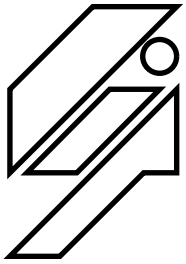
Marking techniques for extraction.

Frederic Prost



Laboratoire de l'Informatique du Parallélisme

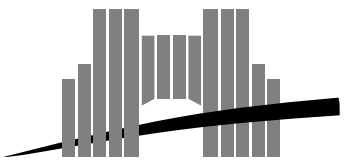
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Marking techniques for extraction

Frédéric Prost

December 95

Research Report N^{o} 95-47



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Marking techniques for extraction

Frédéric Prost

December 95

Abstract

Constructive logic can be used to consider program specifications as logical formulas. The advantage of this approach is to generate programs which are certified with respect to some given specifications.

The programs created in such a way are not efficient because they may contain large parts with no computational meaning. The elimination of these parts is an important issue. Many attempts to solve this problem have been already done. We call this extracting procedure.

In this work we present a new way to understand the extraction problem. This is the marking technique. This new point of view enables us, thanks to a high abstraction level, to unify what was previously done on the subject. It enables also to extend to higher–order languages some pruning techniques developed by Berardi and Boerio, which were only used in first and second order language.

Keywords: Program Verification, Type Theory, Logic, Program proof, Extraction, Marking.

Résumé

La logique constructive peut être utilisée pour produire des programmes en les considérant comme des formules de la logique. L'avantage d'une telle méthodologie est de produire des programmes dont on est sûr qu'ils vérifient certaines spécifications.

Les programmes générés par ces méthodes sont en général peu efficaces car ils comportent de larges parts qui ne servent pas au calcul final. Pour les repérer et les effacer, de nombreuses techniques ont déjà été développées. On parle de *technique d'extraction*.

Dans ce travail nous donnons une nouvelle manière d'approcher ce problème. Il s'agit du marquage. Ce nouvel angle de vision permet, grâce à un niveau d'abstraction élevé, d'unifier les connaissances relatives à l'extraction. Il permet aussi l'extension aux ordres supérieurs des techniques de taillage exploitées par Berardi et Boerio, qui n'étaient valables qu'aux premier et au second ordre.

Mots-clés: Vérification de Programmes, Théorie des types, Logique, Preuves de programmes, Extraction, Marquage.

Marking techniques for extraction

Frédéric Prost

December 15, 1995

Abstract

Constructive logic can be used to consider program specifications as logical formulas. The advantage of this approach is to generate programs which are certified with respect to some given specifications.

The programs created in such a way are not efficient because they may contain large parts with no computational meaning. The elimination of these parts is an important issue. Many attempts to solve this problem have been already done. We will talk about extracting procedure.

In this work we present a new way to understand the extraction problem. It is the marking technique. This new point of view enable us, thanks to a high abstraction level, to unify what was previously done on the subject. It allows also to extend some pruning techniques developed by Berardi and Boerio, which were only used in first and second order language to higher–order languages.

Keywords : Logic, Programs proof, Extraction, marking.

Contents

1 Introduction

Methods of development of programs from the proof of their specifications are based on the Curry–Howard isomorphism. For several years now systems using this approach have been developed: Nuprl [4], Coq [5] and PX are the most famous examples.

A proof showing that, for every object x of type A, there exists y of type B that satisfies a relation $R(x, y)$ defines implicitly a computable function from A to B. A constructive demonstration of a proposition of this kind enables actually to build the function. In other words, we can construct a program from a proof of its specifications.

This approach of programming is interesting, for it leads to error–free programs from its nature. Nevertheless nothing is free of charge and the cost of the certainty is time. Programs generated in this way are inefficient. The codes recovered contain too much useless terms from the algorithmic point of view. Indeed a program just contains what is worthy to realize an algorithm. On the contrary a proof formalizes a great deal of information which is not needed to compute the final result. For example the description of the task to carry out, verifications of the method, the control of dependency between objects are contained in the proof but are not needed for the computation of the result.

It seems a good idea to delete those inefficient parts from the code to get a "good" program. We will use the word of *extraction* to characterize this work. Several extraction techniques have been studied. One can approximately divide those ones into two ways.

The first point of view is to realize the searched optimization within the syntax of the terms. C. Paulin in [9] develops this approach for a higher order system: the Calculus of Construction. The basic idea is to duplicate the system. One part is used for the logical annotations without computational meaning and the second is used to express the parts of the program useful for the computation . This technique works while the program developer is making his proof. One can easily find out what parts are inefficient. these are the ones which are typed in a certain predefined way. The last step is to erase them in order to get an efficient program.

