Termination checking in the $\lambda\Pi$-calculus modulo theory. From a practical and a theoretical viewpoint

Guillaume Genestier

Termination checking in the λΠ-calculus modulo theory

From a practical and a theoretical viewpoint

Guillaume Genestier

From April 24h to August 4th 2017

Abstract

Dedukti is a type-checker for the $\lambda\Pi$-calculus modulo theory, which has the particularity to allow the user to declare rewrite rules, especially in order to encode the logic he/she wants to use. Thanks to the *Curry-Howard-De Bruijn correspondence*, the type checking done by *Dedukti* can be used to check proofs.

Unfortunately, to decide the type-checking, *Dedukti* needs the system of rewrite rules declared by the user, together with β-reduction, to be confluent and terminating. An external tool already exists to check the confluence, but no automatic tool exists for the termination, which must be checked independently by hand, before using *Dedukti*.

The subject of this report (and more widely of the internship) is to develop such a tool.

A first part of it is dedicated to an algorithm of termination checking: the *Size-Change Principle*. After a brief introduction to *Dedukti*, the algorithm is explained and the results obtained by an implementation of it for *Dedukti* are presented.

The second and longest part of this report is much more theoretic, since it presents reducibility candidates for the $\lambda\Pi$-calculus modulo theory. Those reducibility candidates can then be used together with the *Size-Change Principle* detailed in the first part, to check the termination of a rewrite rule system, as explained in [Wah07].

Contents

Chapter 1

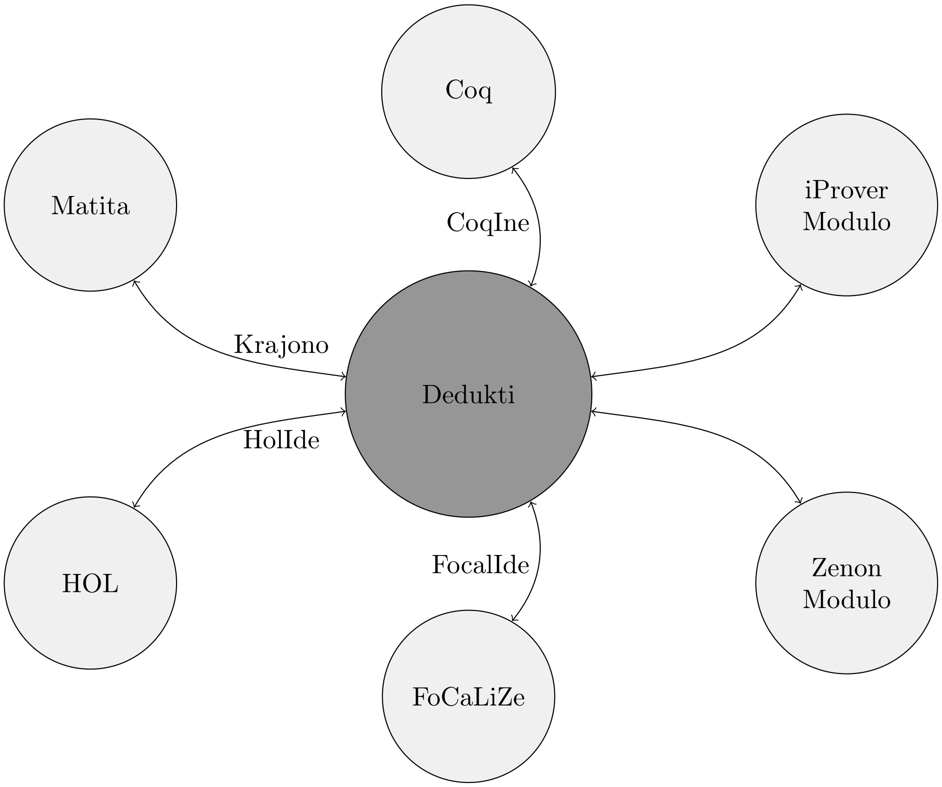
Implementing the Size-Change Principle for *Deduki*

1.1 What is *Dedukti*

Dedukti is a multipurpose type-checker based on the $\lambda\Pi$-calculus modulo theory [CD07].

The project *Dedukti* was initiated by Gilles Dowek and is developed by the *Inria*-team *Deducteam*. A first version was developed by Matthieu Boespflug [Boe11] and Quentin Carbonneaux [BCH12] in *Haskell* and then in *C*. The current version of *Dedukti* was developed mainly by Ronan Saillard [Sai15] and is programmed in *OCaml*.

The $\lambda\Pi$-calculus modulo theory is a calculus including only dependent types and rewrite rules. One can find the inference rules of this calculus, in this report section 2.2 or in the paper describing *Dedukti* [ABC$^+$16]. The rewrite rules allow the user to encode many logics in *Dedukti*, making it multipurpose and peculiarly well-suited for interoperability between proof systems. This is the reason why a lot of programs have been developed to translate proofs from other proof assistants in *Dedukti* and the way back. The graph below illustrate the interaction which exist between *Dedukti* and a range of proof assistants or automatic provers.



Among the rules of $\lambda\Pi$-calculus modulo theory, there is one conversion rule:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A : s \qquad \Gamma \vdash B : s}{\Gamma \vdash t : B} \; A \leftrightsquigarrow^{*} B$$

Without going into the detail of the meaning of the rule, which can be found in section 2.2, this rule means that if we can rewrite the type A into the type B then every term of one type is of the other type too.

To decide type-checking, *Dedukti* must check if two types are joinable by rewriting. To do so, in *Dedukti*, the two types are reduced to their normal forms and then the equality of normal forms is checked syntactically. But this requires that the normal forms exist and will be reached, situation which is ensured if the rewrite rules declared by the user, together with β-reduction, are *confluent* and *terminating*.

In the current version of *Dedukti*, a tool exists to check confluence, but checking the termination stays at the charge of the user and out of the scope of *Dedukti*. Meaning that, when a proof is checked by *Dedukti*, the checking is under the condition the rules declared by the user are terminating.

There exists ad-hoc proofs that some rewrite rule systems are terminating, for instance [Dow17], but no tools actually check it automatically.

The first part of this report presents the implementation of a simple but quite efficient termination checking algorithm for *Dedukti*.

1.2 The Size-Change Principle

The *Size-Change Principle* is a criterion introduced by Lee, Jones and Ben-Amram in [LJBA01] for a first-order functional language.

The principle is to analyse how the arguments evolve between each call of a function. To perform this analysis, we must analyse the arguments of every application on the right-hand side of a rewrite rule.

1.2.1 An example to begin with

Let's start the explanation on an example to get intuition and formalize it right after. For instance the Peano-like definition of the multiplication :

```
[n,m] mult (S m) n --> plus n (mult m n).
```

There are two function applications on the right-hand side of this example :

- `plus` applied to `n` which is exactly the second argument of the left-hand side and `(mult m n)` which is bigger than both arguments of the left-hand side.
- `mult` applied to `m` which is a strict subterm of `(S m)` and `n` which is exactly the second argument of the left-hand side.

We will sum up all of this information on two matrices:

$$\begin{array}{r|cc} {}^{\texttt{plus}}_{\texttt{mult}} & \texttt{n} & (\texttt{m} \times \texttt{n}) \\ \hline (\texttt{S m}) & \infty & \infty \\ \texttt{n} & = & \infty \end{array} \qquad \begin{array}{r|cc} {}^{\texttt{mult}}_{\texttt{mult}} & \texttt{m} & \texttt{n} \\ \hline (\texttt{S m}) & < & \infty \\ \texttt{n} & \infty & = \end{array}$$

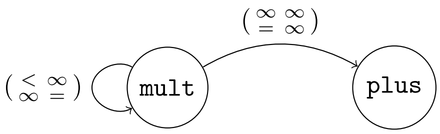
Where

- a $<$ symbol means that the argument of the called function corresponding to the column is a strict subterm of the argument of the caller function corresponding to the row;
- a $=$ symbol means that the argument of the called function and the argument of the caller function are he same;
- a ∞ symbol means that the arguments have no direct link.

In the original article [LJBA01], the variation of the size of the arguments where encoded in bipartite graphs. The idea of summing up it in matrices, which are easier to manipulate with a programming language like *OCaml* comes from Rodolphe Lepigre and Christophe Raffalli in [LR16].

From the calls that are encoded in the rewrite rules, we can construct a call-graph. The arrows are annotated with the matrices that sum up the size-changes.

Here is the call-graph for our example :


$\begin{pmatrix} \infty & \infty \\ = & \infty \end{pmatrix}$
$\begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix}$
mult
plus

In this case we know that the function `mult` is terminating, because, there is only one loop in the call-graph, which is indexed by $\begin{pmatrix} < & \infty \\ \infty & = \end{pmatrix}$ and there is a $<$ on the diagonal of the matrix, meaning that between two recursive calls of the function, at least one argument (here the first one) has strictly decreased and the subterm order is well-founded.

Definition 1.2.1 (Well-founded relation)**.** *We remind the reader that a relation $\mathcal{R}$ is* well-founded *on a class X if for all sets S in X, if S is non-empty, S has a minimal element for $\mathcal{R}$. It can be rewrite :*

$$\forall S \subseteq X, (S \neq \emptyset \Rightarrow \exists m \in S, \forall s \in S, (s, m) \notin \mathcal{R})$$

An other characterization, often more intuitive, is that a relation $\mathcal{R}$ is well-founded *if there is no countable infinite descending chain.*

1.2.2 The tropical semiring and the transitive closure of call-graphs

Of course, the example we have chosen to present the *Size-Change Principle* is peculiarly easy, but in fact the criterion is quite efficient, especially to deal with mutual recursion and permuted arguments.

In those cases, the call-graph does not contain enough arrows to say if the rewrite rules system is terminating or not. Then, we have to compute the transitive closure of the call-graph. To do so, we have to multiply matrices containing $<$, $=$ and ∞, meaning that we must define a multiplication and an addition on such symbols.

In fact, those operations are the one of the *min-plus semiring* (often called the *tropical semiring*), with the values :

symbol	value
$<$	-1
$=$	0
∞	$+\infty$

We then have the tables :

$+$	$<$	$=$	∞
$<$	$<$	$<$	$<$
$=$	$<$	$=$	$=$
∞	$<$	$=$	∞

$\times$	$<$	$=$	∞
$<$	$<$	$<$	∞
$=$	$<$	$=$	∞
∞	∞	∞	∞

Those definitions of the operations are quite intuitive. When a function f_1 calls a function f_2 which calls a function f_3, we can link the arguments of f_3 to the ones of f_1 going through the ones of f_2.

Let's choose an argument of f_1 and one of f_3 that we would like to link. To do so, we can use any of the argument of f_2 as a middle step, so it is possible to go through the best one, that justifies that the addition operation is *min*.

Once the best argument of f_2 is chosen, there is two links to multiply, one between f_1 and f_2 and the other on between f_2 and f_3.

