of associate judgements.

If $\Gamma \equiv \emptyset$, then just apply lemma 5.3.6, otherwise

1.
   - (associate) The computability of the associate judgement of $c = d \in A\,[\Gamma]$ follows by lemma 5.3.8.
   - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$a = c \in A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     $$b = d \in A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     and

     $$a = b \in A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     are computable;

     $$c = d \in A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     has empty context, then it is computable by point 1 of lemma 5.3.6.
   - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

     $$a = c \in A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     $$b = d \in A\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

     and

     $$a = b \in A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     are computable;

     $$c = d \in A\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

     has empty context, then it is computable by point 1 of lemma 5.3.6.

2.
   - (associate) The computability of the associate judgement $B\ type\ [\Gamma]$ follows by point 1 of previous lemma 5.3.9.
   - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$A = B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     and

     $$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     are computable;

     $$a \in B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     has empty context, then it is computable by point 2 of lemma 5.3.6.
   - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

     $$A = B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     and

     $$a \in A\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

     are computable;

     $$a \in B\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

     has empty context, then it is computable by point 3 of lemma 5.3.6.

3. Let us prove the only-if part (the proof of the if-part is completely similar).

   - (associate) The computability of the associate judgement $a \in B\ [\Gamma]$ follows by previous point.
   - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$A = B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     and

     $$a = b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     are computable;

     $$a = b \in B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by point 3 of lemma 5.3.6.

- (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so
$$A = B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
and
$$a = b \in A\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$
are computable;
$$a = b \in B\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$
has empty context, then it is computable by point 3 of lemma 5.3.6.

□

Symmetry rules

Lemma 5.3.11 (Symmetry on elements)**.** *The symmetry on elements rule preserves computability, that is, if*
$$a = b \in A\,[\Gamma]$$
is computable, then
$$b = a \in A\,[\Gamma]$$
is computable.

Proof.
- Subcase $\Gamma = \emptyset$.
  - (associate judgements) The associate judgements are computable by hypothesis.
  - (parts) To prove the computability of the parts we proceed by structural induction on each possible form of the values of a, b and A.
    - If $a \Rightarrow \star$, $b \Rightarrow \star$ and $A \Rrightarrow \top$, then there is no condition to show.
    - If $a \Rightarrow \langle c_1, d_1\rangle$, $b \Rightarrow \langle c_2, d_2\rangle$ and $A \Rrightarrow \Sigma_{x\in C} D(x)$, then $c_1 = c_2 \in C$ and $d_1 = d_2 \in D(c_1)$ are computable and, by inductive hypothesis, also $c_2 = c_1 \in C$ and $d_2 = d_1 \in D(c_1)$ are computable. The judgement $a \in A$ is computable and so is A *type*, thus $D(x)$ *type* $[x \in C]$ and $D(c_2) = D(c_1)$ are computable too. Hence the result follows by point 3 of lemma 5.3.6.

* If $a \Rightarrow \mathsf{eq}_C(c_3)$, $b \Rightarrow \mathsf{eq}_C(c_4)$ and $A \Rrightarrow \mathsf{Eq}\,(C, c_1, c_2)$, then $c_1 = c_2 \in C$ is computable by assumption.
* If $a \Rightarrow [c_1]$, $b \Rightarrow [c_2]$ and $A \Rrightarrow C/\top$, then $c_1 \in C$ and $c_2 \in C$ are computable by assumption.

- Subcase $\Gamma \neq \emptyset$.
  - (associate) The computability of the associate judgement $b \in A\ [\Gamma]$ follows by lemma 5.3.8.
  - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

    $$a = b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    is computable;

    $$b = a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

    $$a = b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
    $$a \in A\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$
    $$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    are computable, then by point 1 of lemma 5.3.6

    $$b = a \in A\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

    is computable.

□

Lemma 5.3.12 (Symmetry on types)**.** *The symmetry on elements rule preserves computability, that is, if*

$$A = B\ [\Gamma]$$

is computable, then

$$B = A\ [\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.
  - (associate judgements) The associate judgements are computable by hypothesis.
  - (parts) To prove the computability of the parts we proceed by structural induction on each possible form of A and B.
    * If $A \Rrightarrow \top$ and $B \Rrightarrow \top$, then there is no condition to show.
    * If $A \Rrightarrow \Sigma_{x \in C_1} C_2(x)$ and $B \Rrightarrow \Sigma_{x \in D_1} D_2(x)$, then $C_1 = D_1$ and $C_2(x) = D_2(x)\ [x \in C_1]$ are computable; by inductive hypothesis, also $D_1 = C_1$ and $D_2(x) = C_2(x)\ [x \in C_1]$, hence the result follows by lemma 5.3.7.
    * If $A \Rrightarrow \mathsf{Eq}\,(C, c_1, c_2)$ and $B \Rrightarrow \mathsf{Eq}\,(D, d_1, d_2)$, then $C = D$, $c_1 = d_1 \in C$ and $c_2 = d_2 \in C$ are computable; by lemma 5.3.11, $d_1 = c_1 \in C$ and $d_2 = c_2 \in C$ are computable, then, by point 3 of lemma 5.3.6, also $d_1 = c_1 \in D$ and $d_2 = c_2 \in D$ are computable. Hence the result follows by inductive hypothesis.
    * If $A \Rrightarrow C/\top$ and $B \Rrightarrow D/\top$, then $C = D$ is computable, hence the result follows by inductive hypothesis.

- Subcase $\Gamma \neq \emptyset$.
  - (associate) The computability of the associate judgement B *type* $[\Gamma]$ follows by lemma 5.3.9.2.
  - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

    $$A = B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    is computable;

    $$B = A\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then, by the symmetry lemma 5.3.11 also $a_1' = a_1 \in A_1, \ldots, a_n' = a_n \in A_n$ is a c.c.s. fitting with Γ, and so

    $$A = B\,[\Gamma][x_1 \leftarrow a_1' = a_1, \ldots, x_n \leftarrow a_n' = a_n]$$

is computable;

$$B = A\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

□

Transitivity rules

Lemma 5.3.13 (Transitivity on elements)**.** *The transitivity on elements rule preserves computability, that is, if*

$$a = b \in A\,[\Gamma]$$

and

$$b = c \in A\,[\Gamma]$$

are computable, then

$$a = c \in A\,[\Gamma]$$

is computable.

Proof. By using the symmetry lemma 5.3.11, we obtain that $b = a \in A\,[\Gamma]$ is computable. Hence $b \in A\,[\Gamma]$ is computable and, by the reflexivity lemma 5.3.2, $b = b \in A\,[\Gamma]$ is computable. Then the result follows by point 1 of lemma 5.3.6. □

Lemma 5.3.14 (Transitivity on types)**.** *The transitivity on types rule preserves computability, that is, if*

$$A = B\,[\Gamma]$$

and

$$B = C\,[\Gamma]$$

are computable, then

$$A = C\,[\Gamma]$$

is computable.

Proof. By using the symmetry lemma 5.3.12, we obtain that $B = A\,[\Gamma]$ is computable. Hence B *type* $[\Gamma]$ is computable and, by the reflexivity lemma 5.3.3, $B = B\,[\Gamma]$ is computable. Then the result follows by lemma 5.3.9.1. □

Lemma 5.3.15 (Assumption of variables)**.** *Let A type $[\Gamma]$ be a computable judgement. Then the judgement*

$$x \in A\ [\Gamma, x \in A, \Delta]$$

is computable.

Proof. Let $\Gamma \equiv [s_1 \in S_1, \ldots, s_k \in S_k], \Delta \equiv [z_1 \in C_1 \ldots z_m \in C_m]$

- (associate) The computability of the associate judgement

  $$A\ type\ [\Gamma, x \in A, \Delta]$$

  follows by weakening lemma 5.3.1.

- (substitution :=) Consider any c.c.s. $d_1 \in S_1, \ldots, d_k \in S_k, a \in A, c_1 \in C_1, \ldots, c_m \in C_m$ fitting with $[\Gamma, x \in A, \Delta]$. Then

  $$x \in A\ [\Gamma, x \in A, \Delta]\ [s_1 := d_1, \ldots, s_k := d_k, x := a, z_1 := c_1, \ldots, z_m := c_m]$$

  is computable by assumption.

- (substitution $\leftarrow$) Consider any c.c.s. $d_1 = d_1' \in S_1, \ldots, d_k = d_k' \in S_k, a = a' \in A, c_1 = c_1' \in C_1, \ldots, c_m = c_m' \in C_m$ fitting with $[\Gamma, x \in A, \Delta]$. Then

  $$x \in A\ [\Gamma, x \in A, \Delta][s_1 := d_1 = d_1', \ldots, s_k \leftarrow d_k = d_k', x \leftarrow a = a', \\ z_1 \leftarrow c_1 = c_1', \ldots, z_m \leftarrow c_m = c_m']$$

  is computable by assumption.

□

5.3.2 The logical rules

We have now to analyze 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 the intuitionistic type theory, nevertheless, a complete formal proof of computability for these rules cannot be carried on without a substantial use of lemmas on structural rules.

Terminal type rules

Lemma 5.3.16 ($\top$-formation rules)**.** *The $\top$-formation rule preserves computability. That is the judgement*

$$\top \ \mathit{type}\ [\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.
  - (evaluation) $\top \Rrightarrow \top$ holds.
  - (correct evaluation) $\top = \top$ is derivable with the following derivation tree

$$\dfrac{\dfrac{}{\top \ \mathit{type}}\ \text{Tr}}{\top = \top}\ \text{refl-ty}$$

  - (parts) No condition to show.

- Subcase $\Gamma \neq \emptyset$.
  - (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$\top \ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n] \equiv \top \ \mathit{type}$$

    is computable by ($\Gamma = \emptyset$) subcase.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$\top \ \mathit{type}\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n'] \equiv \top = \top$$

    is computable, since it is derivable and the associate judgement $\top$ *type* is computable.