The second approach consists in looking at the proof tree, and, by an appropriate study, to locate subtrees without computational meaning. This point of view has been initiated by Goad in [7], and enriched by Takayama in [11]. The last developments of this approach have been done by Berardi and Boerio in [1] and [2]. The basic idea is that the proof contains a lot of information regarding the corresponding program. These techniques propose to use these informations in order to simplify the program obtained.When the proof is done, we seek what parts of the tree are useless and prune them (we talk about *Pruning* techniques). These methods have only been studied in languages of order less than or equal to two. In [3], Berardi and Boerio extend the technique to take in account more simplifications by a mechanism of subtyping. By weakening some typing rules relatively to the application, they manage to avoid the formation of ill typed terms, while the simplification is performed.

We propose to develop a new approach of extraction by the means of *marks*. The idea is simple. We are going to annotate the terms to express their computational meaning. So we are going to introduce in the typing rules some constraints regarding the marking of terms. The aim is to formalize within the logic what was already done technically by Berardi–Boerio's pruning.

The first goal is to unify, thanks to this new formalization, the different extracting methods already known. Indeed, we will prove that marking technique is able to simulate all of them. This new level of abstraction permits to extend to higher order language some result already obtained in first and second order by Berardi and Boerio.

The paper is organized as follows. We first give a brief presentation of the most widely known extracting techniques : the ones coming from the pruning, first developed in the first order by Berardi and then extended by Boerio to the second order. We will see next the techniques proposed by C. Paulin for the Calculus of Constructions. Finally, we will give a new way to approach both techniques by our marking techniques, and we will extend the previous results on pruning to higher order thanks to marking techniques.

2 Miscellaneous approaches of extraction

2.1 The Turin school

2.1.1 The Pruning

All the techniques developed by this school are based on a similar way to optimize terms : the pruning.

The pruning is defined as a relation between two trees. One says that a tree A is a pruning of another tree B, if we can obtain A by erasing some subtrees of B.

Of course one can see any terms of a given formal language as a labeled tree. One naturally derives a notion of pruning with respect to terms of a formal language. If this term is produced from a functional system (typed or not), the pruning can be seen as an optimization. Indeed, to prune a term corresponds to delete some code judged useless. As the pruning suppresses some subterms one can expect that a term a, which is a pruned version of a term b, can be evaluated in a easier way.

The Turin school has studied the behavior of terms in some typed systems regarding the pruning. First, Berardi gives results for a typed system to the first order, results that Boerio then extends to the second order. In both cases, the idea is to suppress the maximum of subterms while staying invariant relatively to some observational equivalence. Schematically their paradigm is " since I do not modify the input-output function of a term I can suppress a subterm".

2.1.2 The Berardi approach for the first order

In [1] Berardi studies the effects of pruning on terms of a variant of the Gödel system $\mathcal{T}$. He shows that if two terms have the same type and if one is a pruned version of the other, then they have the same input–output function. Moreover, for all terms, the existence of a minimal term (with respect to the notion of pruning) is proved.

The modified Gödel system $\mathcal{T}$

Informally, this system is a simply typed λ–calculus having an atomic type N for integers, and the schema of primitive recursion on N. It contains the arrow types, product types, projections, applications, and λ–abstractions (we will use the notation of the Calculus of Construction for λ–abstractions. We will write $[x : N]x$ instead of $\lambda x^N.x$). There are also algebraic constants : 0 and S of respective types N and $N \to N$ and the constant of higher order for the primitive recursion rec_A of type $N \times A \times (N \times A \to A) \to A$, for every type A.

As the goal is to prune terms, we introduce a type U and a constant $\emptyset$ denoting the *hole* which stays after we have removed the useless subterm. $\emptyset$ is be the only inhabitant of type U.

In a more formal way, this system is defined by :

1. If α is U or N (so an atomic type) then the types are defined by :

$$A \quad ::= \alpha \ \mid A \times A \ \mid A \to A$$

2. The set of constants is represented by an initial context :

$$\Sigma_{\mathcal{T}} \stackrel{def}{=} \emptyset : U, 0 : N, S : N \to N, rec_A : N \times A \times (N \times A \to A) \to A$$

3. If A is a type, x a variable of type A, and c a constant of $\mathcal{T}$. the pseudo-terms are defined by :

$$t \quad ::= c \ \mid x \ \mid [x : A]t \ \mid (t \ \ t) \ \mid < t, t > \mid (\pi_{i,A_1,A_2} \ \ t)$$

4. Let t, f, a_1, a_2 be pseudo-terms, A, B, A_1, A_2 types

$$\frac{c : A \in \Gamma}{\Gamma \vdash c : A} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash [x : A]t : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash (f \quad a) : B}$$

$$\frac{\Gamma \vdash a_1 : A_1 \quad \Gamma \vdash a_2 : A_2}{\Gamma \vdash < a_1, a_2 > : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash (\pi_{i,A_1,A_2} \quad t) : A_i}$$

5. The reductions are the following. $\to_\beta$ is the usual β–reduction, $\to_\alpha$ the standard α-conversion, $\to_\pi$ corresponds to a projection and rec_0 with rec_s are the reductions associated to the usual rules of recurrence.