- If a link is labelled ∞ it means that we have no information about one part of the transformation, and we cannot recover it no matter what is the label of the other part.
- On the other hand, if a link is labelled $<$ and the other one $<$ or $=$, we have a large and a strict decrease, so globally, we have a strict decrease.
- If both are labelled $=$, the transitivity of the equality ensures us that the arguments of f_1 and f_3 are the same.

1.2.3 A definition of the *Size-Change Principle*

We now have all the ingredients, to give a definition of the *Size-Change Principle*

Definition 1.2.2 (Fully applied terms)**.** *We consider a language which contains:*

- *variables: x, y;*
- *constants: c, d;*
- *functions: f, g, each having an associated arity.*

The set of fully applied terms is:

$$t ::= x \,|\, c \,|\, f\, t_1 \dots t_n, \text{ where } n \text{ is the arity of } f$$

Definition 1.2.3 (Size-Change Principle)**.** *We assume given a well-founded order on fully applied terms, which is used to construct the call-graph. The subterm order has this property and is the one chosen for the implementation.*

A set of rewrite rules is Size-Change Terminating *if:*

- *All rules are of the form*
$$f\, t_1 \dots t_n \rightsquigarrow \tau$$
where n is the arity of f and $t_1, \dots, t_n, \tau$ are fully applied terms.
- *The transitive closure of the call graph is such that all arrows linking a node with itself are labelled with a matrix having a $<$ on the diagonal.*

1.3 Implementation issues

1.3.1 Adapting an already existing implementation

When they developped *PML*, a program which is absolutely independent of *Dedukti*, Rodolphe Lepigre and Christophe Raffalli implemented the algorithm of the *Size-Change Principle* and they accepted to share their implementation with us.

This implementation provided us with the computation of the transitive closure of a call graph and the checking that every looping arrows are labelled with a matrix having a $<$ on the diagonal.

To do so efficiently, Rodolphe Lepigre and Christophe Raffalli introduced a notion of matrix *subsuming* another one.

Definition 1.3.1 (Subsuming). *A matrix* $A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix}$ subsumes *the matrix* $B = \begin{pmatrix} b_{1,1} & \dots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,n} \end{pmatrix}$ *if the two matrices have the same dimension and for every* i *and* j, $a_{i,j}$ *is smaller than* $b_{i,j}$. *The* subsuming *relation is the pointwise extension of* $<$

Property 1.3.2. *If a matrix* A *subsumes a matrix* B*, then for all matrix* X*, when the dimensions are compatible, then* AX *subsumes* BX *and* XA *subsumes* XB.

Proof. It is direct, because the sum and the product are increasing functions in the tropical semiring. □

If two arrows link the same nodes and one is labelled with a matrix which subsumes the label of the other, then it is not useful to use the smallest matrix to compute the transitive closure. Indeed, when A subsumes B, every $<$ symbol on the diagonal of A is also on the diagonal of B and, since the subsuming relation is stable by multiplication, if the Size-Change Principle is verified removing B from the call-graph, it is also verified with both A and B, since the new labels on the looping arrows contain at least as much $<$ on the diagonal.

1.3.2 The implementation and its results

Since the program developed for *PML* fitted perfectly our purposes for the call-graph computation, the aim was to develop a program interfacing *Dedukti* with it. Especially, it was necessary to identify the "calls" and to associate a matrix to them.

This goal led to the writing of 200 lines of *OCaml* which were added to the 300 from Rodolphe Lepigre and Christophe Raffalli's program, making a complete working first prototype of termination checker for *Dedukti*

Unfortunately, the *Size-Change Principle*, in its original version, was designed for a much poorer language than the λΠ-calculus modulo theory, including only functions applied to *constructor patterns*. Hence, most of the *Dedukti* files are out of the scope of this criterion, mostly because of the application of an argument.

For instance, *Dedukti* allows the user to declare :

```
List : Type.
Nil : List.
Cons : Nat -> List -> List.
def map : (Nat -> Nat) -> List -> List.
[] map _ Nil --> Nil.
[f, x, l] map f (Cons x l) --> Cons (f x) (map f l).
```

and

```
Lamt : Type.
def App : Lamt -> Lamt -> Lamt.
Lam : ( Lamt -> Lamt ) -> Lamt.
[f, x] App (Lam f) x --> f x.
```

In the first example, the function `map` is terminating, whereas the second example is an encoding of the untyped λ-calculus, which is well-known for being non-terminating.

But those two examples are quite close in the point of view of the *Size-Change Principle*: we encounter a call to `f x` where `f` is a variable. A variable does not correspond with any nodes in the building call-graph. Then no called function can be identified and the program must raise an error since we have seen this situation can both occur in terminating and in non-terminating situations.

The implementation is working, but the criterion, in the form which has been implemented is far from being sufficient, since it accepts only some basics functions (like addition, multiplication, exponentiation, Ackerman function, list concatenation or list sorting...) and the encoding of the first-order logic restricted to implication, universal quantification and negation, sometimes called intuitionistic minimal logic.

This main problem led us to a more theoretic part of the internship, in which we had to enrich the criterion in order to make it more useful for our purpose of developing a termination checker for *Dedukti*. A few modifications of the criterion exists in the literature. We can for instance mention Pierre Hyvernat's idea [Hyv13] to enrich the tropical semiring in order to cover the case where the size of an argument increases a little before decreasing more. Or Neil Jones and Nina Bohr [JB08] who modified the criterion to check termination of untyped λ-calculus terms.

Those enrichment were quite interesting, but it did not solve our main problem with the variable application, so we choose to investigate the work of David Wahlstedt [Wah07], who uses the *Size-Change Principle* in an original way and manage to study the termination of a system which is quite close of the λΠ-calculus modulo theory.

Chapter 2

Adaptation of the reducibility proof of Wahlstedt to the $\lambda\Pi$-calculus modulo theory

In his thesis [Wah07], David Wahlstedt proposes a calculus with Martin-Löf types and rewriting. He gives a criterion for the weak reducibility of a system of rewrite rules, coupled to the β-reduction. This work interested us, because it uses the *Size-Change Principle* in a rich system and it is very modular, allowing us to change only one part of the thesis to get the benefit of the entire one.

The structure of Wahlstedt's thesis is the following :

- Define a reducibility predicate for weak normalisation.
- Show that if every function in the signature is reducible then every typable term is reducible.
- Show that if the call relation is well-founded and every type in the signature is reducible, then every function in the signature is reducible.
- The size-change principle is used to show that the call relation is well-founded.
- The reducibility of every type occurring in the signature is decidable.

Our aim in the second part of the internship was to adapt the tree first points to the $\lambda\Pi$-calculus modulo theory. Doing it, we can re-use the demonstration of the fourth point given in the original thesis and use the *Size-Change Principle* in a more efficient way.

2.1 Syntax

To begin, we will give the syntax of the $\lambda\Pi$-calculus modulo theory.

2.1.1 Terms and contexts

We use x and y to denote *variables*.

Definition 2.1.1 (Terms).

$$t, \tau, u, v, l, r ::= x \,|\, \lambda(x : u).t \,|\, t\, u \,|\, c \,|\, f$$

We denote by Λ *the set of all terms.*

Definition 2.1.2 (Types).

$$T, U ::= \lambda(x : U).T \,|\, \Pi(x : U).T \,|\, U\, v \,|\, d \,|\, F$$

Definition 2.1.3 (Kinds).

$$K ::= \text{Type} \,|\, \Pi(x : U).K$$

Remark. For now, d, c and f are only symbols. Their meaning will appear later in this section.

Definition 2.1.4 (Contexts).

$$\Gamma, \Delta ::= [] \,|\, \Gamma, x : t$$

Notations. For readability, we write only the first Π when they are chained.

$$\Pi(x_1 : T_1)\,(x_2 : T_2)\,(x_3 : T_3)\,U \text{ means } \Pi(x_1 : T_1)\;(\Pi(x_2 : T_2)\;(\Pi(x_3 : T_3)\,U))$$

We draw a line over the left part of a Π or a λ, to mean that the declaration is iterated. The variable on which the iteration is done and the bounds are inferred by the reader from the context.

$$\Pi\overline{(x_i : T_i)}\,U \text{ means } \Pi(x_1 : T_1)\ldots(x_n : T_n)\,U$$

Definition 2.1.5 (Closed context). *A context* $\Gamma = (x_1 : t_1, \ldots, x_i : t_i, \ldots)$ *is closed if for all* i*, all free variables appearing in* t_i *are in* $\{x_1, \ldots, x_{i-1}\}$*.*

2.1.2 Substitutions

To express transformation that occur on terms during rewriting, it is necessary to define an operation of substitution, expressing the fact that every free occurrences of a particular variable takes the same value.

Definition 2.1.6 (Substitution). *More generally, a substitution* δ *a function which links every variable to a term. We define a function* $\tilde{\delta}$ *from terms to terms by applying simultaneously* δ *on every free variable. If one bound variable has the same name as a free variable occurring in the image of* δ*, then the bound variable must be renamed, in order to avoid* variable capture*.*

Remark. It is important to note that the linked variables are not affected by substitution. That prevents us from "capturing" variables that should not be.

Notations (Substitution). The notation to express it must mention the term in which the substitution occurs, the variable which is replaced and the term which replaces this variable, so it is quite natural to choose :

- $\tau\,[{}^{t}/_{x}]$ to denote the term τ in which x is replaced by t.
- δ, σ and ϑ are used to denote a general substitution.
- We will denote by $t\,\delta$ the substitution $\tilde{\delta}(t)$ and by $\delta(x)$ the image of the variable x by the function δ.

2.1.3 Signature

Definition 2.1.7 (Signature)**.** *A signature is a 6-tuple* $(\mathbb{D}, \mathcal{D}, \mathbb{C}, \mathcal{C}, \mathbb{F}, \mathcal{F})$*.*

- $\mathbb{D}$ *is a set of* set constructors*,*
- $\mathcal{D}$ *is a function which associates to each* $d \in \mathbb{D}$ *a closed term of the form* $\mathcal{D}(d) = \Pi(x_1 : T_1) \dots (x_k : T_k)\ \mathrm{Type}$*. Under this specification,* k *is called* the arity of d*.*
- $\mathbb{C}$ *is a set of* element constructors*,*
- $\mathcal{C}$ *is a function which associates to each* $c \in \mathbb{C}$ *a closed term of the form* $\mathcal{C}(c) = \Pi(x_1 : U_1) \dots (x_m : U_m)\,(d\,\tau_1 \dots \tau_k)$*, where for all* i*,* $\mathrm{FV}(\tau_i) \subseteq \{x_1, \dots, x_m\}$ *and* k *is the arity of* d*. Under this specification,* m *is called* the arity of c*.*
- $\mathbb{F}$ *is a set of* defined functions*,*
- $\mathcal{F}$ *is a function which associates to each* $f \in \mathbb{F}$ *a closed term of the form* $\mathcal{F}(f) = \Pi(x_1 : T_1) \dots (x_n : T_n)\,U$ *where* U *is not a* Π*. Under this specification,* n *is called* the arity of f*.*

Notations. From now, d, c and f will always represent set constructor, element constructor and defined function respectively.