□

Lemma 5.3.17 ($\top$-introduction rules)**.** *The $\top$-introduction rule preserves computability. That is the judgement*

$$\star \in \top\ [\Gamma]$$

is computable

Proof. • Subcase $\Gamma = \emptyset$.

– (associate judgement) The computability of the associate judgement is immediate by the previous lemma on $\top$-formation rule.

– (evaluation) $\star \Rightarrow \star$ holds.

– (correct evaluation) $\star = \star \in \top$ is derivable with the following derivation tree

$$\dfrac{\dfrac{}{\star \in \top}\ \text{I-Tr}}{\star = \star \in \top}\ \text{refl-tm}$$

– (parts) No condition to show.

• Subcase $\Gamma \neq \emptyset$.

– (associate) The judgement $\top$ *type* $[\Gamma]$ is computable for the previous lemma on $\top$-formation rule

– (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$\star \in \top\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n] \equiv \star \in \top$$

is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$\star \in \top\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n'] \equiv \star = \star \in \top$$

is computable, since it is derivable and the associate judgement $\star \in \top$ is computable.

□

Lemma 5.3.18 ($\top$-equality rule)**.** *The $\top$-equality rule preserves computability, i.e., if the judgement $J \equiv t \in \top\,[\Gamma]$ is computable then the judgement*

$$t = \star \in \top\,[\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

– (associate judgements) The associate judgement $t \in \top$ is computable by assumption, while $\star \in \top$ is computable by the previous lemma on $\top$-introduction rule.
– (parts) No condition to show.

- Subcase $\Gamma \neq \emptyset$.
  – (associate) The judgement $t \in \top\,[\Gamma]$ is computable by assumption.
  – (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$t \in \top\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    is computable;

$$t = \star \in \top\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.
  – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$t = \star \in \top\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n'] \equiv t = \star \in \top\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    is computable by (substitution :=) subcase.

□

Indexed Sum type rules

Lemma 5.3.19 (Σ^*-formation rules)**.** *The Σ^*-formation rules preserve computability. That is*

1. *If $J_1 \equiv B$ type $[\Gamma]$ and $J_2 \equiv C(x)$ type $[\Gamma, x \in B]$ are computable judgement then*

$$\Sigma_{x \in B} C(x) \text{ type } [\Gamma]$$

   is computable.

2. *If $J_1 \equiv B = D\,[\Gamma]$ and $J_2 \equiv C(x) = E(x)\,[\Gamma, x \in B]$ is a computable judgement then*

$$\Sigma_{x \in B} C(x) = \Sigma_{x \in D} E(x)\,[\Gamma]$$

   is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (evaluation) $\Sigma_{x\in B}C(x) \Rrightarrow \Sigma_{x\in B}C(x)$ holds.
   – (correct evaluation) $\Sigma_{x\in B}C(x) = \Sigma_{x\in B}C(x)$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J_1 \quad J_2}{\Sigma_{x\in B}C(x)\ \mathit{type}}\,\Sigma}{\Sigma_{x\in B}C(x) = \Sigma_{x\in B}C(x)}\,\text{refl-ty}$$

   – (parts) The parts are J_1 and J_2 which are assumed to be computable.

2. – (associate judgements) The judgement $C(x)\ \mathit{type}\ [x \in B]$, associate of J_2 is computable by assumption, and, by lemma 5.3.9.2 also $E(x)\ \mathit{type}\ [x \in B]$ is computable. By lemma 5.3.7, we know that also $E(x)\ \mathit{type}\ [x \in D]$ is computable and hence the result follows by case 1.
   – (parts) The parts are J_1 and J_2 which are assumed to be computable.

• Subcase $\Gamma \neq \emptyset$.

1. – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$B\ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by assumption and

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

   is computable by head substitution lemma 5.3.4;

$$\Sigma_{x\in B}C(x)\ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$B\ \mathit{type}\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

   is computable by assumption and

$$C(x)\ \mathit{type}\ [\Gamma, x \in B][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable by head substitution lemma 5.3.4;

$$\Sigma_{x\in B}C(x)\ \mathit{type}\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

2. – (associate) The judgements $B\ \mathit{type}\ [\Gamma]$ and $C(x)\ \mathit{type}\ [\Gamma, x \in B]$ associates of J_1 and J_2 are computable. Hence the result follows by case 1.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$B = D\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by assumption and

$$C(x) = E(x)\ [\Gamma, x \in B][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by head substitution lemma 5.3.4;

$$\Sigma_{x\in B}C(x) = \Sigma_{x\in D}E(x)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$B = D\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

   is computable by assumption and

$$C(x) = E(x)\ [\Gamma, x \in B][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

   is computable by head substitution lemma 5.3.4;

$$\Sigma_{x\in B}C(x) = \Sigma_{x\in D}E(x)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

   has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

□

Lemma 5.3.20 (Σ-introduction rules)**.** *The Σ-introduction rules preserve computability. That is*

1. *If* $J_1 \equiv b \in B\,[\Gamma], J_2 \equiv c \in C(b)\,[\Gamma]$ *and* $J_3 \equiv \Sigma_{x\in B}C(x)$ *type* $[\Gamma]$ *are computable judgements then*

$$\langle b, c\rangle \in \Sigma_{x\in B}C(x)\,[\Gamma]$$

is computable.

2. *If* $J_1 \equiv b = d \in B\,[\Gamma], J_2 \equiv c = e \in C(b)\,[\Gamma]$ *and* $J_3 \equiv \Sigma_{x\in B}C(x)$ *type* $[\Gamma]$ *are computable judgements then*

$$\langle b, c\rangle = \langle d, e\rangle \in \Sigma_{x\in B}C(x)\,[\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.

  1. – (associate judgements) The associate judgement is J_3 which is computable by assumption.
     – (evaluation) $\langle b, c\rangle \Rightarrow \langle b, c\rangle$ holds.
     – (correct evaluation) $\langle b, c\rangle = \langle b, c\rangle \in \Sigma_{x\in B}C(x)$ is derivable by using first the Σ-introduction rule and then the reflexivity on elements rule.

$$\dfrac{\dfrac{J_1 \quad J_2 \quad J_3}{\langle b, c\rangle \in \Sigma_{x\in B}C(x)}\,\text{I-}\Sigma}{\langle b, c\rangle = \langle b, c\rangle \in \Sigma_{x\in B}C(x)}\,\text{refl-tm}$$

     – (parts) The parts are J_1 and J_2 which are assumed to be computable.

  2. – (associate judgements) The judgements $b \in B, d \in B, c \in C(b)$and $e \in C(b)$, associates of J_1 and J_2, are computable by definition. Hence the result follows from case 1.
     – (parts) The parts are J_1 and J_2 which are assumed to be computable.

- Subcase $\Gamma \neq \emptyset$.

  1. – (associate) The associate judgement is J_3 which is computable by assumption.
     – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$b \in B\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$c \in C(b)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$\Sigma_{x \in B} C(x)\ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$\langle b, c \rangle \in \Sigma_{x \in B} C(x)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$b \in B\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

$$c \in C(b)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

and

$$\Sigma_{x \in B} C(x)\ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$\langle b, c \rangle \in \Sigma_{x \in B} C(x)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

2. – (associate) The judgements $b \in B\ [\Gamma]$ and $c \in C(b)\ [\Gamma]$, associate respectively of J_1 and J_2 are computable. J_3 is also computable, hence the result follows by case 1.

– (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$b = d \in B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$c = e \in C(b)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$\Sigma_{x \in B} C(x)\ \mathit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$\langle b, c \rangle \in \Sigma_{x \in B} C(x)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by ($\Gamma = \emptyset$).

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$b = d \in B\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

$$c = e \in C(b)\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

and

$$\Sigma_{x\in B}C(x)\ \mathit{type}\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$\langle b, c\rangle \in \Sigma_{x\in B}C(x)\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$.

□

Lemma 5.3.21 (Σ-elimination rules)**.** *The Σ-elimination rules preserve computability. That is*

1. *If $J_1 \equiv M(z)\ \mathit{type}\,[\Gamma, z \in \Sigma_{x\in B}C(x)]$, $J_2 \equiv d \in \Sigma_{x\in B}C(x)\,[\Gamma]$ and $J_3 \equiv m(x,y) \in M(\langle x,y\rangle)\,[\Gamma, x \in B, y \in C(x)]$ are computable judgements then*

$$El_\Sigma(d, m) \in M(d)\,[\Gamma]$$

is computable.

2. *If $J_1 \equiv M(z)\ \mathit{type}\,[\Gamma, z \in \Sigma_{x\in B}C(x)]$, $J_2 \equiv d = d' \in \Sigma_{x\in B}C(x)\,[\Gamma]$ and $J_3 \equiv m(x,y) = m'(x,y) \in M(\langle x,y\rangle)\,[\Gamma, x \in B, y \in C(x)]$ are computable judgements then*

$$El_\Sigma(d, m) = El_\Sigma(d', m') \in M(d)\,[\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (associate judgements) The computability of the judgement $M(d)\ \mathit{type}$, associate of the judgement $El_\Sigma(d, m) \in M(d)$, follows by assumption.
   – (evaluation) J_2 is computable and $\Sigma_{x\in B}C(x) \Rrightarrow \Sigma_{x\in B}C(x)$, then $d \Rightarrow \langle e, f\rangle$ and the judgements $e \in B$ and $f \in C(e)$ are computable; thus it is a c.c.s.

fitting with $x \in B, y \in C(x)$; therefore $J_3[x := e, y := f]$, which is $m(e,f) \in M(\langle e,f\rangle)$, is a computable judgement. Hence $m(e,f) \Rightarrow g$ and the result follows by using the computation rule.