   We will write $\to_1$ for $\to_\alpha \cup \to_\beta \cup \to_{rec_s} \cup \to_{rec_0}$. The notations $\to^*, \to_n$ will have the usual meaning with respect to $\to_1$. $=_R$ denotes the symmetric and transitive closure of $\to^*$.

The pruning

Lets A, B be two types, s a constant or a variable, x a variable, t, u two terms. We define $\leq$ by induction. $e \leq e'$, where e and e' are two well formed expressions (i.e. that we can type) of the system, is read "e is a pruning of e'.

$(\emptyset)$	$\emptyset \leq t$	$for \ \ any \ \ term \ \ t.$
(U)	$U \leq A$	$for \ \ any \ \ type \ \ A.$
$(atom)$	$\alpha \leq \alpha$	$for \ \ any \ \ atom \ \ \alpha.$
$(\to)$	$A \to B \leq A' \to B'$	$if \ \ A \leq A' \ \ and \ \ B \leq B'.$
$(\times)$	$A \times B \leq A' \times B'$	$if \ \ A \leq A' \ \ and \ \ B \leq B'.$
$(symb)$	$s^A \leq s^{A'}$	$if \ \ A \leq A'.$
$([])$	$[x : A]t \leq [x' : A']t'$	$if \ \ A \leq A' \ \ and \ \ if \ \ t[x := z] \leq t'[x' := z]$ $with \ \ z \ \ a \ \ fresh \ \ variable.$
(app)	$(t \quad u) \leq (t' \quad u')$	$if \ \ t \leq t' \ \ and \ \ u \leq u'.$
$(<>)$	$< t, u > \leq < t', u' >$	$if \ \ t \leq t' \ \ and \ \ u \leq u'.$
(π_i)	$(\pi_i \quad t) \leq (\pi_i \quad t')$	$if \ \ t \leq t'.$

We extend naturally this notion to contexts. LetΓ and Δ be two contexts, we say that $\Gamma \leq \Delta$ iff $\forall x : (x : A \in \Gamma) \Rightarrow (\exists B \geq A)(x : B \in \Delta)$.

Definition of observational equivalence

We define the observational equality in the following manner. If $.^A$ is a variable of type A, then a context with a hole $.^A$ is a closed term $C[.^A]$, where $.^A$occurs exactly once. $C[t]$ is the expression $C[.^A]$ where $.^A$ has been replaced by t. Now if t and u are two terms having the same type A in some common context we define :

$$(t =_{obs} u) \iff \forall C[.^A] : N \ \ binding \ \ every \ \ all \ \ variables \ \ in \ \ t, u \quad : C[t] =_R C[u]$$

In fact one has to consider $C[.^A]$ as an observation, or an experience on the behavior of t and u. If no experience allows to find any difference, then t and u are said observationally equivalent.

Interactions between pruning and observational equivalence

Theorem 1 (Berardi 1993) *Let t and u be two terms of same type A with a common context. Then $t \leq u \Rightarrow t =_{obs} u$.*

The proof proceeds in the following way. One begins to check the validity of this theorem for a normal (i.e. there are no more possible derivation from this term) and closed term. It is a crucial step because the system $\mathcal{T}$ has the following property :

Property 1 *The system $\mathcal{T}$ is Church–Rosser and strongly normalizing.*

So it is possible to associate, by some reduction path, to each term a term which is closed and normal.

It is easy to see, thanks to the simplicity of the system, that the theorem is valid for a closed and normal term. More exactly it has been showed that :

Lemma 1 *If $t \leq u$ and if u is closed and normal and has an atomic type, then either $t = \emptyset$ or $t = u$.*

After Berardi verifies some lemmas about commutativity between $\leq$ and substitution and reductions. More precisely, Berardi demonstrates :

Lemma 2 *If $t \leq t'$ and $t' \rightarrow_n u'$ then one can find $u \leq u'$ such that $t \rightarrow_{\leq n} u$*

Now the demonstration of the theorem is easy.

Proof:

Let's take t, u closed of type $A = N$ such that $t \leq u$. If e is the normal form of u, then there is a n such that $u \rightarrow_n e$, by the preceding lemma one can derive the existence of d such that $t \rightarrow_{\leq n} d$ with d closed and normal and $d \leq e$. From the first lemma we can deduce that $d = e$ (otherwise $d = \emptyset$ which is a trivial case) and so we have $t =_R u$. It concludes the proof in this case because it trivially implies that $t =_{obs} u$.

Now suppose that t, u are of some type A such that $t \leq u$. Let $C[.^A] : N$ be a suitable context (closing the terms etc.). As $\leq$ is compatible with term formation it is clear that $C[t] \leq C[u]$. Then one just have to apply the preceding reasoning and get $C[t] =_{obs} C[u]$. Hence the final result is reached.