We denote $\mathrm{ar}(h)$ the arity of the symbol h.

Remark. The arity of any symbol defined in the signature can be 0.

Definition 2.1.8 (Elementary types)**.** *We name* elementary types *a fully applied set constructor.*

Definition 2.1.9 (Precedence on set constructors)**.** *We define the relation on* $\mathbb{D}$*:* $\preccurlyeq_{\mathcal{D}}$ *as the reflexive-transitive cloture of* $d' \preccurlyeq_{\mathcal{D}} d$ *if:*

- $\mathcal{D}(d) = \Pi\overline{(x_i : T_i)}.\,\mathrm{Type}$ *and* d' *appears in a* T_i
- *or if there is a* c *such that* $\mathcal{C}(c) = \Pi\overline{(x_i : U_i)}.(d\,\bar{s})$ *and* d' *appears in a* U_i*.*

Definition 2.1.10 ($\mathcal{D}$-equivalence)**.** *If $d \preccurlyeq_{\mathcal{D}} d'$ and $d' \preccurlyeq_{\mathcal{D}} d$, d and d' are said* $\mathcal{D}$-equivalent*, what is denoted $d \approx_{\mathcal{D}} d'$.*

Remark. $\preccurlyeq_{\mathcal{D}}$ is by definition a pre-order, so $\approx_{\mathcal{D}}$ is an equivalence relation.
Since we only consider finite signature, $\prec_{\mathcal{D}}$ is well-founded.

Definition 2.1.11 (Strictly positive set constructor)**.** *A set constructor d is said* strictly positive *if:*

- $\mathcal{D}(d) = \Pi\overline{(x_i : T_i)}.\mathrm{Type}$ *and no symbole $\mathcal{D}$-equivalent to d occurs in any T_i*
- *For all c such that $\mathcal{C}(c) = \Pi\overline{(x_i : U_i)}.(d\,\bar{s})$, if the U_i are of the form $\Pi\overline{(y_j : V_j)}.V$ then no symbol $\mathcal{D}$-equivalent to d appears in a V_j.*

We restrict our study to the signatures where every set constructor is stictly pôsitive.

Definition 2.1.12 (Constructor pattern)**.** *We define this syntactical sub-category of terms as :*

$$p ::= x \mid c\, p_1 \ldots p_n$$

Definition 2.1.13 (β-normal term)**.** *We define this syntactical sub-category of terms as :*

$$s ::= x\, s_1 \ldots s_n \mid h\, s_1 \ldots s_{\mathrm{ar}(h)} \mid \lambda(x : T).s$$

where h is one symbol in the signature, set constructor, element constructor or defined function.

Now, we have all the necessary definitions to enrich the signature with rewrite rules.

Definition 2.1.14 (Rewrite rules in the signature)**.** *The signature is enriched with a set $\mathbb{R}$ of* rewrite rules*. Each* rewrite rule *is of the form $f\, p_1 \ldots p_k \to s$ where :*

- *the p_i are constructor patterns,*
- $k \leqslant \mathrm{ar}(f)$*,*
- *p_k is not a variable,*
- *s is β-normal,*
- *s starts with* $(\mathrm{ar}(f) - k)$ *λ-abstractions,*
- *the rule is* left-linear*, meaning that a free variable can't appear twice in $f\, p_1 \ldots p_k$.*

Furthermore, we demand $\mathbb{R}$ *to be* non-overlapping*, meaning that for any two distinct rules* $l_1 \to r_1$ *and* $l_2 \to r_2$ *in* $\mathbb{R}$*, there are no substitutions* δ_1 *and* δ_2 *such that* $l_1\,\delta_1 = l_2\,\delta_2$[1]*.*

Remark. We do not have defined typable terms yet, but once it will be done, we will restrict our study to the rewrite rules that preserve typing.

2.2 Rules of inference

Now that we have presented the syntax of terms and contexts, we can introduce the rules of inference of the $\lambda\Pi$-calculus modulo theory.

Context formation

$$\frac{}{[]\ \text{well-formed}} \qquad \frac{\Gamma \vdash A : \text{Kind}}{\Gamma, x : A\ \text{well-formed}} \qquad \frac{\Gamma \vdash A : \text{Type}}{\Gamma, x : A\ \text{well-formed}}$$

Axioms

$$\frac{\Gamma\ \text{well-formed}}{\Gamma \vdash \text{Type} : \text{Kind}} \qquad \frac{\Gamma\ \text{well-formed}}{\Gamma \vdash x : A}\ x : A \in \Gamma$$

Product

$$\frac{\Gamma \vdash A : \text{Type} \qquad \Gamma, x : A \vdash B : \text{Kind}}{\Gamma \vdash \Pi(x : A)\,B : \text{Kind}} \qquad \frac{\Gamma \vdash A : \text{Type} \qquad \Gamma, x : A \vdash B : \text{Type}}{\Gamma \vdash \Pi(x : A)\,B : \text{Type}}$$

Lambda-abstraction

$$\frac{\Gamma \vdash A : \text{Type} \qquad \Gamma, x : A \vdash B : \text{Kind} \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : \Pi(x : A)\,B}$$

$$\frac{\Gamma \vdash A : \text{Type} \qquad \Gamma, x : A \vdash B : \text{Type} \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : \Pi(x : A)\,B}$$

Application

$$\frac{\Gamma \vdash t : \Pi(x : A)\,B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B\,[{}^{u}/_{x}]}$$

[1] We have chosen here not to enforce variables to be distinct in each rules, since in practice the files considered in *Dedukti* do not follow this restriction. It would have been possible to consider only one substitution and to rename variables, which is equivalent to the presentation with two substitutions.

Signature symbols

$$\frac{\Gamma, x_1 : T_1, \dots, x_i : T_i \vdash T_{i+1} : \text{Type}}{\Gamma \vdash d : \mathcal{D}(d)}\ \mathcal{D}(d) = \Pi\overline{(x_i : T_i)}\ \text{Type}$$

$$\frac{\Gamma, x_1 : U_1, \dots, x_i : U_i \vdash U_{i+1} : \text{Type} \quad \Gamma, \overline{x_i : U_i} \vdash \tau_j : T_j}{\Gamma \vdash c : \mathcal{C}(c)}\ \begin{cases} \mathcal{C}(c) = \Pi\overline{(x_i : U_i)}\,(d\,\tau_1 \dots \tau_k) \\ \mathcal{D}(d) = \Pi\overline{(x_i : T_i)}\ \text{Type} \end{cases}$$

$$\frac{\Gamma, x_1 : T_1, \dots, x_i : T_i \vdash T_{i+1} : \text{Type} \quad \Gamma, \overline{x_i : T_i} \vdash U : \text{Type}}{\Gamma \vdash f : \mathcal{F}(f)}\ \mathcal{F}(f) = \Pi\overline{(x_i : T_i)}\,U$$

$$\frac{\Gamma, x_1 : T_1, \dots, x_i : T_i \vdash T_{i+1} : \text{Type}}{\Gamma \vdash f : \mathcal{F}(f)}\ \mathcal{F}(f) = \Pi\overline{(x_i : T_i)}\ \text{Type}$$

Conversion

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A : \text{Type} \qquad \Gamma \vdash B : \text{Type}}{\Gamma \vdash t : B}\ A \leftrightsquigarrow^* B$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A : \text{Kind} \qquad \Gamma \vdash B : \text{Kind}}{\Gamma \vdash t : B}\ A \leftrightsquigarrow^* B$$

2.2.1 Rewriting

Definition 2.2.1 (Well-typed rules). *Now that we have given the typing rules of the $\lambda\Pi$-calculus, we can restrict our study to signatures which are* well-typed, *meaning that:*

- *all the rewrite rules in $\mathbb{R}$ preserve typing. For all Γ, t, T, u, if $\Gamma \vdash t : T$ and $t \to u \in \mathbb{R}$, then $\Gamma \vdash u : T$.*
- *all the left-hand sides of rules are typable. For all $f\,\bar{p}_i \to u \in \mathbb{R}$ with $\mathcal{F}(f) = \Pi(\overline{x_i : T_i}).U$ there is a Δ_0 such that for all $i \leqslant m$,*

$$\Delta_0 \vdash p_i : T_i \left[{}^{p_1}/_{x_1} \dots {}^{p_{i-1}}/_{x_{i-1}}\right]$$

When we add to the rewrite rules declared in $\mathbb{R}$ the β-reduction $(\lambda x.t)\,u \to u\,[{}^{t}/_{x}]$, we can define the rewriting relation $\rightsquigarrow$.

Definition 2.2.2 (Rewriting relation). *We define inductively the relation, by :*

- *if t β-reduces to u then $t \rightsquigarrow u$,*
- *if there is $l \to r \in \mathbb{R}$ and δ such that $l\,\delta = t$ and $r\,\delta = u$ then $t \rightsquigarrow u$,*
- *if $t \rightsquigarrow u$ then for all v, $t\,v \rightsquigarrow u\,v$, $v\,t \rightsquigarrow v\,u$,*
- *if $t \rightsquigarrow u$ then $\lambda x.t \rightsquigarrow \lambda x.u$ and $\Pi x.t \rightsquigarrow \Pi x.u$.*

Notations. We denote by $\rightsquigarrow^*$ the reflexive-transitive closure of $\rightsquigarrow$ and by $\leftrightsquigarrow^*$ the reflexive-symmetric-transitive closure of $\rightsquigarrow$

Definition 2.2.3 (Confluence)**.** *A system of rewrite rules is confluent if for any terms t, t_1 and t_2, if $t \rightsquigarrow^* t_1$ and $t \rightsquigarrow^* t_2$ then there is a u such that $t_1 \rightsquigarrow^* u$ and $t_2 \rightsquigarrow^* u$.*

Proposition 2.2.4. *With all the constraints on rewrite rules, which are given by definition 2.1.14 and the preservation of typing, we can ensure that $\mathbb{R}$ is confluent.*

We won't prove this proposition, which is a consequence of orthogonality [OR94], because our purpose is to study termination and not confluence. The interested reader can find the demonstration, which is quite long and does not introduce techniques that will be useful anywhere else in this report, in [Wah07, section 2.2.3 page 31]

2.3 Reducibility predicate

2.3.1 Weak normalisation

Definition 2.3.1 (Normalisation predicates)**.**

$$\begin{aligned}
\mathrm{NF}(u) &\equiv \neg\,(\exists v.u \rightsquigarrow v)\\
\mathrm{SN}(u) &\equiv \neg\,(\exists (v_i)_{i\in\mathbb{N}}.v_0 = u \wedge \forall i.v_i \rightsquigarrow v_{i+1})\\
\mathrm{WN}(u) &\equiv \exists v.u \rightsquigarrow^* v \wedge \mathrm{NF}(v)\\
u \Downarrow v &\equiv u \rightsquigarrow^* v \wedge \mathrm{NF}(v)
\end{aligned}$$

Definition 2.3.2 (Strongly neutral terms)**.**

$$\begin{aligned}
b ::= &\; x\,t_1 \dots t_n \;\textit{where}\; \mathrm{NF}(t_i)\\
&|f\,t_1 \dots t_n \;\textit{where}\; \mathrm{NF}(f\,t_1 \dots t_n) \;\textit{and}\; n \geqslant \mathrm{ar}(f)
\end{aligned}$$

We denote by $\mathcal{N}$ the set of strongly neutral terms.