– (correct evaluation) Since J_2 is computable, we know that there exists a derivation of the judgement $\Sigma_{x\in B}C(x)$ *type*; hence $x \in \Sigma_{x\in B}C(x)$ is a correct assumption, and $d = \langle e,f\rangle \in \Sigma_{x\in B}C(x)$ is derivable. Let Π_1 be

$$\dfrac{d = \langle e,f\rangle \in \Sigma_{x\in B}C(x) \qquad \dfrac{J_1 \qquad x \in \Sigma_{x\in B}C(x)\ [x \in \Sigma_{x\in B}C(x)] \qquad J_3}{El_\Sigma(x,m) \in M(x)\ [x \in \Sigma_{x\in B}C(x)]}\ \text{E-}\Sigma}{El_\Sigma(d,m) = El_\Sigma(\langle e,f\rangle, m) \in M(p)}\ \text{sub-tm-eq}$$

then Π_2 is

$$\dfrac{\Pi_1 \qquad \dfrac{d = \langle e,f\rangle \in \Sigma_{x\in B}C(x) \qquad J_1}{M(p) = M(\langle e,f\rangle)}\ \text{sub-tm-eq}}{El_\Sigma(d,m) = El_\Sigma(\langle e,f\rangle, m) \in M(\langle e,f\rangle)}\ \text{conv-eq}$$

Since J_2 is computable, so are $e \in B$ and $f \in C(e)$ and then $J_3[x := e, y := f]$ is computable. Thus the judgements $m(e,f) = g \in M(\langle e,f\rangle)$, $e \in B$ and $f \in C(e)$ are derivable. Let Π_3 be

$$\dfrac{\dfrac{J_1 \qquad e \in B \qquad f \in C(e) \qquad J_3}{El_\Sigma(\langle e,f\rangle, m) = m(e,f) \in M(\langle e,f\rangle)}\ \text{C-}\Sigma \qquad m(e,f) = g \in M(\langle e,f\rangle)}{El_\Sigma(\langle e,f\rangle, m) = g \in M(\langle e,f\rangle)}\ \text{trans-tm}$$

then

$$\dfrac{\Pi_2 \qquad \Pi_3}{El_\Sigma(d,m) = g \in M(\langle e,f\rangle)}\ \text{trans-tm}$$

Hence

$$\dfrac{El_\Sigma(d,m) = g \in M(\langle e,f\rangle) \qquad \dfrac{\dfrac{d = \langle e,f\rangle \in \Sigma_{x\in B}C(x) \qquad J_1}{M(d) = M(\langle e,f\rangle)}\ \text{sub-tm-eq}}{M(\langle e,f\rangle) = M(d)}\ \text{sym-ty}}{El_\Sigma(d,m) = g \in M(d)}\ \text{conv-eq}$$

– (parts) Since J_2 is computable we know that $d \Rightarrow \langle e,f\rangle$, then $e \in B$ and $f \in (C(e)$ are computable by the previous lemma on nat-introduction rules. Hence we can deduce that the judgement $d = \langle e,f\rangle \in \Sigma_{x\in B}C(x)$ is computable. Therefore, since J_1 is computable, we obtain that the judgement $M(\langle e,f\rangle) = M(d)$ is a computable judgement. Then, since J_3 is computable, so is $J_3[x := e, y := f]$ and $m(e,f) \in M(d)$ is computable by point 2 of lemma 5.3.6. Hence, since $m(e,f) \Rightarrow g$, the parts of g which is also the value of $El_\Sigma(d,m)$, satisfy the computability requirements in $M(d)$.

2. – (associate judgements) The computability of $El_\Sigma(d, m) \in M(d)$ follows by case 1. Also the computability of $El_\Sigma(d', m') \in M(d')$ follows by case 1, since from the fact that J_2 and J_3 are computable, by lemma 5.3.8, we obtain that $d' \in \Sigma_{x\in B}C(x)$ and $m'(x, y) \in M(\langle x, y\rangle)$ $[x \in B, y \in C(x)]$ are computable judgements. Then, since the judgement $M(d) = M(d')$ is computable, by point 2 of lemma 5.3.6, $El_\Sigma(d', m') \in M(d)$ is computable.
   – (parts) The judgement J_2 is computable, then $d \Rightarrow \langle b, c\rangle$, $d' \Rightarrow \langle b', c'\rangle$ and the judgements $b = b' \in B$ and $c = c' \in C(b)$ are computable. Moreover $b = b' \in B$ and $c = c' \in C(b)$ are c.c.s. for $x \in B$ and $y \in C(x)$ in J_3, respectively, and then $J_3[x \leftarrow b = b', y \leftarrow c = c']$, that is the judgement $m(b, c) = m(b', c') \in M(\langle b, c\rangle)$ is computable. Then, by point 3 of lemma 5.3.6, $m(b, c) = m(b', c') \in M(d)$ is a computable judgement, since, as in the previous point, we can prove that $M(\langle b, c\rangle) = M(d)$. So, if $m(b, c) \Rightarrow g_m$ and $m'(b, c) \Rightarrow g_{m'}$, the parts satisfy the computability requirements in $M(d)$.

- Subcase $\Gamma \neq \emptyset$.

1. – (associate) The computability of the judgement $M(d)$ *type* $[\Gamma]$, associate of the judgement $El_\Sigma(d, m) \in M(d)$ $[\Gamma]$, follows by substitution lemma 5.3.5.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$M(z) \textit{ type } [\Gamma, z \in \Sigma_{x\in B}C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by head substitution lemma 5.3.4;

$$d \in \Sigma_{x\in B}C(x) \ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable and

$$m(x, y) \in M(\langle x, y\rangle) \ [\Gamma, x \in B, y \in C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by head substitution lemma 5.3.4;

$$El_\Sigma(d, m) \in M(d) \ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$M(z)\ \mathit{type}\ [\Gamma, z \in \Sigma_{x\in B}C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by head substitution lemma 5.3.4;

$$d \in \Sigma_{x\in B}C(x)\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable and

$$m(x,y) \in M(\langle x,y\rangle)\ [\Gamma, x \in B, y \in C(x)][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable by head substitution lemma 5.3.4;

$$El_\Sigma(d,m) \in M(d)\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

2. – (associate) The computability of the judgement $El_\Sigma(d,m) \in M(d)\ [\Gamma]$, associate of the judgement $El_\Sigma(d,m) = El_\Sigma(d',m') \in M(d)\ [\Gamma]$, follows by case 1.

   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

   $$M(z)\ \mathit{type}\ [\Gamma, z \in \Sigma_{x\in B}C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by head substitution lemma 5.3.4;

   $$d = d' \in \Sigma_{x\in B}C(x)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable and

   $$m(x,y) = m'(x,y) \in M(\langle x,y\rangle)\ [\Gamma, x \in B, y \in C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable by head substitution lemma 5.3.4;

   $$El_\Sigma(d,m) = El_\Sigma(d',m') \in M(d)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

   $$M(z)\ \mathit{type}\ [\Gamma, z \in \Sigma_{x\in B}C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by head substitution lemma 5.3.4;

$$d = d' \in \Sigma_{x \in B} C(x)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable and

$$m(x,y) = m'(x,y) \in M(\langle x, y \rangle)$$
$$[\Gamma, x \in B, y \in C(x)][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable by head substitution lemma 5.3.4;

$$El_\Sigma(d, m) = El_\Sigma(d', m') \in M(d)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

□

Lemma 5.3.22 (Σ-equality rule)**.** *The Σ-equality rule preserve computability. That is if* $J_1 \equiv M(z)$ *type* $[\Gamma, z \in \Sigma_{x \in B} C(x)], J_2 \equiv b \in B\ [\Gamma], J_3 \equiv c \in C(b)\ [\Gamma], J_4 \equiv \Sigma_{x \in B} C(x)$ *type* $[\Gamma]$ *and* $J_5 \equiv m(x,y) \in M(\langle x, y \rangle)\ [\Gamma, x \in B, y \in C(x)]$ *are computable judgements then*

$$El_\Sigma(\langle b, c \rangle, m) = m(b,c) \in M(\langle b, c \rangle)\ [\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.
  - (associate judgements) J_2, J_3 and J_4 are computable by assumption, thus, lemma 5.3.20 on Σ-introduction rules yields that $\langle b, c \rangle \in \Sigma_{x \in B} C(x)$ is a computable judgement, and hence $El_\Sigma(\langle b, c \rangle, m) \in M(\langle b, c \rangle)$ is computable by the previous lemma on Σ-elimination rules. Moreover, since J_2, J_3 and J_5 are computable, we obtain $J_5[x := b, y := c]$, i.e. the second associate judgement $m(b,c) \in M(\langle b, c \rangle)$, is computable.
  - (parts) By the computability rule, $El_\Sigma(\langle b, c \rangle, m)$ and $m(b,c)$ evaluate into the same canonical element, so all cases are trivial.

- Subcase $\Gamma \neq \emptyset$.
  - (associate) J_2, J_3 and J_4 are computable by assumption, thus, lemma 5.3.20 on Σ-introduction rules yields that $\langle b, c \rangle \in \Sigma_{x \in B} C(x)\ [\Gamma]$ is a computable judgement,

and hence $El_\Sigma(\langle b, c\rangle, m) \in M(\langle b, c\rangle)\ [\Gamma]$ is computable by the previous lemma on Σ-elimination rules.

– (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$M(z)\ type\ [\Gamma, z \in \Sigma_{x\in B} C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

$$m(x, y) \in M(\langle x, y\rangle)\ [\Gamma, x \in B, y \in C(x)][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4;

$$b \in B\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$c \in C(b)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

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

are computable and

$$El_\Sigma(\langle b, c\rangle, m) = m(b, c) \in M(\langle b, c\rangle)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$El_\Sigma(\langle b, c\rangle, m) = m(b, c) \in M(\langle b, c\rangle)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by (substitution $:=$) subcase.