□

Properties of $\leq$

Berardi has also showed some properties on the structure generated by $\leq$. The first is quite straightforward :

Property 2 *$\leq$ is an order relation on well formed terms .*

He then defines two important structures :

Definition 1 *For any term t such that, $\Gamma \vdash t : T$:*

1. $LE(t) = \{t' | t' \leq t\}$
2. $CLE(t) = \{t' | t' \leq t, \Gamma' \vdash t' : T \quad and \quad \Gamma' \leq \Gamma\}$

He shows the following proposition :

Property 3 *Let t be a term :*

1. *$LE(t)$ is a finite complete lattice with respect to $\leq$.*
2. *$CLE(t)$ is a sublattice of $LE(t)$.*

One can deduce from this property and Theorem 1 that for each term u there exists an optimal version (with respect to $\leq$) of this term, namely the greatest lower bound of $CLE(u)$.

To see what kind of optimizations are allowed by pruning we are going to develop an example.

Example 1 (C. Paulin) *Let us consider a program which takes an integer n in input, having two parameters x, y of integer type, and two auxiliary functions f, g of respective types $N \to N$ and $N \times N \to N$ and which returns an integer.*

Let a, b be the initial values of x, y. The program replaces x by $(f\ \ x)$ n times and y by $(g\ \ x\ \ y)$, the output being x.

To encode this program in the system $\mathcal{T}$ we proceed that way : we represent the pair $< x, y >$ by a single variable w of type $N \times N$, then we define :

$$u = [n : N](\pi_1\ \ (rec\ n\ \ < a, b >\ \ h))$$

where

$$h = [m : N][w : N \times N] < (f\ \ (\pi_1\ \ w)), (g\ (\pi_1\ \ w)\ \ (\pi_2\ \ w)) >$$

Then an optimized version $t \leq u$ which does not contain the function g, having the same type $(N \to N)$ is :

$$t = [n : N](\pi_1(rec\ \ n\ \ < a, \emptyset >\ \ h'))$$

where h' is :

$$h' = [m : N][w : N \times U] < (f\ \ (\pi_1\ \ w)), \emptyset >$$

If we erase every occurrences of $\emptyset$ which is the term which denotes the empty we find the simpler expression :

$$t = [n : N](rec\ \ n\ \ a\ \ ([m : N][w : N](f\ \ w)))$$

Discussion

Berardi has proposed a dynamic,completely automatic, technique, to optimize terms by deleting redundant parts of it. To realize it, he only uses the information carried by the type of the term. Everything which does not contribute to the type of the term is eliminated, that is to say replaced by $\emptyset$.

What is done, from a logical point of view, is nothing but marking ,inside a proof tree, that we have been able to prove something, but that the proof of this thing is useless. We then transform the tree in an another one into an another system where all useless terms have been deleted.

As this system is applied to the first order, Berardi can mix the two steps : marking the tree and then transformating it regarding this marking without trouble. At every moment only well formed terms are manipulated.

Under this shape, this method is hardly extensible. By studying the F_2 system, Boerio had to switch the way to see things.

2.1.3 Boerio's extension for the second order

The extension of the preceding results in [2] can be situated in the straight line of Berardi's work. It is based on deleting subtrees judged useless regarding some criterions (here observational equality). But if the starting point of the problem is the same, it is not the case for the approach of how to solve it. This approach represents a real contribution by introducing the notion of marking.

The Pruning in F_2

Boerio considers the F_2 system. If the Gödel's system $\mathcal{T}$ allows to describe the arithmetic of the first order, F_2 is a propositional calculus of the second order. More precisely, F_2 is a stable extension of $\mathcal{T}$ which means that every true statement of t is also a true statement in F_2. Moreover these judgments have been added the following elements :

1. Product types. If X is a variable of type and T a type, then $(X)T$ is a type (one can note that instead of $\forall X.T$ we will write $(X)T$ as in the Calculus of Constructions). The primitive types are the same as those in $\mathcal{T}$: integers denoted by N and U the type denoting the *hole*.

2. Abstraction on types variables. If X is a type variable and t a term than $[X]t$ is also a term.

3. The applications for types. If t is a term than $(t\ \ T)$ is also a term.

The new typing rules in this system are the following ones :

$$\frac{c : A \in \Gamma}{\Gamma \vdash c : A} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash [x : A]t : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash (f\ \ a) : B}$$

$$\frac{\Gamma \vdash a_1 : A_1 \quad \Gamma \vdash a_2 : A_2}{\Gamma \vdash < a_1, a_2 > : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash (\pi_{i,A_1,A_2}\ \ t) : A_i}$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash [X]t : (X)T} \qquad \frac{\Gamma \vdash t : (X).T}{\Gamma \vdash (t\ \ A) : T[X := A]}$$

Where Γ is an environment containing all the constants defined in the system $\mathcal{T}$. The differences between the two systems can be summed up to the two last rules added : abstraction of types on terms and application of types to terms.

We also add a β-reduction relatively to terms which are applied to types :

$$([X]t\ \ T) \to_\beta t[X := T]$$

Boerio shows in a way close to Berardi that this system also verifies the properties of the $\mathcal{T}$ system for the pruning. If an order relation is introduced with respect to the pruning, as in the $\mathcal{T}$ system, then there is an analogous of Berardi's theorem : if $t \leq u$ and both t, u have the same type then $t =_{obs'} u$, where $=_{obs'}$ is an observational equality defined as above in the $\mathcal{T}$ system.