Lemma 2.3.3 (Weak normalisation of an application)**.** *If x is a variable and t a term, then* $\mathrm{WN}(t\,x) \Rightarrow \mathrm{WN}(t)$

Proof. Assume $\mathrm{WN}(t\,x)$. Then there is a reduction sequence

$$t\,x \rightsquigarrow t_1 \rightsquigarrow t_2 \rightsquigarrow \dots \rightsquigarrow t_n$$

where t_n is normal. We proceed by induction on the length n of the shortest reduction.

- If $n = 0$ then $t\,x$ is normal, then t is normal too.
- Else $t\,x$ contains a redex. Applying a term to a variable cannot be of the form $l\,\delta$ where l is the left-hand side of a rule and δ a substitution, so the first step $t\,x \rightsquigarrow t_1$ must be of the form :

- $(\lambda y.u)x \rightsquigarrow u\left[{}^{x}/_{y}\right]$ and since $u\left[{}^{x}/_{y}\right]$ is normalizable, u is normalizable too, hence $\mathrm{WN}(t)$.
- $t\,x \rightsquigarrow u\,x$, because $t \rightsquigarrow u$. Then $u\,x$ reduce to t_n so we have $\mathrm{WN}(u\,x)$ and there is a reduction in $n-1$ steps so by the induction hypothesis, we have $\mathrm{WN}(u)$. We can deduce $\mathrm{WN}(t)$ □

2.3.2 Reducible terms

Definition 2.3.4 (Reducibility of a term)**.** *We define mutually the predicates* RED *for* Kind*, the inhabitants of* Kind *and the inhabitants of those inhabitants.*

$\mathrm{RED}_{(\mathrm{Kind})}\,(t)$ *holds if one of the following conditions occurs:*

- $t = \mathrm{Type}$ *and then* $\mathrm{RED}_{(\mathrm{Type})}\,(U)$ *holds if one of the following conditions occurs:*

  - $$\exists d, u_1, \ldots, u_k.\begin{cases} U \Downarrow d\,u_1 \ldots u_k \\ \mathcal{D}(d) = \Pi(x_1 : T_1) \ldots (x_k : T_k)\ \mathrm{Type} \\ \forall i.\, \mathrm{RED}_{(\mathrm{Type})}\,(T_i) \wedge \mathrm{RED}_{(T_i)}\,(u_i) \end{cases} \quad \textit{and then}\ \mathrm{RED}_{(U)}\,(v)$$

    holds if one of the following conditions occurs:

    - $$\exists c, v_1, \ldots, v_m.\begin{cases} v \Downarrow c\,v_1 \ldots v_m \\ \mathcal{C}(c) = \Pi(x_1 : U_1) \ldots (x_m : U_m)\,(d\,\tau_1 \ldots \tau_k) \\ \forall i.\, \mathrm{RED}_{(\mathrm{Type})}\left(U_i\left[{}^{v_1}/_{x_1}, \ldots, {}^{v_{i-1}}/_{x_{i-1}}\right]\right) \\ \forall i.\, \mathrm{RED}_{\left(U_i\left[{}^{v_1}/_{x_1}, \ldots, {}^{v_{i-1}}/_{x_{i-1}}\right]\right)}\,(v_i) \end{cases}$$
    - $\exists b \in \mathcal{N}.v \Downarrow b$

  - $$\exists A, B.\begin{cases} U \Downarrow \Pi(x : A)\ B \\ \mathrm{RED}_{(\mathrm{Type})}\,(A) \\ \forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{(\mathrm{Type})}\,(B\,[{}^{a}/_{x}]) \end{cases} \quad \textit{and then}\ \mathrm{RED}_{(U)}\,(v)$$

    holds if $\forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{(B[{}^{a}/_{x}])}\,(v\,a)$

  - $\exists b \in \mathcal{N}.U \Downarrow b$ *and then* $\mathrm{RED}_{(U)}\,(v)$ *holds if* $\exists b' \in \mathcal{N}.v \Downarrow b'$

- $$\exists A, B.\begin{cases} t = \Pi(x : A)\ B \\ \mathrm{RED}_{(\mathrm{Type})}\,(A) \\ \forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{(\mathrm{Kind})}\,(B\,[{}^{a}/_{x}]) \end{cases} \quad \textit{and then}\ \mathrm{RED}_{(\Pi(x:A)\,B)}\,(u)$$

  holds if $\forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow \mathrm{RED}_{(B[{}^{a}/_{x}])}\,(u\,a)$

Are the predicates RED **well-defined ?**

We will ensure that the predicates RED are well-defined for elementary types by defining inflating sequences of sets and name RED their least fixpoints.

Definition 2.3.5 (Elementary type interpretations). *Let $\mathcal{A} = \{d_i\}_{i\in\{1,\dots,n\}}$ be a $\approx_{\mathcal{D}}$-equivalence class.*

$$\begin{array}{rccc} F_{\mathcal{A}} : & \prod_{i=1}^{n} \mathcal{P}(\Lambda) & \to & \prod_{i=1}^{n} \mathcal{P}(\Lambda) \\ & (X_i)_{i\in\{1,\dots,n\}} & \mapsto & \left(\{\, u \mid \exists b. u \Downarrow b\} \cup \left\{ u \left| \begin{array}{c} u \Downarrow c\, \bar{v} \\ \exists c, v_1, \cdots, v_m. \mathcal{C}(c) = \Pi\overline{(x_j : U_j)}.(d_i\, \bar{s}) \\ \forall j. v_j \in R_{U_j}(\bar{X}) \end{array} \right. \right\} \right)_{i\in\{1,\dots,n\}} \end{array}$$

where $R_\alpha(\bar{X}) = \begin{cases} \llbracket d \rrbracket & \textit{if} \quad \alpha = d\, \bar{s}, \textit{ where } d \textit{ is fully applied and } \forall u \in \mathcal{A}. d \prec_{\mathcal{D}} u \\ X_i & \textit{if} \quad \alpha = d_i\, \bar{s}, \textit{ where } d \textit{ is fully applied} \\ \mathcal{N} & \textit{if} \quad \alpha \in \mathcal{N} \\ \left\{ t \in \Lambda \,\middle|\, \forall u \in R_{T_1}(\bar{X}). t\, u \in R_{T_2[{}^{u}/_{x}]}(\bar{X}) \right\} & \textit{if } \alpha = \Pi(x : T_1).T_2 \end{cases}$

where, for all $\approx_{\mathcal{D}}$-equivalence class $\mathcal{A}'$ where $\mathcal{A}' \prec_{\mathcal{D}} \mathcal{A}$ and $d \in \mathcal{A}'$, $\llbracket d \rrbracket$ is the least fixpoint of $F_{\mathcal{A}'}$.

Remark. We must insist here on the fact that $\llbracket d \rrbracket$ must be seen as the interpretation of any elementary type of the form $d\, t_1 \dots t_n$ where d is fully applied.

Existence of the least fix point. We will show that $F_{\mathcal{A}}$ is increasing. Thus, from Knaster-Tarski theorem, the function has a least fixpoint. Let's take $\bar{X} \subseteq_{\text{prod}} \bar{Y} \subseteq \Lambda$ and $u \in F_{\mathcal{A}}(\bar{X})$. Two cases occur:

- $u \Downarrow b$, then $u \in F_{\mathcal{A}}(\bar{Y})$
- $\exists c. u \Downarrow c\, \bar{v}, \mathcal{C}(c) = \Pi\overline{(x_j : U_j)}.(d\, \bar{s})$ and $\forall j. v_j \in R_{U_j}(\bar{X})$. Then two subcases appear:
  - $U_j = \Pi\overline{(x_k : T_k)}.d'\, \bar{\tau}$ such that for all $d_i \in \mathcal{A}$ we have $d' \prec_{\mathcal{D}} d_i$. As every elementary type is strictly positive, all the symbols occuring in the T_k are stricly smaller for the order $\preccurlyeq_{\mathcal{D}}$ than those in $\mathcal{A}$, so for all k, $R_{T_k}(\bar{Y}) = R_{T_k}(\bar{X})$. For $\tau_k \in R_{T_k}(\bar{X})$ then $v_j\, \bar{\tau} \in \llbracket d' \rrbracket$ so $v_j \in R_{U_j}(\bar{Y})$,
  - $U_j = \Pi\overline{(x_k : T_k)}.d_i\, \bar{\tau}$. As every elementary type is strictly positive, all the symbols occuring in the T_k are stricly smaller for the order $\preccurlyeq_{\mathcal{D}}$ than those in $\mathcal{A}$, so $R_{T_1}(\bar{Y}) = R_{T_1}(\bar{X})$. For $\tau_k \in R_{T_k}(\bar{X})$ then $v_j\, \bar{\tau} \in X_i$, then $v_j\, \bar{\tau} \in Y_i$ because $\bar{X} \subseteq_{\text{prod}} \bar{Y}$ so $v_j \in R_{U_j}(\bar{Y})$.

  So for all j, we have $v_j \in R_{U_j}(\bar{Y})$, hence, $u = c\, \bar{v}$ is in $F_{\mathcal{A}}(\bar{Y})$

So $F_{\mathcal{A}}(\bar{X}) \subseteq F_{\mathcal{A}}(\bar{Y})$, hence $F_{\mathcal{A}}$ is increasing. □

Hence, since $\preccurlyeq_{\mathcal{D}}$ is well founded, we proceed inductively on the $\approx_{\mathcal{D}}$ equivalence classes to define the interpretation of every elementary types.