Moreover, since $J_5[x := b, y := c]$ is computable, also $J_5[x := b, y := c][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$ which is

$$m(b, c) \in M(\langle b, c\rangle)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable and then the result follows by transitivity lemma 5.3.13.

□

Lemma 5.3.23 (Eq-formation rules)**.** *The Eq-formation rules preserve computability. That is*

1. *If $J_1 \equiv C\ type\ [\Gamma], J_2 \equiv c \in C\ [\Gamma]$ and $J_3 \equiv d \in C\ [\Gamma]$ are computable judgements then*

$$\mathsf{Eq}(C, c, d)\ type\ [\Gamma]$$

is computable.

2. *If* $J_1 \equiv C = E\,[\Gamma], J_2 \equiv c = e \in C\,[\Gamma]$ *and* $J_3 \equiv d = f \in C\,[\Gamma]$ *are computable judgements then*

$$\mathsf{Eq}(C, c, d) = \mathsf{Eq}(E, e, f)\ \textit{type}\ [\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (evaluation) $\mathsf{Eq}(C, c, d) \Rrightarrow \mathsf{Eq}(C, c, d)$ holds.
   – (correct evaluation) $\mathsf{Eq}(C, c, d) = \mathsf{Eq}(C, c, d)$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J_1 \qquad J_2 \qquad J_3}{\mathsf{Eq}(C, c, d)\ \textit{type}}\ \text{Eq}}{\mathsf{Eq}(C, c, d) = \mathsf{Eq}(C, c, d)}\ \text{refl-ty}$$

   – (parts) The parts are J_1, J_2 and J_3 which are assumed to be computable.

2. – (associate judgements) The judgements $C\ \textit{type}$, $E\ \textit{type}$, $c \in C$, $e \in C$, $d \in C$ and $f \in C$, associates of J_1, J_2 and J_3 are computable by assumption, and, by point 1 of lemma 5.3.6 also $e \in E$ and $f \in E$ are computable. Hence the result follows by case 1.
   – (parts) The parts are J_1, J_2 and J_3 which are assumed to be computable.

• Subcase $\Gamma \neq \emptyset$.

1. – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

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

$$c \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$d \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable;

$$\mathsf{Eq}(C, c, d)\ \textit{type}\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$C \text{ type } [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

$$c \in C\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$
$$d \in C\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

are computable;

$$\mathsf{Eq}(C, c, d) \textit{ type } [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then ($\Gamma = \emptyset$) subcase.

2. – (associate) The judgements C type $[\Gamma]$, $c \in C\,[\Gamma]$, $d \in C\,[\Gamma]$ associate respectively of J_1, J_2 and J_3 are computable. Hence the result follows by case 1.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$C = E\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$c = e\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
$$d = f\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable;

$$\mathsf{Eq}(C, c, d) = \mathsf{Eq}(E, e, f)\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$C = E\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

$$c = e\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$
$$d = f\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

   are computable;

$$\mathsf{Eq}(C, c, d) = \mathsf{Eq}(E, e, f)\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

   has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

□

Lemma 5.3.24 (Eq-introduction rules)**.** *The Eq-introduction rules preserve computability. That is*

1. *If* $J \equiv c \in C\,[\Gamma]$ *is a computable judgement then*

$$\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)\,[\Gamma]$$

is computable.

2. *If* $J \equiv c = d \in C\,[\Gamma]$ *is a computable judgement then*

$$\mathsf{eq}_C(c) = \mathsf{eq}_C(d) \in \mathsf{Eq}(C, c, c)\,[\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.

  1. – (associate judgements) The judgement *C type*, associate of *J* is computable by assumption, hence by point 1 of the previous lemma also $\mathsf{Eq}(C, c, c)$ is computable.
     – (evaluation) $\mathsf{eq}_C(c) \Rightarrow \mathsf{eq}_C(c)$ holds.
     – (correct evaluation) $\mathsf{eq}_C(c) = \mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J}{\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)}\,\text{I-Eq}}{\mathsf{eq}_C(c) = \mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)}\,\text{refl-ty}$$

     – (parts) The part is $c = c \in C$ which is computable by lemma 5.3.2 about reflexivity on elements.

  2. – (associate judgements) The judgements $c \in C$ and $d \in C$, associates of *J*, are computable. Then, by case 1, also $\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)$ and $\mathsf{eq}_C(d) \in \mathsf{Eq}(C, d, d)$ are computable.
     – (parts) The part is *J* which is assumed to be computable.

- Subcase $\Gamma \neq \emptyset$.

1. – (associate) The judgement $C\ \mathit{type}\ [\Gamma]$, associate of J is computable by assumption, hence by point 1 of the previous lemma also $\mathsf{Eq}(C, c, c)\ \mathit{type}\ [\Gamma]$ is computable.

   – (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

   $$c \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable;

   $$\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

   $$c \in C\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

   are computable;

   $$\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

2. – (associate) The judgement $c \in C\,[\Gamma]$, which is the associate of J, is computable by definition, hence the result follows by case 1.

   – (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

   $$c = d \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable;

   $$\mathsf{eq}_C(c) = \mathsf{eq}_C(d) \in \mathsf{Eq}(C, c, c)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

   $$c = d \in C\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

are computable;

$$\mathsf{eq}_C(c) = \mathsf{eq}_C(d) \in \mathsf{Eq}(C, c, c)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

□

Lemma 5.3.25 (Eq*-elimination rule)**.** *The Eq*-elimination rule preserves computability. That is if the judgements* $J_1 \equiv p \in \mathsf{Eq}(C, c, d)\ [\Gamma], J_2 \equiv C\ type\ [\Gamma], J_3 \equiv c \in C\,[\Gamma]$ *and* $J_4 \equiv d \in C\,[\Gamma]$ *are computable then the judgement*

$$c = d \in C\,[\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$. The result follows by definition of computability of J.
- Subcase $\Gamma \neq \emptyset$.
  - (associate) The associate judgement is J_3 which is computable by assumption.
  - (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

    $$p \in \mathsf{Eq}(C, c, d)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    $$C\ type\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
    $$c \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$
    $$d \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    are computable;

    $$c = d \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

    $$c = d \in C\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    is computable by (substitution :=) subcase. Moreover

    $$d \in C\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

    is computable and then the result follows by transitivity lemma 5.3.13.

□

Lemma 5.3.26 (Eq-equality rules)**.** *The Eq-equality rule preserves computability. That is if the judgement* $J \equiv p \in \mathsf{Eq}(C, c, d)\ [\Gamma]$ *is computable then the judgement*

$$p = \mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, d)\ [\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.
  - (associated judgements) The first associate judgement is J which is computable by assumption.

    Since J is computable, then $c = d \in C$, $c \in C$ and C *type* are computable; then, by point 1 of lemma on Eq-introduction rules 5.3.24, the judgement $\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, c)$ is computable. Moreover, by reflexivity lemmas 5.3.3 and 5.3.2, $C = C$ and $c = c \in C$ are computable, thus, by lemma on Eq-formation rules 5.3.23, also $\mathsf{Eq}(C, c, c) = \mathsf{Eq}(C, c, d)$ *type* is computable. Hence, by point 2 of lemma 5.3.6, $\mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, d)$ is computable.
  - (parts) The judgement $c = d \in C$ is computable by assumption.

- Subcase $\Gamma \neq \emptyset$.
  - (associate) The associate judgement is J which is computable by assumption.
  - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

    $$p \in \mathsf{Eq}(C, c, d)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    is computable;

    $$p = \mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, d)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

    has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

    $$p \in \mathsf{Eq}(C, c, d)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable;

$$p = \mathsf{eq}_C(c) \in \mathsf{Eq}(C, c, d)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

□

Lemma 5.3.27 (Qtr-formation rules)**.** *The Qtr-formation rules preserve computability. That is*

1. *If* $J \equiv A\ type\ [\Gamma]$ *is a computable judgement then*

$$A/\top\ type\ [\Gamma]$$

is computable.

2. *If* $J \equiv A = B\ [\Gamma]$ *is a computable judgement then*

$$A/\top = B/\top\ [\Gamma]$$

is computable.

Proof. By induction on the computational complexity α of J.

- Subcase $\Gamma = \emptyset$.

  1. – (evaluation) $A/\top \Rrightarrow A/\top$ holds.
     – (correct evaluation) $A/\top = A/\top$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J}{A/\top\ type}\ \text{Qtr}}{A/\top = A/\top}\ \text{refl-ty}$$

     – (parts) The part is J which is assumed to be computable.

  2. – (associate judgements) The judgements $A\ type$ and $B\ type$, associates of J are computable by assumption, hence the result follows by case 1.
     – (parts) The part is J which is assumed to be computable.

- Subcase $\Gamma \neq \emptyset$.

  1. – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$A \; type \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     is computable by assumption;

     $$A/\top \; type \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
     – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$A \; type \; [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

     is computable by assumption;

     $$A/\top \; type \; [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

     has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

  2. – (associate) The judgement $A \; type \; [\Gamma]$ associate of J is computable. Hence the result follows by case 1.
     – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$A = B \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     is computable by assumption;

     $$A/\top = B/\top \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
     – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$A = B \; [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

     is computable by assumption;

     $$A/\top = B/\top \; [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

□

Lemma 5.3.28 (Qtr-introduction rules)**.** *The Qtr-introduction rules preserve computability. That is*

1. *If* $J \equiv a \in A\ [\Gamma]$ *is a computable judgement then*

$$[a] \in A/\top\ [\Gamma]$$

is computable.