One can also define in F_2 the lattices $LE(t), CLE(t)$ as above.

Now the goal is to find explicitly the greatest lower bound of $CLE(t)$ for a given u to get the optimal version of u.

The pruning algorithm

In a simply typed system, it is not difficult to delete subterms while keeping a well formed term, as the terms only depend on terms. With the second order that is another story : terms can depend on types (think to λ–abstraction on types). In particular, a *simple* algorithm like a data–flow analysis recursive one, was enough in a system of first order. Here it is no longer the case.

Instead of giving *in extenso* Boerio's algorithm, we will give general ideas to understand how it works. We provide an example at the end of this section. More precision can be found in Boerio's paper [2].

The ground idea is to work on the proof tree of the term and to transform it into a new optimized one.

Before going any further we introduce some important notions for the understanding of the algorithm.

In Boerio's interpretation a mark is a label given to an atomic type, a constant or a variable. One distinguishes two marks r and c. The first means that the labeled item is useless and the second that it *may* be useful for the computation. After a *correct* marking of the tree, it will be possible to replace parts marked with r by $\emptyset$ for terms and U for types.

The *simplified tree* is a syntactical tree in which every node marked with r is replaced by $\emptyset$ or U. $Simplify(M, t)$ will denote such a tree for a marking M given on a term t.

To be more precise, we give the following definitions :

Definition 2 *If t is a term such that $\Gamma \vdash t : T$ then :*

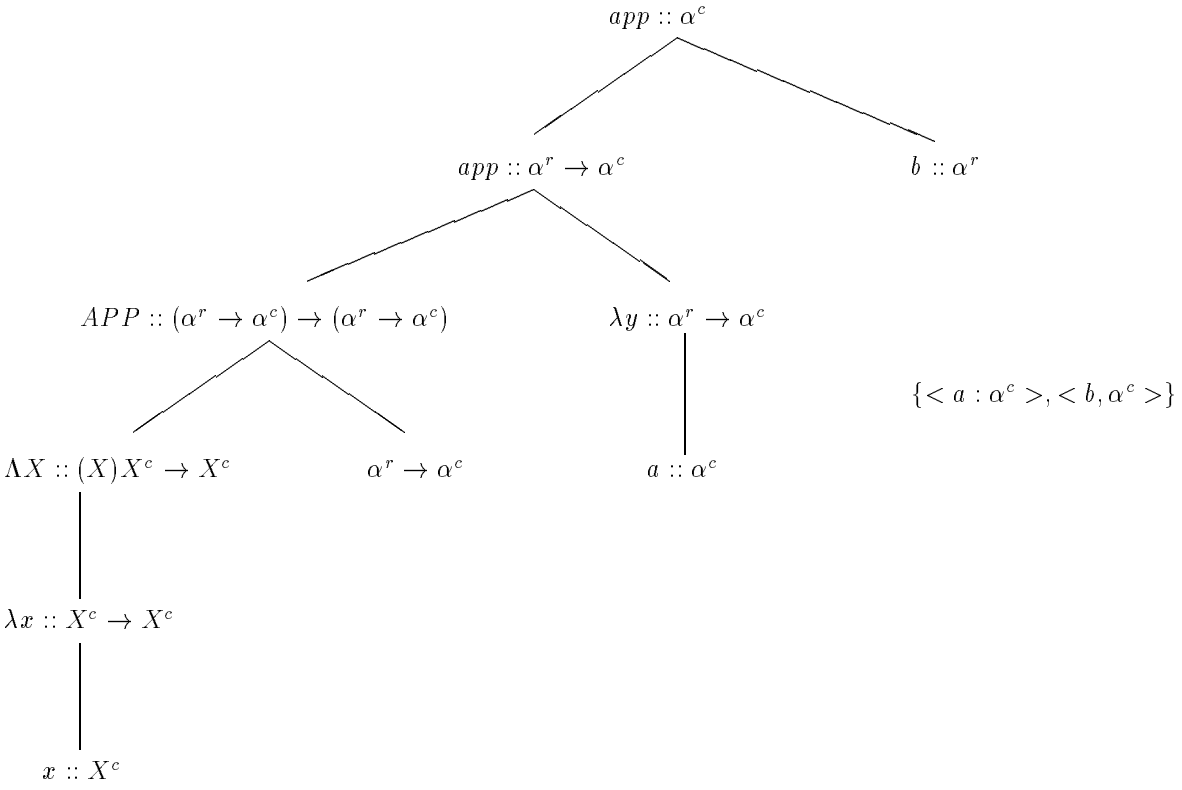


Figure 1: $FDV(t)$ with an example of marking

1. *The fully decorated tree of t, $FD(t)$, is the syntax tree of t in which each node that identifies a subterm is decorated by the type of the subterm itself.*
2. *The fully decorated version of $\Gamma \vdash t : T$, $FDV(\Gamma \vdash t : T)$ is the ordered pair $< \Gamma, FD(t) >$, formed by the context Γ and $FD(t)$.*
3. *Let Lab be the set formed by two labels r and c. A marking M of $FDV(\Gamma \vdash t : T)$, is a map from the occurrences of atoms of the types which decorates the nodes and those of atoms in the types of the variable of the context to Lab. When we talk of the marking M of t we talk about the restriction of the map to $FD(t)$.*

Example 2 (Boerio) *Let t be the following term :*

$$t \equiv ((([X : Prop][x : X]x \quad (\alpha \to \alpha))([y : \alpha]a))b)$$

.