Definition 2.3.6 (Reducible terms). *We can define* $\mathrm{RED}_T(u)$*, where T reduces to an elementary type by*

$$\exists d, u_1, \dots, u_m. T \Downarrow d\, u_1 \dots u_m \textit{ and } u \in \llbracket d \rrbracket.$$

We can define $\mathrm{RED}_T(u)$*, where* T *reduces to a strongly normal type by* $u \in \mathcal{N}$*.*

Then we can define, by induction on the number of Π *in the normal form,* $\mathrm{RED}_T(u)$*, where* T *reduces to* $\Pi(x : A)U$ *by* $\{\, t \in \Lambda |\, \forall u \in \llbracket A \rrbracket . t\, u \in \llbracket U\, [{}^{u}/_{x}] \rrbracket \}$*.*

Definition 2.3.7 (Reducible types)**.** *Once reducible terms are well-defined, we can define reducible types by the increasing sequence:*

- $\mathbb{T}\mathrm{ype}^{(0)} = \{\, U |\, \exists b. U \Downarrow b\}$

- *For any ordinal* α*,*

$$\begin{aligned}\mathbb{T}\mathrm{ype}^{(\alpha+1)} &= \mathbb{T}\mathrm{ype}^{(\alpha)} \\ &\cup \left\{ U \,\middle|\, \begin{array}{c} U \Downarrow d\, u_1 \dots u_m \\ \exists d, u_1, \dots, u_m . \mathcal{D}(d) = \Pi(x_1 : T_1) \dots (x_k : T_k) \text{ Type} \\ \forall i. T_i \in \mathbb{T}\mathrm{ype}^{(\alpha)} \wedge \mathrm{RED}_{(T_i)}\,(u_i) \end{array} \right\} \\ &\cup \left\{ U \,\middle|\, \exists A, B. \begin{array}{c} U \Downarrow \Pi(x : A)\ B \\ A \in \mathbb{T}\mathrm{ype}^{(\alpha)} \\ \forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow B\,[{}^{a}/_{x}] \in \mathbb{T}\mathrm{ype}^{(\alpha)} \end{array} \right\}\end{aligned}$$

- *For any limit* λ*,*
$$\mathbb{T}\mathrm{ype}^{(\lambda)} = \bigcup_{\alpha < \lambda} \mathbb{T}\mathrm{ype}^{(\alpha)}$$

Then $\mathrm{RED}_{(\mathrm{Type})}\,(T)$ *if and only if* T *is in the least fixpoint of* $\left(\mathbb{T}\mathrm{ype}^{(\alpha)}\right)_\alpha$*.*

Definition 2.3.8 (Reducible kinds)**.** *Similarly, we then can define reducible kinds by the increasing sequence:*

- $\mathbb{K}\mathrm{ind}^{(0)} = \{\mathrm{Type}\}$

- *For any ordinal* α*,*

$$\begin{aligned}\mathbb{K}\mathrm{ind}^{(\alpha+1)} &= \mathbb{K}\mathrm{ind}^{(\alpha)} \\ &\cup \left\{ \Pi(x : A)\ B \,\middle|\, \begin{array}{c} \mathrm{RED}_{(\mathrm{Type})}\,(A) \\ \forall a.\, \mathrm{RED}_{(A)}\,(a) \Rightarrow B\,[{}^{a}/_{x}] \in \mathbb{K}\mathrm{ind}^{(\alpha)} \end{array} \right\}\end{aligned}$$

- *For any limit* λ*,*
$$\mathrm{RED}^{(\lambda)} = \bigcup_{\alpha < \lambda} \mathrm{RED}^{(\alpha)}$$

Then $\mathrm{RED}_{(\mathrm{Kind})}\,(K)$ *if and only if* K *is in the least fixpoint of* $\left(\mathbb{K}\mathrm{ind}^{(\alpha)}\right)_\alpha$*.*

Properties of reducibility

Proposition 2.3.9 (Conversion does not change reducibility)**.** *If* $T \rightsquigarrow^* T'$ *and* $t \rightsquigarrow^* t'$ *then* $\mathrm{RED}_{(T)}(t) \Leftrightarrow \mathrm{RED}_{(T')}(t')$

Proof. Since the reducibility predicate definition, does a matching on disjoint cases considering only the normal form of a term and since the rewrite rules considered are confluent, this property holds. □

Proposition 2.3.10. *For all* T, t, *if* $\mathrm{RED}_{(\mathrm{Kind})}(T)$ *then*

1. $\mathrm{RED}_{(T)}(t) \Rightarrow \mathrm{WN}(t)$
2. $t \Downarrow b \Rightarrow \mathrm{RED}_{(T)}(t)$

Proof. We prove both mutually by induction.

1. - If $T = \mathrm{Type}$ it works by definition of the reducibility predicate.
   - If $T = \Pi(x : U)\, V$. Suppose $\mathrm{RED}_{(\mathrm{Kind})}(\Pi(x : U)\, V)$, we have $\mathrm{RED}_{(\mathrm{Type})}(U)$ and $\left(\forall u, \mathrm{RED}_{(U)}(u) \Rightarrow \mathrm{RED}_{(\mathrm{Kind})}(V\,[{}^{u}/_{x}])\right)$.
     Suppose we furthermore have $\mathrm{RED}_{(\Pi(x:U)\,V)}(t)$, we can then deduce $\left(\forall u, \mathrm{RED}_{(U)}(u) \Rightarrow \mathrm{RED}_{(V[{}^{u}/_{x}])}(t\,u)\right)$. Since $\mathrm{RED}_{(\mathrm{Type})}(U)$ and x is strongly neutral, by induction case 2, we obtain $\mathrm{RED}_{(U)}(x)$ so $\mathrm{RED}_{(\mathrm{Kind})}(V)$ and $\mathrm{RED}_{(V)}(t\,x)$. We can deduce $\mathrm{WN}(t\,x)$ by induction and then $\mathrm{WN}(t)$ by lemma 2.3.3.
2. - If $T = \mathrm{Type}$ it works by definition of the reducibility predicate.
   - If $T = \Pi(x : U)\, V$. Suppose $\mathrm{RED}_{(\mathrm{Kind})}(\Pi(x : U)\, V)$, we have $\mathrm{RED}_{(\mathrm{Type})}(U)$ and $\left(\forall u, \mathrm{RED}_{(U)}(u) \Rightarrow \mathrm{RED}_{(\mathrm{Kind})}(V\,[{}^{u}/_{x}])\right)$. We take a u such that $\mathrm{RED}_{(U)}(u)$. By induction case 1, there is a u' such that $u \Downarrow u'$ and by definition $b\,u'$ is strongly neutral. So by induction case 2, $\mathrm{RED}_{(V[{}^{u}/_{x}])}(t\,u)$ because $t\,u \Downarrow b\,u'$ which is strongly neutral. So $\mathrm{RED}_{(\Pi(x:U)\,V)}(t)$ by definition of the reducibility predicate. □

Proposition 2.3.11. *For all* T, t, *if* $\mathrm{RED}_{(\mathrm{Type})}(T)$ *then*

1. $\mathrm{RED}_{(T)}(t) \Rightarrow \mathrm{WN}(t)$
2. $t \Downarrow b \Rightarrow \mathrm{RED}_{(T)}(t)$

Proof. We prove both mutually by induction.

1. - If $T \Downarrow b$ or $T \Downarrow d\,t_1 \ldots t_n$ it works by definition of the reducibility predicate.
   - If $T \Downarrow \Pi(x : U)\, V$. Suppose $\mathrm{RED}_{(\mathrm{Type})}(\Pi(x : U)\, V)$, we have $\mathrm{RED}_{(\mathrm{Type})}(U)$ and $\left(\forall u, \mathrm{RED}_{(U)}(u) \Rightarrow \mathrm{RED}_{(\mathrm{Type})}(V\,[{}^{u}/_{x}])\right)$.
     Suppose we furthermore have $\mathrm{RED}_{(\Pi(x:U)\,V)}(t)$, we can then deduce $\left(\forall u, \mathrm{RED}_{(U)}(u) \Rightarrow \mathrm{RED}_{(V[{}^{u}/_{x}])}(t\,u)\right)$. Since $\mathrm{RED}_{(\mathrm{Type})}(U)$ and x

is strongly neutral, by induction case 2, we obtain $\mathrm{RED}_{(U)}\,(x)$ so $\mathrm{RED}_{(\mathrm{Type})}\,(V)$ and $\mathrm{RED}_{(V)}\,(t\,x)$. We can deduce $\mathrm{WN}(t\,x)$ by induction and then $\mathrm{WN}(t)$ by lemma 2.3.3.

2.
- If $T \Downarrow b$ or $T \Downarrow d\,t_1 \dots t_n$ it works by definition of the reducibility predicate.
- If $T \Downarrow \Pi(x : U)\,V$. Suppose $\mathrm{RED}_{(\mathrm{Type})}\,(T)$, we have $\mathrm{RED}_{(\mathrm{Type})}\,(U)$ and $\left(\forall u, \mathrm{RED}_{(U)}\,(u) \Rightarrow \mathrm{RED}_{(\mathrm{Type})}\,(V\,[{}^{u}/_{x}])\right)$. We take a u such that $\mathrm{RED}_{(U)}\,(u)$. By induction case 1, there is a u' such that $u \Downarrow u'$ and by definition $b\,u'$ is strongly neutral. So by induction case 2, $\mathrm{RED}_{\left(V[{}^{u}/_{x}]\right)}\,(t\,u)$ because $t\,u \Downarrow b\,u'$ which is strongly neutral. So $\mathrm{RED}_{(\Pi(x:U)\,V)}\,(t)$ by definition of the reducibility predicate. □

2.3.3 Reducibility in a context

Definition 2.3.12 (Reducibility of a substitution)**.** $\mathrm{RED}_{(x_1:T_1,\dots,x_n:T_n)}(\gamma)$ *holds whenever* $\forall i.\,\mathrm{RED}_{(\mathrm{Type})}\,(T_i\,\gamma) \wedge \mathrm{RED}_{(T_i\,\gamma)}\,(\gamma(x_i))$.