2. *If* $J_1 \equiv a \in A\ [\Gamma]$ *and* $J_2 \equiv b \in A\ [\Gamma]$ *are computable judgements then*

$$[a] = [b] \in A/\top\ [\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (associate judgements) The judgement A *type*, associate of J is computable by assumption, hence by point 1 of the previous lemma also $A/\top$ *type* is computable.
   – (evaluation) $[a] \Rightarrow [a]$ holds.
   – (correct evaluation) $[a] = [a] \in A/\top$ is derivable with the following derivation tree

$$\dfrac{\dfrac{J}{[a] \in A/\top}\ \text{I-Qtr}}{[a] = [a] \in A/\top}\ \text{refl-tm}$$

   – (parts) The part is J which is assumed to be computable.

2. – (associate judgements) The result follows from case 1.
   – (parts) The judgements $a \in A$ and $b \in B$ are computable by assumption.

- Subcase $\Gamma \neq \emptyset$.

  1. – (associate) The associate judgement of $J, A\ \mathit{type}\ [\Gamma]$, is computable by assumption; then, by previous lemma, also $A/\top\ \mathit{type}\ [\Gamma]$ is computable.
     – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     is computable;

     $$[a] \in A/\top\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
     – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ and $a'_1 \in A_1, \ldots, a'_n \in A_n$ are c.c.s. fitting with Γ, and so

     $$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     and

     $$a \in A\ [\Gamma][x_1 := a'_1, \ldots, x_n := a'_n]$$

     are computable;

     $$[a] \in A/\top\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

     has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

  2. – (associate) The result follows by case 1.
     – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

     $$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     and

     $$b \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     are computable;

     $$[a] = [b] \in A/\top\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

     has empty context, then it is computable by $(\Gamma = \emptyset)$.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$[a] = [b] \in A/\top\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by (substitution $:=$) subcase; then

$$[b] \in A/\top\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable by case 1. Hence

$$[a] = [b] \in A/\top\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable by transitivity lemma 5.3.13.

□

Lemma 5.3.29 (Qtr-elimination rules)**.** *The Qtr-elimination rules preserve computability. That is*

1. *If the judgements* $J_1 \equiv L(z)$ *type* $[\Gamma, z \in A/\top], J_2 \equiv p \in A/\top\,[\Gamma], J_3 \equiv l(x) \in L([x])\,[\Gamma, x \in A]$ *and* $J_4 \equiv l(x) = l(y) \in L([x])\,[\Gamma, x \in A, y \in A]$ *are computable then the judgement*

$$El_Q(l, p) \in L(p)\,[\Gamma]$$

is computable.

2. *If the judgements* $J_1 \equiv L(z)$ *type* $[\Gamma, z \in A/\top], J_2 \equiv p = p' \in A/\top\,[\Gamma], J_3 \equiv l(x) = l'(x) \in L([x])\,[\Gamma, x \in A]$ *and* $J_4 \equiv l(x) = l(y) \in L([x])\,[\Gamma, x \in A, y \in A]$ *are computable then the judgement*

$$El_Q(l, p) = El_Q(l', p') \in L(p)\,[\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (associate judgements) The computability of the judgement $L(p)$ *type*, associate of the judgement $El_Q(l, p) = El_Q(l', p') \in L(p)$, follows by assumption.

– (evaluation) J_2 is computable and $A/\top \Rightarrow A/\top$, then $p \Rightarrow [a]$ and the judgement $a \in A$ is computable; thus it is a c.c.s. fitting with $x \in A$; therefore $J_3[x := a]$, which is $l(a) \in L([a])$, is a computable judgement. Hence $l(a) \Rightarrow g$ and the result follows by using the computation rule.

– (correct evaluation) Since J_2 is computable, we know that there exists a derivation of the judgement $A/\top$ *type*; hence $x \in A/\top$ is a correct assumption , and $p = [a] \in A/\top$ is derivable. Let Π_1 be the following derivation

$$\dfrac{p = [a] \in A/\top \qquad \dfrac{J_1 \quad x \in A/\top\,[x \in A/\top] \quad J_3 \quad J_4}{El_Q(l,x) \in L(x)\,[x \in A/\top]}\,\text{E-Qtr}}{El_Q(l,p) = El_Q(l,[a]) \in L(p)}\,\text{sub-tm-eq}$$

then Π_2 is

$$\dfrac{\Pi_1 \qquad \dfrac{p = [a] \in A/\top \quad J_1}{L(p) = L([a])}\,\text{sym-ty}}{El_Q(l,p) = El_Q(l,[a]) \in L([a])}\,\text{conv-eq}$$

Since J_2 is computable, so is $a \in A$ and then $J_3[x := a]$ is computable.
Thus the judgements $l(a) = g \in L([a])$ and $a \in A$ are derivable. Let Π_3 be

$$\dfrac{\Pi_2 \qquad \dfrac{\dfrac{J_1 \quad a \in A \quad J_3 \quad J_4}{El_Q(l,[a]) = l(a) \in L([a])}\,\text{C-Qtr} \qquad l(a) = g \in L([a])}{El_Q(l,[a]) = g \in L([a])}\,\text{trans-tm}}{El_Q(l,p) = g \in L([a])}\,\text{trans-tm}$$

Hence

$$\dfrac{\Pi_3 \qquad \dfrac{\dfrac{p = [a] \in A/\top \quad J_1}{L(p) = L([a])}\,\text{sym-ty}}{L([a]) = L(p)}\,\text{conv-eq}}{El_Q(l,p) = g \in L(p)}$$

– (parts) The part is J which is assumed to be computable.

2. – (associate judgements) The computability of $El_Q(l,p) \in L(p)$ follows by case 1. Then, since J_3 is computable also $l(y) = l(y') \in L([y]\ [y \in A]$; then, by weakening lemma 5.3.1, by Qtr-introduction lemma 5.3.28, by point 3 of lemma 5.3.6, both $l(x) = l(x') \in L([x])\ [x \in A, y \in A]$ and $l(y) = l(y') \in L([x])[x \in A, y \in A]$ are computable, so, by point 1 of 5.3.6, $l'(x) = l'(y) \in L([x])\ [x \in A, y \in A]$. The computability of $El_Q(l',p') \in L(p')$, thus, follows by case 1, since from the fact that J_2 and J_3 are computable, by lemma 5.3.8, we obtain that $p' \in A/\top$ and $l'(x) \in L([x])\ [x \in A]$ are computable

judgements. Then, since the judgement $L(p) = L(p')$ is computable, by point 2 of lemma 5.3.6, $El_Q(l', p') \in L(p)$ is computable.

– (parts) The judgement J_2 is computable, then $p \Rightarrow [a]$, $p' \Rightarrow [a']$ and the judgements $a \in A$ and $a' \in A$ are computable. Moreover they are c.c.s. for $x \in A, y \in A$ in J_4, and then $J_4[x := a, y := a']$, that is the judgement $l(a) = l(a') \in L([a])$ is computable; then $J_3[x := a']$, that is $l(a') = l'(a') \in L([a'])$, is computable, hence, by point 2 of Qtr-introduction lemma 5.3.28 and by point 3 of lemma 5.3.6, also $l(a') = l'(a') \in L([a])$ is computable. By transitivity lemma, $l(a) = l'(a') \in L([a])$ is a computable judgement. Then, by point 3 of lemma 5.3.6, $l(a) = l'(a') \in L(p)$ is a computable judgement, since, as in the previous point, we can prove that $L([a]) = L(p)$ is a computable judgement. So, if $l(a) \Rightarrow g_l$ and $l'(a') \Rightarrow g_{l'}$, the parts of g_l and $g_{l'}$ satisfy the computability requirements.

• Subcase $\Gamma \neq \emptyset$.

1. – (associate) The computability of the judgement $L(p)$ *type* $[\Gamma]$, associate of the judgement $El_Q(l, p) \in L(p)$ $[\Gamma]$, follows by substitution lemma 5.3.5.

– (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z) \textit{ type } [\Gamma, z \in A/\top][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x) \in L([x]) \; [\Gamma, x \in A][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x) = l(y) \in L([x]) \; [\Gamma, x \in A, y \in A][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4 and

$$p \in A/\top \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable;

$$El_Q(l, p) \in L(p) \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$L(z) \textit{ type } [\Gamma, z \in A/\top][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x) \in L([x])\ [\Gamma, x \in A][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$
$$l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4;

$$p \in A/\top\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

is computable;

$$El_Q(l, p) \in L(p)\ [\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

2. – (associate) The computability of the judgement $El_Q(l, p) \in L(p)\ [\Gamma]$, associate of the judgement $El_Q(l, p) = El_Q(l', p') \in L(p)\ [\Gamma]$, follows by case 1.

– (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z)\ \mathit{type}\ [\Gamma, z \in A/\top][x_1 := a_1, \ldots, x_n := a_n]$$
$$l(x) = l'(x) \in L([x])\ [\Gamma, x \in A][x_1 := a_1, \ldots, x_n := a_n]$$
$$l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4 and

$$p = p' \in A/\top\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable;

$$El_Q(l, p) = El_Q(l', p') \in L(p)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$L(z)\ \mathit{type}\ [\Gamma, z \in A/\top][x_1 := a_1, \ldots, x_n := a_n]$$
$$l(x) = l'(x) \in L([x])\ [\Gamma, x \in A][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n]$$
$$l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4 and

$$p = p' \in A/\top\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable;

$$El_Q(l, p) = El_Q(l', p') \in L(p)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

□

Lemma 5.3.30 (Qtr-equality rule)**.** *The Qtr-equality rule preserves computability, i.e., if the judgements* $J_1 \equiv L(z)$ *type* $[\Gamma, z \in A/\top], J_2 \equiv a \in A\ [\Gamma], J_3 \equiv l(x) \in L([x])\ [\Gamma, x \in A]$ *and* $J_4 \equiv l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A]$ *are computable then the judgement*

$$El_Q(l, [a]) = l(a) \in L([a])\ [\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.
  - (associated judgements) J_2 is a computable judgement by assumption. Thus, lemma on Qtr-introduction rules yields that $[a] \in A/\top$ is a computable judgement, and hence $El_Q(l, [a]) \in L([a])$ is computable by the previous lemma on Qtr-elimination rules. Moreover, since J_3 is computable, we obtain that $J_3\ [x := a]$, i.e. the second associate judgement $l(a) \in L([a])$, is computable.
  - (parts) Since $El_Q(l, [a])$ and $l(a)$ evaluate into the same canonical element, the computability of the judgement $El_Q(l, [a]) = l(a) \in L([a])$ follows from the computability of the associated judgements.