With α an atom, a and b two variables of type α.
In the rest of this section we will call the marking, represented on Figure 1 by M_1.

Definition 3 *We call the* simplified tree, *the syntactic tree in which every node labeled with r is replaced by $\emptyset$ or U : we will note $Simplify(M, t)$ such a tree for a marking M given, on a term t.*

The algorithm builds an *optimum* marking with regards to some criterions. Not every marking makes sense regarding the pruning. Let us first give some new definitions:

Definition 4 *1. A marking M is **canonical** if for each node V of $FDT(t)$ no atom in the types associated to descendent nodes of V is marked with a c, when the type associated to V has all of its atom labeled with r.*

2. *A marking M of $FDV(\Gamma \vdash t : T)$ is* ***consistent*** *iff the simplified tree generated is the syntactic tree of a well formed term in some inferior (with respect to pruning relation) context to Γ.*

3. *A marking M is* ***saturated*** *if it is both consistent and canonical.*

4. *Let M and M' be two markings of $FDV(\Gamma \vdash t : T)$. One will state that $M \leq M'$ if for all c assigned in M it is also assigned in M'.*

In our example M_1 is saturated. If we apply the simplification map we get :

$$Simplify(M_1, t) \equiv ((([X : Prop][x : X]x(U \to \alpha))([y : U]a))\emptyset)$$

It is possible to show that each of the orders corresponds to the other. In other words, $M \leq M'$ is equivalent to $Simplify(M, t) \leq Simplify(M', t)$.

As we are not looking for the minimal marking, which is obviously the one that assigns r to every node and, hence, produces $\emptyset$ as the simplified version,, but a marking which respects the observational equality, one has to preserve the final type of the term. This observation leads to the definition of an initial marking which expresses what we know *a priori*. The definition of the initial marking M_0 is the following one : given a $FDV(\Gamma \vdash t : T)$, M_0 is the marking which associates c to every atom of the type associated to the root and the atoms in the type of the variables inside the context, and which sets all the other atoms to r.

The initial marking of a term is *canonical* but not *consistent* . The only remaining task to find the right marking is to transform this initial marking to a *consistent* one M', superior to it. In order to do that, Boerio develops an algorithm, named *saturation* algorithm, which is based on the installation of links between the atoms of the fully decorated version of the term.

The idea of the *saturation* procedure is to build edges between the different atoms of the tree to enable the c information to *flow* along those paths from the root to the totality of the tree with respect to some conditions. One can sum up those conditions by the formula "If I have some contribution to the final type I am marked with c, if not, I will keep my r mark ".

Discussion

Boerio's work is important for our approach because it has showed the existence of a correspondence between the pruning and the marking. But Boerio only uses this correspondence for technical reasons. It is just mentioned in order to realize an algorithm.

Actually the notion of *link* introduced is very interesting. Indeed it shows how the marking can be gradually built. But here too, one can not find explanations of this notion : he only uses it in a casual way to build an optimal marking.

By a deeper abstraction of those two notions one could understand the semantique of the pruning. It would permit to see why *links* are defined this way and why other options do not fit the problem.

2.1.4 Using subtyping for pruning [3]

In their precedent works, Berardi and Boerio have developed pruning techniques. One of the weaknesses of these ones leads to the fact that some optimizations are not done, because they could lead to ill typed function applications.

To overcome those difficulties, they have developed a theory in which they introduce the notion of *subtype*. Now a function application will be well typed if the type of the argument is not only equal but also included in the input type of the corresponding function. This way more optimizations will be done : the extension is conservative.

Brief presentation

This theory have been developed on a first order language which is an extension of the $\mathcal{T}$ language used by Berardi. We will note $\mathcal{T}_\Omega$ the extension of $\mathcal{T}$ obtained by adding a special atomic Type Ω.