Proposition 2.3.13. *If* Γ *is closed,* $\mathrm{RED}_{\Gamma}(\gamma)$*,* $\mathrm{RED}_{(T\,\gamma)}\,(t)$*,* $x \notin \mathrm{supp}(\Gamma)$ *and* $x \notin \mathrm{FV}(T)$*, then* $\mathrm{RED}_{(\Gamma,x:T)}\,([\gamma, {}^{t}/_{x}])$

Proof. Assume given $(y : U)$ in $(\Gamma, x : T)$. We have two cases :

$x = y$ Then $T = U$ and $y\,[\gamma, {}^{t}/_{x}] = x\,[\gamma, {}^{t}/_{x}] = t$ and $T\,[\gamma, {}^{t}/_{x}] = T\,\gamma$ because x is not free in T. By assumption we have $\mathrm{RED}_{(T\,\gamma)}\,(t)$ thus we have $\mathrm{RED}_{\left(T\,[\gamma, {}^{t}/_{x}]\right)}\,(y\,[\gamma, {}^{t}/_{x}])$

$x \neq y$ Then $y\,[\gamma, {}^{t}/_{x}] = y\,\gamma$. From $\mathrm{RED}_{\Gamma}(\gamma)$ we have $\mathrm{RED}_{(U\,\gamma)}\,(\gamma(y))$ and we have $U[\gamma, {}^{t}/_{x}] = U\,\gamma$ because Γ is closed and x is not free in U. So $\mathrm{RED}_{\left(U\,[\gamma, {}^{t}/_{x}]\right)}\,(y\,[\gamma, {}^{t}/_{x}])$. □

2.3.4 Reducibility of well-typed terms

Definition 2.3.14. *Let* $\mathrm{RED}(\mathcal{F})$ *be the property :*

For all f*, if* $\mathcal{F}(f)$ *is of the form* $\Pi\overline{(x_i : T_i)}$ Type *then we have* $\mathrm{RED}_{(\mathrm{Kind})}\,(\mathcal{F}(f))$ *else we have* $\mathrm{RED}_{(\mathrm{Type})}\,(\mathcal{F}(f))$

Lemma 2.3.15. *If* $\mathrm{RED}(\mathcal{F})$ *holds, then, for all* f *such that* f *has no rewrite rule, we have* $\mathrm{RED}_{(\mathcal{F}(f))}\,(f)$*.*

Proof. Let $\mathcal{F}(f)$ be $\Pi\overline{(x_i : T_i)}\,U$. Take $\bar{t_i}$ such that for all i, $\mathrm{RED}_{\left(T_i\left[{}^{t_1}/_{x_1},\dots,{}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(t_i)$. By $\mathrm{RED}(\mathcal{F})$, we have $\mathrm{RED}_{(\mathrm{Sort})}\left(U\left[\overline{{}^{t_i}/_{x_i}}\right]\right)$ where Sort is Type or Kind depending on the case in which f is. By proposition 2.3.10 there are $\bar{u}_i$ such that $\bar{t_i} \Downarrow \bar{u}_i$. And since f has no rewrite rule, $f\,u_1 \dots u_n$ is strongly neutral. Then, using the second case of proposition 2.3.10 we obtain $\mathrm{RED}_{\left(U\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}\,(f\,u_1 \dots u_n)$.

Finally, we can conclude $\mathrm{RED}_{\left(U\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}\,(f\,t_1\ldots t_n)$ using proposition 2.3.9.

□

Theorem 2.3.16. *If* $\mathrm{RED}(\mathcal{F})$ *and* $\forall f.\,\mathrm{RED}_{(\mathcal{F}(f))}\,(f)$ *then*

$$\Gamma \vdash t : T \Rightarrow \left[\forall\gamma.\,\mathrm{RED}_{\Gamma}(\gamma) \Rightarrow \mathrm{RED}_{(T\,\gamma)}\,(t\,\gamma)\right]$$

Proof. We will prove by induction on the typing derivation that :

$$\forall\gamma.\,\mathrm{RED}_{\Gamma}(\gamma) \Rightarrow \mathrm{RED}_{(T\,\gamma)}\,(t\,\gamma)$$

Assume $\Gamma \vdash t : T$ and given a γ such that $\mathrm{RED}_{\Gamma}(\gamma)$.

- $\dfrac{\Gamma\ \text{well-formed}}{\Gamma \vdash \mathrm{Type} : \mathrm{Kind}}$

  By definition, we have $\mathrm{RED}_{(\mathrm{Kind})}\,(\mathrm{Type})$.

- $\dfrac{\Gamma\ \text{well-formed}}{\Gamma \vdash x : A}\ x : A \in \Gamma$

  We have $\mathrm{RED}_{\Gamma}(\gamma)$, meaning that we have $\mathrm{RED}_{(A\,\gamma)}\,(\gamma(x))$ for every $(x : A) \in \Gamma$.

- $\dfrac{\Gamma \vdash A : \mathrm{Type} \qquad \Gamma, x : A \vdash B : \mathrm{Sort}}{\Gamma \vdash \Pi(x : A)\,B : s}$ Sort is Kind or Type

  By induction hypothesis, we have $\mathrm{RED}_{(\mathrm{Type})}\,(A\,\gamma)$. Let's take a a such that $\mathrm{RED}_{(A\,\gamma)}\,(a)$. By proposition 2.3.13, we have $\mathrm{RED}_{(\Gamma,x:A)}\,([\gamma, {}^{a}/_{x}])$ and by induction $\mathrm{RED}_{(\mathrm{Sort})}\,(B\ [\gamma, {}^{a}/_{x}])$.

  It is exactly the definition of $\mathrm{RED}_{(\mathrm{Sort})}\,((\Pi(x : A)\,B)\,\gamma)$.

- $\dfrac{\Gamma \vdash A : \mathrm{Type} \qquad \Gamma, x : A \vdash B : \mathrm{Sort} \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).t : \Pi(x : A)\,B}$ Sort is Kind or Type

  By induction hypothesis, we have $\mathrm{RED}_{(\mathrm{Type})}\,(A\,\gamma)$. Let's take a a such that $\mathrm{RED}_{(A\,\gamma)}\,(a)$. By proposition 2.3.13, we have $\mathrm{RED}_{(\Gamma,x:A)}\,([\gamma, {}^{a}/_{x}])$ and by induction $\mathrm{RED}_{(B\,[\gamma, {}^{a}/_{x}])}\,(t\ [\gamma, {}^{a}/_{x}])$. Furthermore, $(\lambda(x : A).t)\,a \rightsquigarrow t\,[{}^{a}/_{x}]$ so $((\lambda(x : A).t)\,a)\ \gamma \rightsquigarrow t\,[\gamma, {}^{a}/_{x}]$. Then, by proposition 2.3.9, we can conclude that we have $\mathrm{RED}_{\left(B\left[{}^{a}/_{x}\right]\gamma\right)}\,(((\lambda(x : A).t)\,a)\ \gamma)$

  It is exactly the definition of $\mathrm{RED}_{((\Pi(x:A)\,B)\,\gamma)}\,((\lambda(x : A).t)\,\gamma)$.

- $\dfrac{\Gamma \vdash t : \Pi(x : A)\,B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B\,[{}^{u}/_{x}]}$

  By induction hypothesis, we have $\mathrm{RED}_{((\Pi(x:A)\,B)\,\gamma)}\,(t\,\gamma)$ and $\mathrm{RED}_{(A\,\gamma)}\,(u\,\gamma)$. Then, by definition of the reducibility of a Π we have $\mathrm{RED}_{\left(\left(B\left[{}^{u}/_{x}\right]\right)\gamma\right)}\,((t\,u)\,\gamma)$.

- $\dfrac{\Gamma, x_1 : T_1, \ldots, x_i : T_i \vdash T_{i+1} : \mathrm{Type}}{\Gamma \vdash d : \mathcal{D}(d)}\ \mathcal{D}(d) = \Pi\overline{(x_i : T_i)}\ \mathrm{Type}$

Assume given $\bar{t}_i$ such that $\mathrm{RED}_{\left(T_i\left[\gamma, {}^{t_1}/_{x_1}, \ldots, {}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(t_i)$. By proposition 2.3.10, there exist $\bar{u}_i$ such that $t_i \Downarrow u_i$. By proposition 2.3.9, we have $\mathrm{RED}_{\left(T_i\left[\gamma, {}^{t_1}/_{x_1}, \ldots, {}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(u_i)$. Furthermore, by induction hypothesis, we have $\mathrm{RED}_{(\mathrm{Type})}\,(T_i\,\gamma)$.

Hence, we obtain $\mathrm{RED}_{(\mathrm{Type})}\,(d\,t_1 \ldots t_k)$, and this is exactly the definition of $\mathrm{RED}_{((\Pi(x_1:T_1)\ldots(x_k:T_k)\ \mathrm{Type})\,\gamma)}\,(d)$

- $$\frac{\Gamma, y_1 : U_1, \ldots, y_i : U_i \vdash U_{i+1} : \mathrm{Type} \quad \Gamma, \overline{x_i : U_i} \vdash \tau_j : T_j}{\Gamma \vdash c : \mathcal{C}(c)} \begin{cases} \mathcal{C}(c) = \Pi\overline{(x_i : U_i)}\,(d\,\tau_1 \ldots \tau_k) \\ \mathcal{D}(d) = \Pi\overline{(x_i : T_i)}\ \mathrm{Type} \end{cases}$$
  Assume given $\bar{u}_i$ such that $\mathrm{RED}_{\left(U_i\left[\gamma, {}^{u_1}/_{x_1}, \ldots, {}^{u_{i-1}}/_{x_{i-1}}\right]\right)}\,(u_i)$. By proposition 2.3.10, there exist $\bar{v}_i$ such that $u_i \Downarrow v_i$. By proposition 2.3.9, we have $\mathrm{RED}_{\left(U_i\left[\gamma, {}^{u_1}/_{x_1}, \ldots, {}^{u_{i-1}}/_{x_{i-1}}\right]\right)}\,(v_i)$. Furthermore, by induction hypothesis, we have $\mathrm{RED}_{(\mathrm{Type})}\,(U_i\,\gamma)$.

  Hence, we obtain $\mathrm{RED}_{\left((d\,\tau_1 \ldots \tau_k)\left[\gamma, \overline{{}^{u_i}/_{x_i}}\right]\right)}\,((c\,u_1 \ldots u_m)\ \gamma)$, and this is exactly the definition of $\mathrm{RED}_{((\Pi(x_1:U_1)\ldots(x_m:U_m)\,(d\,\tau_1\ldots\tau_k))\,\gamma)}\,(c)$

- $$\frac{\Gamma, x_1 : T_1, \ldots, x_i : T_i \vdash T_{i+1} : \mathrm{Type} \quad \Gamma, \overline{x_i : T_i} \vdash U : \mathrm{Type}}{\Gamma \vdash f : \mathcal{F}(f)}\ \mathcal{F}(f) = \Pi\overline{(x_i : T_i)}\,U$$
  By hypothesis, we have $\mathrm{RED}_{(\mathcal{F}(f))}\,(f)$. Since $\mathcal{F}(f)$ is closed, we have $\mathrm{RED}_{(\mathcal{F}(f)\,\gamma)}\,(f\,\gamma)$.

- $$\frac{\Gamma, x_1 : T_1, \ldots, x_i : T_i \vdash T_{i+1} : \mathrm{Type}}{\Gamma \vdash f : \mathcal{F}(f)}\ \mathcal{F}(f) = \Pi\overline{(x_i : T_i)}\ \mathrm{Type}$$
  The same argument here show that we have $\mathrm{RED}_{(\mathcal{F}(f)\,\gamma)}\,(f\,\gamma)$.