- Subcase $\Gamma \neq \emptyset$.
  - (associate) J_2 is a computable judgement by assumption. Thus, lemma on Qtr-introduction rules yields that $[a] \in A/\top\ [\Gamma]$ is a computable judgement, and hence $El_Q(l, [a]) \in L([a])\ [\Gamma]$ is computable by the previous lemma on Qtr-elimination rules.

– (substitution :=) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z)\ \mathit{type}\ [\Gamma, z \in A/\top][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x) \in L([x])\ [\Gamma, x \in A][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x) = l(y) \in L([x])\ [\Gamma, x \in A, y \in A][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4 and

$$a \in A\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by assumption;

$$El_Q(l, [a]) = l(a) \in L([a])\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$El_\Sigma(\langle b, c\rangle, m) = m(b, c) \in M(d)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by (substitution :=) subcase.

Moreover, since $J_5[x := b, y := c]$ is computable, also $J_5[x := b, y := c][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$ which is

$$m(b, c) \in M(d)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable and then the result follows by transitivity lemma, 5.3.13.

□

5.4 The computability theorem

Now we can state our main theorem about computability/operational semantics: 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 5.4.1 (Computability theorem)**.** *Let J be any derivable judgement. Then J is computable.*

Proof. If J is derivable, then a proof Π of J is built only with the rules of $\mathcal{T}_{reg}$. All the rules of $\mathcal{T}_{reg}$ preserve computability, then J is computable. □

A normal form theorem is a theorem that states that any proof can be transformed in a new one with the same conclusion but enjoying stronger structure properties. The computability theorem does not regard derivations but still is strongly related to normal form theorems as the following definitions will clarify.

Definition 5.4.1 (Canonical proof)**.** *A proof* Π *of a judgement* J *is canonical if*

1. $J \equiv A$ *type or* $J \equiv A = B$ *and the last inference step in* Π *is a formation rule.*
2. $J \equiv a \in A$ *or* $J \equiv a = b \in A$ *and the last inference step in* Π *is 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.

Definition 5.4.2 (Canonical form)**.** *Let* J *be a closed judgement. Then*

- *if* $J \equiv A$ *type and* $A \Rightarrow G_A$ *then the canonical form of* J *is* G_A *type;*
- *if* $J \equiv A = B$ *and* $A \Rightarrow G_A$ *and* $B \Rightarrow G_B$ *then the canonical form of* J *is* $G_A = G_B$*;*
- *if* $J \equiv a \in A$ *and* $a \Rightarrow g_a$ *and* $A \Rightarrow G_A$ *then the canonical form of* J *is* $g_a \in G_A$*;*
- *if* $J \equiv a = b \in A$ *and* $a \Rightarrow g_a$ *and* $b \Rightarrow g_b$ *and* $A \Rightarrow G_A$ *then the canonical form of* J *is* $g_a = g_b \in G_A$*.*

Corollary 5.4.1 (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 $A \Rightarrow G_A$ 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 $A \Rightarrow G_A$ and $B \Rightarrow G_B$ 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 $J \equiv a \in A$ and $a \Rightarrow g_a$ and $A \Rightarrow G_A$ then the canonical form of J is $g_a \in G_A$ and the computability of J yields that $A = G_A$ and $a = g_a \in A$ are derivable judgements.
- if $J \equiv a = b \in A$ and $a \Rightarrow g_a, b \Rightarrow g_b$ and $A \Rightarrow G_A$ then the canonical form of J is $g_a = g_b \in G_A$ and the computability of J assures that $A = G_A, a = g_a \in A$ and $b = g_b \in A$ are derivable judgements.

6

The canonical form theorem for $\mathcal{T}_{areg}$

The proof of the canonical-form theorem for $\mathcal{T}_{areg}$ follows the same structure as the proof of the canonical-form theorem for $\mathcal{T}_{reg}$ in the previous chapter, paying attention to when the proofs involve case analysis, where also the type of the natural numbers must be considered. Furthermore, the preservation of computability must be proved for all the rules of the type of natural numbers, which we report below.

Lemma 6.0.1 (nat-formation rule)**.** *The nat-formation rule preserves computability. That is the judgement*

$$N\,type\,[\Gamma]$$

is computable.

Proof.

- Subcase $\Gamma = \emptyset$.
  - (evaluation) $N \Rrightarrow N$ holds.
  - (correct evaluation) $N = N$ is derivable with the following derivation tree

$$\dfrac{\dfrac{\Gamma\,cont}{N\,type}\,\text{nat}}{N = N}\,\text{refl-ty}$$

  - (parts) There is no condition to prove.

- Subcase $\Gamma \neq \emptyset$.

  - (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

    $$N\,type\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n] \equiv N\,type$$

    is computable by $(\Gamma = \emptyset)$ subcase.
  - (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

    $$N\,type\,[\Gamma][x_1 \leftarrow a_1 = a'_1, \ldots, x_n \leftarrow a_n = a'_n] \equiv N = N$$

    is computable, since it is derivable and the associate judgement $N\ type$ is computable.

□

Lemma 6.0.2 (nat-introduction rules)**.** *The nat-introduction rule preserves computability. That is*

1. *The judgement*

   $$0 \in N\,[\Gamma]$$

   is computable

2. *If $J \equiv n \in N\,[\Gamma]$ is a computable judgement then*

   $$\mathsf{s}(n) \in N\,[\Gamma]$$

   is computable.

3. *If $J \equiv m = n \in N\,[\Gamma]$ is a computable judgement then*

   $$\mathsf{s}(m) = \mathsf{s}(n) \in N\,[\Gamma]$$

   is computable.

Proof.

- Subcase $\Gamma = \emptyset$.

1. – (associate judgement) The computability of the associate judgement is immediate by the previous lemma on nat-formation rule.
   – (evaluation) $0 \Rightarrow 0$ holds.
   – (correct evaluation) $0 = 0 \in N$ is derivable with the following derivation tree
   $$\dfrac{\dfrac{\Gamma \; cont}{0 \in N}\,\text{I}_1\text{-nat}}{0 = 0 \in N}\,\text{refl-tm}$$
   – (parts) No condition to show.

2. – (associate judgements) The computability of the associate judgement is immediate by the previous lemma on nat-formation rule.
   – (evaluation) $\mathsf{s}(n) \Rightarrow \mathsf{s}(n)$ holds.
   – (correct evaluation) $\mathsf{s}(n) = \mathsf{s}(n) \in N$ is derivable by using first the second nat-introduction rule and then the reflexivity on elements rule.
   $$\dfrac{\dfrac{J}{\mathsf{s}(n) \in N}\,\text{I}_2\text{-nat}}{\mathsf{s}(n) = \mathsf{s}(n) \in N}\,\text{refl-tm}$$
   – (parts) The part is J which is assumed to be computable.

3. – (associate judgements) The judgements $m \in N$ and $n \in N$, associates of J, are computable by definition. Hence the result follows from case 2.
   – (parts) The part is J which is assumed to be computable.

• Subcase $\Gamma \neq \emptyset$.

1. – (associate) The judgement $N \; type \; [\Gamma]$ is computable for the previous lemma on nat-formation rule.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then
   $$0 \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n] \equiv 0 \in N$$
   is computable by $(\Gamma = \emptyset)$ subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then
   $$0 \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n'] \equiv 0 = 0 \in N$$

is computable, since it is derivable and the associate judgement $0 \in N$ is computable.

2. – (associate) The judgement $N\ type\ [\Gamma]$ is computable for the previous lemma on nat-formation rule.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$n \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable;

$$\mathsf{s}(n) \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$n \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

   are computable;

$$\mathsf{s}(n) \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

3. – (associate) The judgement $m \in N\,[\Gamma]$, associate of J is computable. Hence the result follows by case 2.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$m = n \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable;

$$\mathsf{s}(m) = \mathsf{s}(n) \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$m = n \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable;

$$\mathsf{s}(m) = \mathsf{s}(n) \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by $(\Gamma = \emptyset)$.

□

Lemma 6.0.3 (nat-elimination rules)**.** *The nat-elimination rules preserve computability. That is*

1. *If* $J_1 \equiv L(z)$ *type* $[\Gamma, z \in N]$, $J_2 \equiv n \in N\,[\Gamma]$, $J_3 \equiv a \in L(0)\,[\Gamma]$ *and* $J_4 \equiv l(x,y) \in L(\mathsf{s}(x))\,[\Gamma, x \in N, y \in L(x)]$ *are computable judgements then*

$$El_N(a, l, n) \in L(n)\,[\Gamma]$$

is computable.