Definition 5 *1. The Types of $\mathcal{T}_\Omega$ are defined inductively from the atomic types N and Ω using the $\to$ and $\times$ operator.*

2. Let A, A_1, A_2, B, B_1, B_2 and C be types. The relation of inclusion on types of $\mathcal{T}_\Omega$ is defined by the following rules :

$$N \subseteq_\Omega \Omega$$

$$N \subseteq_\Omega N \qquad \Omega \subseteq_\Omega \Omega$$

$$\frac{B_1 \subseteq_\Omega A_1 \quad B_1 \subseteq_\Omega A_1}{A_1 \to A_2 \subseteq_\Omega B_1 \to B_2} \qquad \frac{B_1 \subseteq_\Omega A_1 \quad B_1 \subseteq_\Omega A_1}{A_1 \times A_2 \subseteq_\Omega B_1 \times B_2}$$

3. The notion of Ω–type is also defined. A type T will be said an Ω–type if it can be produced by the following grammar :

$$O ::= \Omega | A \to O | O \times O$$

where A is any type.

The preterms are the same ones as in the $\mathcal{T}$ system. The set of constants is extended by a set of constants d_O for each Ω–type O. They denote a canonical element for a given Ω–type.

The new typing rules are the following :

$$\frac{c : A \in \Gamma}{\Gamma \vdash c : A} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash [x : A].t : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A' \quad A' \subseteq_\Omega A}{\Gamma \vdash (f \quad a) : B}$$

$$\frac{\Gamma \vdash a_1 : A_1 \quad \Gamma \vdash a_2 : A_2}{\Gamma \vdash < a_1, a_2 > : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash (\pi_{i, A_1, A_2} \quad t) : A_i}$$

The definitions of observational equality and context with hole are the same as in the $\mathcal{T}$ system. Moreover, this system also verifies the same properties as the $\mathcal{T}$ system, it is Church–Rosser and strongly normalizing.

Semantics of the new system

Let us discuss about the role of Ω and of the inclusion $A' \subseteq_\Omega A$. One can consider that N is the set of integers with standard equality, whereas Ω is the type of integers which are equivalent to each other. It represents the *useless* integers. Indeed if a term is of type Ω, one knows that it is a number but nothing more. The Ω type *carries* just this information whereas N brings more informations allowing to positively identify the number. From the last point of view, the membership of Ω is weaker than the one of N, hence the preceding inclusion.

One can now see clearly what means to *simplify* a term in this system. A set of subtypes and subterms in a given term can be useless for the final computation and then replaced with Ω (useless type) and the subterm with d_O (useless subterm), while keeping the *well formation of the term.*

Results obtained

The authors develop an order relation on the words of $\mathcal{T}_\Omega$ in a similar way as the one defined in $\mathcal{T}$. The only difference comes from the following rule :

$$\frac{O \leq U}{d_O \leq u}$$

for u of any type U.

As in the precedent systems, they show that if a term is inferior to another with the same type, is observationally equivalent to the previous one. Moreover, the structure generated by this relation is a complete lattice, and the subsets formed by terms of a common type are sublattices.

So we have the same results for this system as for the two previous ones regarding the pruning.

Example 3 (Boerio-Berardi) *Now we are going to give an example which shows that this new method is more powerful than the previous ones.*

Let suppose the following extension of the constant set : it is added the constant set $it_A = \{N \to A \to (A \to A) \to A | A anytype\}$ *to the original constant set* Σ_Ω. *These constants implement iterations on natural numbers. We also introduce new reduction rules which are* $(it_A\ 0\ a\ f) \to_{it} a$ *and* $(it_a\ (succ\ n)\ a\ f) \to_{it} (f\ (it_A\ n\ a\ f))$ *where* a *is of type* A *and* f *of type* $A \to A$.

Consider the expression :

$$t \equiv [n : N][v : N \times N](\pi_1\ it_{N\times N}\ n\ v\ F)$$

with

$$F \equiv [w : N \times N] < (f\ (\pi_1\ w)), (g\ (\pi_2\ w)) >$$

where f *and* g *are free variables of type* $N \to N$.

t *is of Type* $N \to N \times N \to N$. *With the previous versions of the pruning one can not simplify* t. *Now using the subtyping we get the simplified* t' *term :*

$$t' \equiv [n : N][v : N \times \Omega](\pi_1\ (it_{N\times\Omega}\ n\ v\ F'))$$

with

$$F' \equiv [w : N \times \Omega] < (f\ (\pi_1\ w)), (g\ d_\Omega) >$$

2.1.5 Some comments on Turin school

The pruning presents without any doubt many advantages. The first is that it is an optimal technique regarding this kind of optimization where it is only seeked to erase useless parts of a term. The second one is its *automatic* aspect : the user has nothing to do but to watch the machine solving the problem.

Nevertheless, the fact that those techniques are only valid for first and second order is not satisfactory. One would like to use these techniques in a more realistic frame. It would be fine if such optimizations were possible in systems like Coq or Nuprl.

Another bad point is clearly lightened by the evolutions of pruning. One has certainly notice that to extend earlier results it has been each time necessary to build new theories. The lack of a uniform approach suggests to search what are the common points of the different pruning techniques.

2.2 C. Paulin's approach [9]

C. Paulin has developed another approach to the extraction in a the Calculus of Constructions.

2.2.1 The Calculus of Constructions

For a more complete presentation refer to [9].