- $$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A : \mathrm{Sort} \qquad \Gamma \vdash B : \mathrm{Sort}}{\Gamma \vdash t : B} \begin{cases} A \leftrightsquigarrow^* B \\ \mathrm{Sort\ is\ Kind\ or\ Type} \end{cases}$$
  By induction, we have $\mathrm{RED}_{(A\,\gamma)}\,(t\,\gamma)$ and by proposition 2.3.9, we obtain that $\mathrm{RED}_{(B\,\gamma)}\,(t\,\gamma)$, because from $A \rightsquigarrow^* B$, we have $A\,\gamma \rightsquigarrow^* B\,\gamma$. □

2.4 Call-relation and reducibility of defined functions

2.4.1 Call relation

Definition 2.4.1 (Formal call). *We define* $(f, (p_1, \ldots, p_m)) \succ (g, (u_1, \ldots, u_n))$ *by :*

- *there is a* k *such that* $f\,p_1 \ldots p_k \to s$ *is in* $\mathbb{R}$*,*
- $\mathrm{ar}(f) = m$*,* $\mathrm{ar}(g) = n$*,*

- *$g\, u_1 \dots u_n$ is a subterm of $s\, p_{k+1} \dots p_m$.*

These types of calls are the one on which the *Size-Change Principle* algorithm, described section 1.2, will be performed. However, we will use another notion of call to state some interesting results about our reducibility predicates.

Definition 2.4.2 (Instantiated call)**.** *$(f,(t_1,\dots,t_m)) \widetilde{\succ} (g,(v_1,\dots,v_n))$ holds if there exist $p_1,\dots,p_m,u_1,\dots,u_n$ and a substitution γ such that :*

- *$(f,(p_1,\dots,p_m)) \succ (g,(u_1,\dots,u_n))$,*
- *$\forall i \leqslant m. t_i \rightsquigarrow^* p_i\, \gamma$,*
- *$\forall i \leqslant m. \mathrm{WN}(t_i)$,*
- *$\forall j \leqslant n. v_j = u_j\, \gamma$.*

2.4.2 Proof of reducibility for defined functions

Theorem 2.4.3. *If $\widetilde{\succ}$ is well-founded and $\mathrm{RED}(\mathcal{F})$ then*

$$\forall f.\, \mathrm{RED}_{(\mathcal{F}(f))}\,(f)$$

Proof. As the hypothesis suggests, we want to perform a well-founded induction on $\widetilde{\succ}$. But some work is required before doing this induction.

For all $f \in \mathbb{F}$, let's write

$$\mathcal{F}(f) = \Pi(x_1 : T_{f,1}) \dots (x_{\mathrm{ar}(f)} : T_{f,\mathrm{ar}(f)})\, U_f$$

We will denote by $\Phi(f,\bar{t})$ the property :

$$\forall i.\, \mathrm{RED}_{\left(T_{f,i}\left[{}^{t_1}/_{x_1},\dots,{}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(t_i) \Rightarrow \mathrm{RED}_{\left(U_f\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}\,(f\,\bar{t})$$

Here, we should note that $\forall f. \forall \bar{t}. \Phi(f,\bar{t})$ is the definition of $\forall f.\, \mathrm{RED}_{(\mathcal{F}(f))}\,(f)$ which is our goal.

Assume given f and $t_1,\dots,t_{\mathrm{ar}(f)}$ such that

$$\begin{aligned}
&\forall i.\, \mathrm{RED}_{(\mathrm{Type})}\left(T_{f,i}\left[{}^{t_1}/_{x_1},\dots,{}^{t_{i-1}}/_{x_{i-1}}\right]\right)\\
&\forall i.\, \mathrm{RED}_{\left(T_{f,i}\left[{}^{t_1}/_{x_1},\dots,{}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(t_i)\\
&\forall g. \forall \bar{u}. (f,\bar{t}) \widetilde{\succ} (g,\bar{u}) \Rightarrow \Phi(g,\bar{u})
\end{aligned}$$

From proposition 2.3.10, there exist $\bar{v}$ such that $\bar{t} \Downarrow \bar{v}$. Then we have two cases:

- $f\,\bar{v}$ is normal, then since f is fully applied, $f\,\bar{v}$ is strongly neutral and applying the other case of the proposition 2.3.10 we obtain $\mathrm{RED}_{\left(U_f\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}\,(f\,\bar{t})$

- Otherwise $f\,\bar{v}$ matches a rewrite rule declared by the user. So there are a k, a rewrite rule $f\,p_1 \dots p_k \to s$ and a substitution ϑ such that $\forall i \leqslant k.v_i = p_i\,\vartheta$. Then $s = \lambda(x_{k+1} : T_{k+1}) \dots (x_{\mathrm{ar}(f)} : T_{\mathrm{ar}(f)}).s_0$ and
$$f\,v_1 \dots v_{\mathrm{ar}(f)} \rightsquigarrow^* s_0 \left[\vartheta, {}^{v_{k+1}}/_{x_{k+1}}, \dots, {}^{v_{\mathrm{ar}(f)}}/_{x_{\mathrm{ar}(f)}}\right]$$
We will denote γ the substitution $\left[\vartheta, {}^{v_{k+1}}/_{x_{k+1}}, \dots, {}^{v_{\mathrm{ar}(f)}}/_{x_{\mathrm{ar}(f)}}\right]$ and prove later that $\mathrm{RED}_{\left(U_f\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}(s_0\,\gamma)$. Once we have it, by proposition 2.3.9, we obtain $\mathrm{RED}_{\left(U_f\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}(f\,\bar{t})$

In both cases, we can conclude $\Phi(f, \bar{t})$, then we have :

$$\forall f.\forall \bar{t}.\left(\forall g.\forall \bar{u}.(f, \bar{t})\widetilde{\succ}(g, \bar{u}) \Rightarrow \Phi(g, \bar{u})\right) \Rightarrow \Phi(f, \bar{t})$$

and then by the induction principle on the well-founded relation $\widetilde{\succ}$, we can conclude $\forall f.\forall \bar{t}.\Phi(f, \bar{t})$ which is the definition of

$$\forall f.\,\mathrm{RED}_{(\mathcal{F}(f))}(f)$$

Remaining goal 1. We will now show that if $f\,\bar{t}$ matches a rewrite rule declared by the user, we have
$$\mathrm{RED}_{\left(U_f\left[\overline{{}^{t_i}/_{x_i}}\right]\right)}(s_0\,\gamma)\,.$$

Proof. Since we have supposed that the left-hand side of all the rewrite rules are typable, there is a Δ_0 such that:

$$\forall i \leqslant m.\Delta_0 \vdash p_i : T_{f,i}\left[{}^{p_1}/_{x_1} \dots {}^{p_{i-1}}/_{x_{i-1}}\right]$$
$$\forall m < i \leqslant \mathrm{ar}(f).\Delta_0 \vdash x_i : T_{f,i}\left[{}^{p_1}/_{x_1} \dots {}^{p_m}/_{x_m}\right]$$

Since all the rewrite rules preserve typing, we know that $\Delta_0 \vdash s_0 : U_f\ \left[\overline{{}^{p_i}/_{x_i}}\right]$.

We will show that for all subterms s of s_0, all contexts Θ extending Δ_0, all U and all substitutions σ which are equal to γ on Δ_0, we have:

$$\left(\Theta \vdash s : U \wedge \mathrm{RED}_\Theta(\sigma) \wedge \mathrm{RED}_{(\mathrm{Type})}(U\,\sigma)\right) \Rightarrow \mathrm{RED}_{(U\,\sigma)}(s\,\sigma)$$

Once it is proved, since we have $\mathrm{RED}_{\Delta_0}(\gamma)$, then by taking $s = s_0$, $\Theta = \Delta_0$, $U = U_f$ and $\sigma = \gamma$ we obtain the expected result.

Remaining goal 2. Let firstly justify that $\mathrm{RED}_{\Delta_0}(\gamma)$.

Proof. To prove it, we will have to perform an induction on the number of constructor under which is each variable. Let's take an i between 1 and m.

If p_i is a variable, we have by hypothesis that $\mathrm{RED}_{\left(T_{f,i}\left[{}^{t_1}/_{x_1}, \dots, {}^{t_{i-1}}/_{x_{i-1}}\right]\right)}(t_i)$. And $t_i = p_i\,\vartheta = \gamma(p_i)$.

Otherwise there is a c_i such that $\mathcal{C}(c_i) = \Pi(x_1 : U_1) \ldots (x_k : U_k)\,(d\,\tau_1 \ldots\, \tau_l)$ and $p_i = c_i\, p_{i,1} \ldots p_{i,k}$. We have by hypothesis that $\mathrm{RED}_{\left(T_{f,i}\left[{}^{t_1}/_{x_1},\ldots,{}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(t_i)$. And $t_i = p_i\,\vartheta = c_i\,(p_{i,1}\,\vartheta) \ldots (p_{i,k}\,\vartheta)$ and by definition of the reducibility of a term which begins with a constructor, we have $\mathrm{RED}_{\left(U_j\left[{}^{p_{i,1}}/_{x_1}\ldots{}^{p_{i,j-1}}/_{x_{j-1}}\right]\right)}\,(p_{i,j})$ and by induction, we can keep going under the constructors until we reach a variable.

Remaining goal 3. For all subterms s of s_0, all contexts Θ extending Δ_0, all U and all substitutions σ which are equal to γ on Δ_0, we have :

$$\big(\Theta \vdash s : U \wedge \mathrm{RED}_\Theta(\sigma) \wedge \mathrm{RED}_{(\mathrm{Type})}\,(U\,\sigma)\big) \Rightarrow \mathrm{RED}_{(U\,\sigma)}\,(s\,\sigma)$$

Proof. We will show it by induction on the subterms of s_0, knowing that the subterm-order is well-founded.

Let's take a subterm s of s_0, a context Θ extending Δ_0, a U and a substitution σ which is equal to γ on Δ_0, such that $\Theta \vdash s : U$, $\mathrm{RED}_{(\mathrm{Type})}\,(U\,\sigma)$ and $\mathrm{RED}_\Theta(\sigma)$. We will proceed by case analysis on the form of s. Each form induces the end of the derivation of $\Theta \vdash s : U$. Here for readability, some hypothesis, which are necessary to really perform the inferences but which are not relevant to perform the induction are omitted.