2. *If* $J_1 \equiv L(z)$ *type* $[\Gamma, z \in N]$, $J_2 \equiv n = n' \in N\,[\Gamma]$, $J_3 \equiv a = a' \in L(0)\,[\Gamma]$ *and* $J_4 \equiv l(x,y) = l'(x,y) \in L(\mathsf{s}(x))\,[\Gamma, x \in N, y \in L(x)]$ *are computable judgements then*

$$El_N(a, l, n) = El_N(a', l', n') \in L(n)\,[\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (associate judgements) The computability of the judgement $L(n)$ *type*, associate of the judgement $El_N(a, l, n) \in L(n)$, follows by assumption.
   – (evaluation) J_2 is computable and $N \Rrightarrow N$, then either $n \Rightarrow 0$ or $n \Rightarrow \mathsf{s}(m)$.
     * If $n \Rightarrow 0$, then, since J_3 is computable, we have that $a \Rightarrow g$; hence the result follows by the computation rule.
     * If $n \Rightarrow \mathsf{s}(m)$, then $m \in N$ and $El_N(a, l, m) \in L(m)$ are computable and it is a c.c.s. fitting with $x \in N, y \in L(x)$; therefore $J_3[x := m, y := El_N(a, l, m)]$, which is $l(m, El_N(a, l, m)) \in L(\mathsf{s}(m))$, is a computable judgement. Hence $l(m, El_N(a, l, m)) \Rightarrow g$ and the result follows by using the computation rule.

– (correct evaluation) Since J_2 is computable, we know that there exists a derivation of the judgement $N\ type$; hence $x \in N$ is a correct assumption, and either $n = 0$ or $n = \mathsf{s}(m) \in N$ are derivable.

  * If $n = 0$ then, let Π_1 be

$$\dfrac{n = 0 \in N \qquad \dfrac{J_1 \qquad x \in N\,[x \in N] \qquad J_3 \qquad J_4}{El_N(a,l,n) \in L(x)\,[x \in N]}\ \text{E-nat}}{El_N(a,l,n) = El_N(a,l,0) \in L(n)}\ \text{sub-tm-eq}$$

    then Π_2 is

$$\dfrac{\Pi_1 \qquad \dfrac{n = 0 \in N \qquad J_1}{L(n) = L(0)}\ \text{sub-ty-eq}}{El_N(a,l,n) = El_N(a,l,0) \in L(0)}\ \text{conv-eq}$$

    Since J_3 is computable, the judgement $a = g \in L(0)$ is derivable. Let Π_3 be

$$\dfrac{\dfrac{J_1 \qquad J_3 \qquad J_4}{El_N(a,l,0) = a \in L(0)}\ \text{C}_1\text{-nat} \qquad a = g \in L(0)}{El_N(a,l,0) = g \in L(0)}\ \text{trans-tm}$$

    then

$$\dfrac{\Pi_2 \qquad \Pi_3}{El_N(a,l,n) = g \in L(0)}\ \text{trans-tm}$$

    Hence

$$\dfrac{El_N(a,l,n) = g \in L(0) \qquad \dfrac{\dfrac{n = 0 \in N \qquad J_1}{L(n) = L(0)}\ \text{sub-ty-eq}}{L(0) = L(n)}\ \text{sym-ty}}{El_N(a,l,n) = g \in L(n)}\ \text{conv-eq}$$

  * If $n = \mathsf{s}(m) \in N$, then let Π_1 be

$$\dfrac{n = \mathsf{s}(m) \in N \qquad \dfrac{J_1 \qquad x \in N\,[x \in N] \qquad J_3 \qquad J_4}{El_N(a,l,n) \in L(x)\,[x \in N]}\ \text{E-nat}}{El_N(a,l,n) = El_N(a,l,\mathsf{s}(m)) \in L(n)}\ \text{sub-tm-eq}$$

    then Π_2 is

$$\dfrac{\Pi_1 \qquad \dfrac{n = \mathsf{s}(m) \in N \qquad J_1}{L(n) = L(\mathsf{s}(m))}\ \text{sub-ty-eq}}{El_N(a,l,n) = El_N(a,l,\mathsf{s}(m)) \in L(\mathsf{s}(m))}\ \text{conv-eq}$$

    Since J_2 is computable, so is $m \in N$ and then $J_4[x := m, y := El_N(a,l,m)]$ is computable. Thus the judgements $l(m, El_N(a,l,m)) = g \in L(\mathsf{s}(m))$ and $m \in N$ are derivable. Let Π_3 be

$$\dfrac{\dfrac{J_1 \quad m \in N \quad J_3 \quad J_4}{El_N(a,l,\mathsf{s}(m)) = l(m, El_N(a,l,m)) \in L(\mathsf{s}(m))}\,\text{C}_2\text{-nat} \qquad l(m, El_N(a,l,m)) = g \in L(\mathsf{s}(m))}{El_N(a,l,\mathsf{s}(m)) = g \in L(\mathsf{s}(m))}\,\text{trans-tm}$$

then

$$\dfrac{\Pi_2 \qquad \Pi_3}{El_N(a,l,n) = g \in L(\mathsf{s}(m))}\,\text{trans-tm}$$

Hence

$$\dfrac{El_N(a,l,n) = g \in L(\mathsf{s}(m)) \qquad \dfrac{\dfrac{n = \mathsf{s}(m) \in N \qquad J_1}{L(n) = L(\mathsf{s}(m))}\,\text{sub-ty-eq}}{L(\mathsf{s}(m)) = L(n)}\,\text{sym-ty}}{El_N(a,l,n) = g \in L(n)}\,\text{conv-eq}$$

- (parts) Since J_2 is computable we know that either $n \Rightarrow 0$ or $n \Rightarrow \mathsf{s}(m)$.
  - If $n \Rightarrow 0$, then $0 \in N$ is computable by the previous lemma on nat-introduction rules. Hence we can deduce that the judgement $n = 0 \in N$ is computable. Therefore, since J_1 is computable, we obtain that the judgement $L(0) = L(n)$ is a computable judgement. Then, since J_3 is computable, so is $a \in L(n)$ by point 2 of lemma 5.3.6. Hence, since $a \Rightarrow g$, the parts of g which is also the value of $El_N(a,l,n)$, satisfy the computability requirements in $L(n)$.
  - If $n \Rightarrow \mathsf{s}(m)$, then $m \in N$ and $\mathsf{s}(m) \in N$ is computable by the previous lemma on nat-introduction rules. Hence we can deduce that the judgement $n = \mathsf{s}(m) \in N$ is computable. Therefore, since J_1 is computable, we obtain that the judgement $L(\mathsf{s}(m)) = L(n)$ is a computable judgement. Then, since J_4 is computable, so is $J_4[x := m, y := El_N(a,l,m)]$ and $l(m, El_N(a,l,m)) \in L(n)$ is computable by point 2 of lemma 5.3.6. Hence, since $l(m, El_N(a,l,m)) \Rightarrow g$, the parts of g which is also the value of $El_N(a,l,n)$, satisfy the computability requirements in $L(n)$.

2. - (associate judgements) The computability of $El_N(a,l,n) \in L(n)$ follows by case 1. Also the computability of $El_N(a',l',n') \in L(n')$ follows by case 1, since from the fact that J_2, J_3 and J_4 are computable, by point 4 of lemma 5.3.6, we obtain that $n' \in N, J_3 \equiv a \in L(0)\ [\Gamma]$ and $l'(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]$ are computable judgements. Then, since the judgement $L(n) = L(n')$ is computable, by point 2 of lemma 5.3.6, $El_N(a',l',n') \in L(n)$ is computable.
   - (parts) The judgement J_2 is computable, then either $n \Rightarrow 0$ and $n' \Rightarrow 0$ or $n \Rightarrow \mathsf{s}(m)$ and $n' \Rightarrow \mathsf{s}(m')$

* If $n \Rightarrow 0$ and $n' \Rightarrow 0$, then, by point 3 of lemma 5.3.6, $a = a' \in L(n)$ is a computable judgement, since, as in the previous point, we can prove that $L(0) = L(n)$. So, if $a \Rightarrow g$ and $a' \Rightarrow g'$, the parts satisfy the computability requirements in $L(n)$.
* If $n \Rightarrow \mathsf{s}(m)$ and $n' \Rightarrow \mathsf{s}(m')$ then $m = m' \in N$ is computable. Moreover $m = m' \in N$ and $El_N(a, l, m) = El_N(a', l', m') \in L(m)$ are c.c.s. for $x \in N$ and $y \in L(x)$ in J_4, respectively, and then $J_4[x \leftarrow m = m', y \leftarrow El_N(a, l, m) = El_N(a', l', m')]$, that is the judgement $l(m, El_N(a, l, m)) = l'(m', El_N(a', l', m')) \in L(\mathsf{s}(m))$ is computable. Then, by point 3 of lemma 5.3.6, $l(m, El_N(a, l, m)) = l'(m', El_N(a', l', m')) \in L(n)$ is a computable judgement, since, as in the previous point, we can prove that $L(\mathsf{s}(m)) = L(n)$. So, if $l(m, El_N(a, l, m)) \Rightarrow g_m$ and $l'(m', El_N(a', l', m')) \Rightarrow g_{m'}$, the parts satisfy the computability requirements in $L(n)$.

- Subcase $\Gamma \neq \emptyset$.

1. – (associate) The computability of the judgement $L(n)$ *type* $[\Gamma]$, associate of the judgement $El_N(a, l, n) \in L(n)$ $[\Gamma]$, follows by substitution lemma 5.3.5.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z) \textit{ type } [\Gamma, z \in N][x_1 := a_1, \ldots, x_n := a_n]$$

   and

$$l(x, y) \in L(\mathsf{s}(x)) \; [\Gamma, x \in N, y \in L(x)][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable by head substitution lemma 5.3.4,

$$n \in N \, [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$a \in L(0) \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable;

$$El_N(a, l, n) \in L(n) \; [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a'_1 \in A_1, \ldots, a_n = a'_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a

c.c.s. fitting with Γ, and so

$$L(z)\ \mathit{type}\ [\Gamma, z \in N][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$l(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

are computable by head substitution lemma 5.3.4,

$$n \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

and

$$a \in L(0)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

are computable;

$$El_N(a,l,n) \in L(n)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

2. – (associate) The computability of the judgement $El_N(a,l,n) \in L(n)\ [\Gamma]$, associate of the judgement $El_N(a,l,n) = El_N(a',l',n') \in L(n)\ [\Gamma]$, follows by case 1.