- $$\frac{\Theta \vdash s_i : T_i}{\Theta \vdash g\, s_1 \ldots s_n : U}\begin{cases}\mathcal{F}(g) = \Pi\overline{(x_1 : T_1)}.U_{\mathrm{ar}(g)+1}\\ \text{If } n > \mathrm{ar}(g), U_{\mathrm{ar}(g)+1}\left[{}^{s_1}/_{x_1},\ldots,{}^{s_{\mathrm{ar}(g)}}/_{x_{\mathrm{ar}(g)}}\right]\\ \qquad \leftrightsquigarrow^* \Pi(x_{\mathrm{ar}(g)+1} : T_{\mathrm{ar}(g)+1}).U_{\mathrm{ar}(g)+2}\\ \forall\, \mathrm{ar}(g)+1 < i \leqslant n. U_i\left[{}^{s_{i-1}}/_{x_{i-1}}\right] \leftrightsquigarrow^* \Pi(x_i : T_i).U_{i+1}\\ U \leftrightsquigarrow^* U_{n+1}\left[{}^{s_n}/_{x_n}\right]\end{cases}$$

  By induction, we have $\mathrm{RED}_{(T_i\,\sigma)}\,(s_i\,\sigma)$. Since $g\,s_1\,\ldots\,s_n$ is a subterm of s_0 and $\mathrm{ar}(g) \geqslant n$, we have $(f,\bar{p}) \succ (g,\bar{s}_i)$. Here we must recall that we are under the hypothesis that

  $$\forall i.\,\mathrm{RED}_{\left(T_{f,i}\left[{}^{t_1}/_{x_1},\ldots,{}^{t_{i-1}}/_{x_{i-1}}\right]\right)}\,(t_i) \;\text{ and }\; \forall g.\forall \bar{u}.(f,\bar{t})\widetilde{\succ}(g,\bar{u}) \Rightarrow \Phi(g,\bar{u})$$

  Since σ and γ are equal on Δ_0 which is a context in which all the variables appearing in the p_i are bounded, from $t_i \rightsquigarrow^* p_i\,\gamma$, we can deduce $t_i \rightsquigarrow^* p_i\,\sigma$. Furthermore, by proposition 2.3.10 we have $\mathrm{WN}(t_i)$, so we get

  $$(f,\bar{t})\widetilde{\succ}(g,\overline{s_i\,\sigma})$$

  Then we get $\Phi(g,\overline{s_i\,\sigma})$, hence $\mathrm{RED}_{\left(U_{\mathrm{ar}(g)+1}\left[\overline{{}^{s_i}/_{x_i}}\right]\right)}\,\big((g\,s_1 \ldots s_{\mathrm{ar}(g)})\,\sigma\big)$, then for all $j \geqslant \mathrm{ar}(g)$, $\mathrm{RED}_{\left(U_{j+1}\left[\overline{{}^{s_i}/_{x_i}}\right]\right)}\,((g\,s_1 \ldots s_j)\,\sigma)$ and by the conversion rule, $\mathrm{RED}_{(U\,\sigma)}\,(s\,\sigma)$.

- $$\frac{\Theta \vdash s_i : T_i\;\left[{}^{s_1}/_{x_1},\ldots,{}^{s_{i-1}}/_{x_{i-1}}\right]}{\Theta \vdash d\,s_1 \ldots s_k : \mathrm{Type}}\;\mathcal{D}(s) = \Pi\overline{(x_i : T_1)}\;\mathrm{Type}$$

By induction, we have $\text{RED}_{\left(T_i \left[{}^{s_1}/_{x_1},\ldots,{}^{s_{i-1}}/_{x_{i-1}}\right] \sigma\right)} \, (s_i \, \sigma)$.
Then $\text{RED}_{(\text{Type})} \, ((d \, s_1 \ldots s_k) \, \sigma)$ which is our goal.

- $$\frac{\Theta \vdash s_i : T_i \, \left[{}^{s_1}/_{x_1}, \ldots, {}^{s_{i-1}}/_{x_{i-1}}\right]}{\Theta \vdash c \, s_1 \ldots s_m : U} \begin{cases} \mathcal{C}(c) = \Pi\overline{(x_i : T_1)} \, (d \, \tau_1 \ldots \tau_k) \\ U \leftrightsquigarrow^* (d \, \tau_1 \ldots \tau_k) \left[\overline{{}^{s_i}/_{x_i}}\right] \end{cases}$$
  By induction, we have $\text{RED}_{\left(T_i \left[{}^{s_1}/_{x_1},\ldots,{}^{s_{i-1}}/_{x_{i-1}}\right] \sigma\right)} \, (s_i \, \sigma)$.
  Then $\text{RED}_{(d \, (\tau_1 \, \sigma) \ldots (\tau_k \, \sigma))} \, ((c \, s_1 \ldots s_m) \, \sigma)$ which by the conversion rule is $\text{RED}_{(U \, \sigma)} \, (s \, \sigma)$.

- $$\frac{\Theta \vdash t : \Pi(x : T_1).T_2 \qquad \Theta \vdash u : T_1}{\Theta \vdash t \, u : U} \begin{cases} U \leftrightsquigarrow^* T_2 \, [{}^{u}/_{x}] \\ t \text{ does not start by a constructor} \\ \qquad \text{or a function symbol} \end{cases}$$

  By induction, we have $\text{RED}_{(T_1 \, \sigma)} \, (u \, \sigma)$ and $\text{RED}_{(\Pi(x:T_1).T_2 \, \sigma)} \, (t \, \sigma)$, hence by definition we have $\text{RED}_{(U)} \, ((t \, u) \, \sigma)$.

- $$\frac{\Theta, x : T \vdash s_1 : V}{\Theta \vdash \lambda(x : T).s_1 : \Pi(x : T) \, V}$$
  By hypothesis, we have $\text{RED}_{(\text{Sort})} \, ((\Pi(x : T) \, V) \, \sigma)$. Let's take a t such that $\text{RED}_{(T \, \sigma)} \, (t)$. Then by definition $\text{RED}_{(\text{Sort})} \, (V \, [\sigma, {}^{t}/_{x}])$ $\Theta, x : T$ enrich Δ_0 because Θ do so and $[\sigma, {}^{t}/_{x}]$ is equal to γ on Δ_0 because σ is. So by induction, we have $\text{RED}_{\left(V \left[\sigma, {}^{t}/_{x}\right]\right)} \, (s_1 \, [\sigma, {}^{t}/_{x}])$. Knowing that $((\lambda(x : T).s_1) \, \sigma) \, t \rightsquigarrow s_1 \, [\sigma, {}^{t}/_{x}]$, we have $\text{RED}_{\left(V \left[\sigma, {}^{t}/_{x}\right]\right)} \, (((\lambda(x : T).s_1) \, \sigma) \, t)$ and since t was arbitrary, $\text{RED}_{((\Pi(x:T) \, V) \, \sigma)} \, ((\lambda(x : T).s_1) \, \sigma)$ which was our goal. □

By combining theorem 2.4.3, theorem 2.3.16 and proposition 2.3.10 we obtain the following corollary, which was the aim of our work, because it gives conditions under which typability implies weak normalisation.

Corollary 2.4.4. *If* $\tilde{\succ}$ *is well-founded and* $\text{RED}(\mathcal{F})$ *then*

$$\vdash t : T \Rightarrow \text{WN}(t)$$

Once we have this result, the *Size-Change Principle* can be use to verify if $\tilde{\succ}$ is well-founded. Thanks to the modularity of Wahlstedt's work, the demonstration given in [Wah07, Chapter 4] can be re-used to prove this.

Inspired by the work of Frédéric Blanqui [Bla01], an adaptation of the reducibility predicates in order to show *strong normalisation* has been tried during the internship. We manage to show the theorem 2.3.16 with a reducibility predicate adapted for *strong normalisation.* Unhappily, we did not had the time to adapt the proof of theorem 2.4.3 for this predicate, if it is possible, and we did not used the λΠ-calculus modulo theory, but the Wahldstedt's system to do this adaptation. This is the reason why, this on-going work is not presented in this report, since it would have needed to introduce another system, which is not the one in which our problem was designed, making this report much heavier and probably less clear, for a not finished work. We can expect this new adaptation to be foundable in another document soon, once it will be achieved.

Nomenclature

$<$, $=$, ∞	Expressing the relation between the arguments of the caller and the called functions in a rewrite rule, page 5
$\approx_{\mathcal{D}}$	Equivalence on set constructors, page 13
$\overset{\sim}{\succ}$	Instantiated call, page 23
δ, σ, ϑ	Substitutions, page 12
$\Downarrow$	Relation linking a term to its normal form, page 15
$\llbracket d \rrbracket$	Interpretation of the elementary types constructed with d, page 17
$\Pi\overline{(x_i : T_i)}\, U$	Function of multiple arguments, page 11
$\preccurlyeq_{\mathcal{D}}$	Precedence on set constructors, page 12
$\rightsquigarrow$	The rewriting relation, union of the rules declared by the user and β-reduction, page 15
$\rightsquigarrow^*$	The transitive and reflexive closure of the rewriting relation, page 15
$\tau\,[{}^{t}/_{x}]$	Substitution of x by t in τ, page 12
$\succ$	Formal call, page 23
ar	Arity of a symbol, page 12
b	Strongly neutral term, page 16
c	Element constructor, page 12
$\mathcal{C}(c)$	Type associated to the element constructor c, page 12
d	Set constructor, page 12
$\mathcal{D}(d)$	Type associated to the set constructor d, page 12
f	Defined function, page 12

Index

Bibliography

[ABC$^+$16] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Fédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a Logical Framework based on the λΠ-Calculus Modulo Theory. 36 pages, 2016.

[BCH12] Mathieu Boespflug, Quentin Carbonneaux, and Olivier Hermant. The λΠ-calculus Modulo as a Universal Proof Language. In Tjark Weber David Pichardie, editor, *the Second International Workshop on Proof Exchange for Theorem Proving (PxTP 2012)*, volume Vol. 878, pages pp. 28–43, Manchester, United Kingdom, June 2012.

[Bla01] Frédéric Blanqui. Definitions by rewriting in the calculus of constructions. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 9–18, 2001.

[Boe11] Mathieu Boespflug. *Design and implementation of a proof verifying kernel for the λΠ-calculus modulo.* Theses, Ecole Polytechnique X, January 2011.

[CD07] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 102–117, 2007.

[Dow17] Gilles Dowek. Models and termination of proof reduction in the lambda pi-calculus modulo theory. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 109:1–109:14, 2017.

[Hyv13] Pierre Hyvernat. The Size-Change Termination Principle for Constructor Based Languages. 19 pages + 3 pages d'appendice, March 2013.

[JB08] Neil D. Jones and Nina Bohr. Call-by-value termination in the untyped lambda-calculus. *Logical Methods in Computer Science*, 4(1), 2008.

[LJBA01] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '01, pages 81–92, New York, NY, USA, 2001. ACM.

[LR16] Rodolphe Lepigre and Christophe Raffalli. Subtyping-based type-checking for system f with induction and coinduction. *arXiv preprint arXiv:1604.01990*, 2016.

[OR94] Vincent van Oostrom and Femke van Raamsdonk. Weak orthogonality implies confluence: The higher order case. In *Proceedings of the Third International Symposium on Logical Foundations of Computer Science*, LFCS '94, pages 379–392, London, UK, UK, 1994. Springer-Verlag.

[Sai15] Ronan Saillard. *Typechecking in the lambda-Pi-Calculus Modulo : Theory and Practice. (Vérification de typage pour le lambda-Pi-Calcul Modulo : théorie et pratique).* PhD thesis, Mines ParisTech, France, 2015.

[Wah07] David Wahlstedt. *Dependent Type Theory with Parameterized First-Order Data Types and Well-Founded Recursion.* PhD thesis, Chalmers University of Technology, 2007. ISBN 978-91-7291-979-2.