   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z)\ \mathit{type}\ [\Gamma, z \in N][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$l(x,y) = l'(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4,

$$n = n' \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

$$a = a' \in L(0)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$El_N(a,l,n) = El_N(a',l',n') \in L(n)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.

– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$L(z)\ \mathit{type}\ [\Gamma, z \in N][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$l(x,y) = l'(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

are computable by head substitution lemma 5.3.4,

$$n = n' \in N\,[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

and

$$a = a' \in L(0)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

are computable;

$$El_N(a,l,n) = El_N(a',l',n') \in L(n)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

has empty context, then it is computable by ($\Gamma = \emptyset$) subcase.

□

Lemma 6.0.4 (nat-equality rules)**.** *The nat-equality rules preserve computability. That is*

1. *If* $J_1 \equiv L(z)\ \mathit{type}\ [\Gamma, z \in N]$, $J_2 \equiv a \in L(0)\ [\Gamma]$ *and* $J_3 \equiv l(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]$ *are computable judgements then*

$$El_N(a,l,0) = a \in L(0)\ [\Gamma]$$

is computable.

2. *If* $J_1 \equiv L(z)\ \mathit{type}\ [\Gamma, z \in N]$, $J_2 \equiv n \in N\,[\Gamma]$, $J_3 \equiv a \in L(0)\ [\Gamma]$ *and* $J_4 \equiv l(x,y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)]$ *are computable judgements then*

$$El_N(a,l,\mathsf{s}(n)) = l(n, El_N(a,l,n)) \in L(\mathsf{s}(n))\ [\Gamma]$$

is computable.

Proof. • Subcase $\Gamma = \emptyset$.

1. – (associate judgements) Lemma 6.0.2 on nat-introduction rules yields that $0 \in N$ is a computable judgement, and hence $El_N(a, l, 0) \in L(0)$ is computable by the previous lemma on nat-elimination rules. Moreover, the other associate is J_2 which is assumed to be computable.
   – (parts) By the computability rule, $El_N(a, l, 0)$ and a evaluate into the same canonical element, so all cases are trivial.

2. – (associate judgements) J_2 is computable by assumption, thus, lemma 6.0.2 on nat-introduction rules yields that $\mathsf{s}(n) \in N$ is a computable judgement, and hence $El_N(a, l, \mathsf{s}(n)) \in L(\mathsf{s}(n))$ is computable by the previous lemma on nat-elimination rules. Moreover, since J_2, J_3 and J_5 are computable, we obtain $J_5[x := n, y := l(n, El_N(a, l, n))]$, i.e. the second associate judgement $l(n, El_N(a, l, n)) \in L(\mathsf{s}(n))$, is computable.
   – (parts) By the computability rule, $El_N(a, l, \mathsf{s}(n))$ and $l(n, El_N(a, l, n))$ evaluate into the same canonical element, so all cases are trivial.

- Subcase $\Gamma \neq \emptyset$.

1. – (associate) Lemma 6.0.2 on nat-introduction rules yields that $0 \in N\,[\Gamma]$ is a computable judgement, and hence $El_N(a, l, 0) \in L(0)\,[\Gamma]$ is computable by the previous lemma on nat-elimination rules.
   – (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z)\ \mathit{type}\ [\Gamma, z \in N][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x, y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)][x_1 := a_1, \ldots, x_n := a_n]$$

   are computable by head substitution lemma 5.3.4;

$$a \in L(0)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   is computable and

$$El_N(a, l, 0) = a \in L(0)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

   has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
   – (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a

c.c.s. fitting with Γ, and so

$$El_N(a, l, 0) = a \in L(0)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by (substitution $:=$) subcase.
Moreover,

$$a \in L(0)\ [\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable and then the result follows by transitivity lemma 5.3.13.

2. – (associate)J_2 is computable by assumption, thus, lemma 6.0.2 on nat-introduction rules yields that $\mathsf{s}(n) \in N[\Gamma]$ is a computable judgement, and hence $El_N(a, l, \mathsf{s}(n)) \in L(\mathsf{s}(n))\ [\Gamma]$ is computable by the previous lemma on nat-elimination rules.
– (substitution $:=$) Consider any c.c.s. $a_1 \in A_1, \ldots, a_n \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then

$$L(z)\ \mathit{type}\ [\Gamma, z \in N][x_1 := a_1, \ldots, x_n := a_n]$$

$$l(x, y) \in L(\mathsf{s}(x))\ [\Gamma, x \in N, y \in L(x)][x_1 := a_1, \ldots, x_n := a_n]$$

are computable by head substitution lemma 5.3.4;

$$n \in N\,[\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

and

$$a \in L(0)\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

are computable;

$$El_N(a, l, \mathsf{s}(n)) = l(n, El_N(a, l, n)) \in L(\mathsf{s}(n))\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

has empty context, then it is computable by $(\Gamma = \emptyset)$ subcase.
– (substitution $\leftarrow$) Consider any c.c.s. $a_1 = a_1' \in A_1, \ldots, a_n = a_n' \in A_n$ fitting with $\Gamma \equiv [x_1 \in A_1, \ldots, x_n \in A_n]$, then also $a_1 \in A_1, \ldots, a_n \in A_n$ is a c.c.s. fitting with Γ, and so

$$El_N(a, l, \mathsf{s}(n)) = l(n, El_N(a, l, n)) \in L(\mathsf{s}(n))\ [\Gamma][x_1 := a_1, \ldots, x_n := a_n]$$

is computable by (substitution $:=$) subcase.
Moreover, since$J_4[x := n, y := l(n, El_N(a, l, n))]$ is computable, also$J_4[x := n, y := l(n, El_N(a, l, n))][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$ which is

$$l(n, El_N(a, l, n)) \in L(\mathsf{s}(n))[\Gamma][x_1 \leftarrow a_1 = a_1', \ldots, x_n \leftarrow a_n = a_n']$$

is computable and then the result follows by transitivity lemma 5.3.13.

□

7

Formalisation in Coq

The work has been partially formalised in the Coq proof assistant [11] on top of the Lumsdaine-Bauer-Haselwarter General Type Theories framework [12]. The General Type Theories framework is, in turn, built on top of the Homotopy Type Theory library [13] with an eye towards future formalisation of the categorical semantics of type theories, but it does not depend on concepts such as the Univalence axiom or the Uniqueness of identity proofs. The only axiom that it is used is the function extensionality axiom and the development is constructive, since there are no uses of excluded middle or the axiom of choice.

The formalisation deals with selected topics of the thesis, starting from defining the signature and the types of $\mathcal{T}_{reg}$ together with the logical and the computation rules; then, the notion of computable judgement is introduced and, finally, it's showed the preservation of the computability of the logical rules in the case of empty context.

The formalization is publicly available on [14], where the General Type Theories library has been fixed to work with the latest version of the Coq-HoTT [15] library.

References

[1] P. Martin-Löf, "Intuitionistic type theory, notes by g. sambin of a series of lectures given in padua, june 1980." Bibliopolis, Naples, 1984.

[2] M. E. Maietti, "Modular correspondence between dependent type theories and categories including pretopoi and topoi." 2005, under consideration for publication in Math. Struct. in Comp. Science.

[3] S. L. M. Valentini, "Meta-mathematical aspects of martin-l¨of's type theory," Ph.D. dissertation, Katholieke Universiteit Nijmegen, 2000.

[4] B. Jacobs, *Introduction to Coalgebra. Towards Mathematics of States and Observations.* Cambridge University Press, 2016.

[5] ——, *Categorical Logic and Type Theory.*, ser. Studies in Logic. Elsevier, 1999, vol. 141.

[6] B. Nordström, K. Peterson, and J. Smith, *Programming in Martin L¨of 's Type Theory*. Clarendon Press, Oxford., 1990.

[7] N. G. de Bruijn, "Telescopic mappings in typed lambda calculus," *Information and Computation*, vol. 91, pp. 189–204, apr 1991.

[8] A. Bossi and S. Valentini, "An intuitionistic theory of types with assumptions of high-arity variables," *Annals of Pure and Applied Logic*, vol. 57, pp. 93–149, 1992.

[9] P. Martin-Löf, *Hauptsatz for the intuitionistic theory of iterated inductive definitions*, ser. Studies in Logic and the Foundations of Mathematics. J.E. Fenstad, 1971, vol. Proceedings of the Second Scandinavian Logic Symposium.

[10] W. Tait, "Intensional interpretation of functionals of finite type i," *Journal of Symbolic Logic*, vol. 32, no. 2, pp. 198–212, jun 1967.

[11] C. development team. (2023) Coq. [Online]. Available: http://coq.inria.fr/

[12] P. L. Lumsdaine, A. Bauer, and P. G. Haselwarter. (2020) A formalisation of general type theories in coq. [Online]. Available: https://github.com/peterlefanulumsdaine/general-type-theories/tree/arXiv

[13] P. L. Andrej Bauer, Jason Gross *et al.*, "The hott library: a formalization of homotopy type theory in coq." *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pp. 164–172, jan 2017.

[14] R. Borsetto. (2023) A canonical normal form for the type theory of regular categories. [Online]. Available: https://github.com/eapiova/canonical-normal-form-regular-categories

[15] H. T. T. development team. (2023) Coq-hott. [Online]. Available: https://github.com/HoTT/Coq-HoTT