Actually, we will consider a slightly different version compared to the original model. It is called the Calculus of Constructions with Realizations.

Syntax of the language

We will denote by Λ the smallest language containing the following constructions:

- Three constants: $Prop$, Set, $Type$.
- An enumerable set of variables V.
- Applications: $(M\ \ N)$ with $M, N \in \Lambda$
- Products: $((x : M)N)$ with $x \in V$ and $M, N \in \Lambda$
- Abstractions: $([x : M]N)$ with $x \in V$ and $M, N \in \Lambda$

An environment is a list of bindings, $x_1 : A_1, \ldots, x_n : A_n$ where x_i is a variable and $A_i \in \Lambda$. We note the empty list by $[]$.

One can emit two sort of statements in this language:

1. Γ is valid where Γ an environment.
2. $\Gamma \vdash M \in N$ which is read: M is well formed of type N in the environment Γ. If M and N are terms of Λ.

Inference rules

We will denote by $\mathcal{K}$ one of the three constants $Prop, Set, Type$.

$$[]\ \ is\ \ valid$$

$$\frac{\Gamma \vdash M \in \mathcal{K} \quad x \in \Lambda\ \ does\ \ appear\ \ in\ \ \Gamma \quad \Gamma\ \ is\ \ valid}{\Gamma, x : M\ \ \ is\ \ \ valid}$$

$$\frac{\Gamma\ \ is\ \ valid \quad x : N\ \ appears\ \ in\ \ \Gamma \quad N \in \Lambda}{\Gamma \vdash x \in N}$$

$$\frac{\Gamma\ \ is\ \ valid}{\Gamma \vdash Prop \in Type} \qquad \frac{\Gamma\ \ is\ \ valid}{\Gamma \vdash Set \in Type}$$

$$\frac{\Gamma, x : P \vdash M \in \mathcal{K}}{\Gamma \vdash ((x : P)M) \in \mathcal{K}}$$

$$\frac{\Gamma, x : P \vdash M \in N \quad N \neq Type}{\Gamma \vdash ([x : P]M) \in ((x : P)N)}$$

$$\frac{\Gamma \vdash M \in ((x : P)N) \quad \Gamma \vdash R \in Q \quad P =_{\beta\eta} Q}{\Gamma \vdash (M\ \ R) \in (N[x/R])}$$

$$\frac{\Gamma \vdash M \in N \quad \Gamma \vdash P \in \mathcal{K}\ \ N =_{\beta\eta} P}{\Gamma \vdash M \in P}$$

In spite of a uniform presentation which could let think that there are no distinction between terms, on can derive some classes of terms. They are identified with the distance between them and the constant $Type$. More precisely, we have :

Property 4 *Let M be a well formed term of Λ of type N, there are three different cases :*

1. $N = Type$
2. *N is well formed of type $Type$.*

3. N is well formed of type $Prop$, or Set.

This property leads to the definition of *levels* in the Calculus of Constructions.

Definition 6

- *If $N = Type$ then M is of level 2, we talk about propositional types.*
- *If N is well formed of type $Type$, M is of level 1, it is a propositional scheme, or, more simply a proposition, if $N = Prop$.*
- *If N is well formed of type $Prop$ or Set then M is of level 0 and we say that M is a proof.*

This hierarchical structure is important to prove properties and give well formed inductive definitions.

2.2.2 Extraction in the Calculus of Constructions

To determine what parts of proof's terms are useless on a computational point of view, C. Paulin duplicates the original Calculus of Constructions thanks to the two constants $Prop$ and Set. The constant $Prop$ will be used for the building of logical propositions (which will not be used in the final program), whereas the Set constant will denote terms that have an algorithmical content.The notion of "having an interesting content") is introduced in the following manner.

A term will be said of empty content if :

- This term is $Prop$.
- If M is a term with an empty content then $(x : N)M$ is also of empty content.
- If M is a term with an empty content then $[x : N]M$ is also of empty content.
- If M is a term with an empty content then $(M \quad N)$ is also of empty content.

A variable that has a type of empty content is also of empty content.
If a term is not of empty content, it will be called positive.
The extraction technique consist in erasing all parts of the term that have an empty content.

2.2.3 Discussion

The main drawback of this technique is its lack of flexibility. Indeed, when a term is declared positive it can't in no way change its status and becomes a term of empty content. This is regrettable because the notion of 'algorithmic content of a term appears more as a contextual notion than syntactic. Consider map g which takes an integer and returns this integer multiplied by 2. In absolute terms, this map has an algorithmic content. Now, one can easily design a program that takes a map from integers to integers as argument and which gives back the first projection of the pair formed by n iterations of the g map, and m of the same map. The term representing this program will have two subterms coding the g map. Nevertheless, from, the observational point of view, only the code of the map which computes the first projection will have an informative content. That way we have the case of a function which in absolute has an algorithmic content but which can in some particular case have no longer interest because of the context which surrounds it. Consider again the first example. With some slight modifications it appears that the Paulin's extraction technique one can not obtain the results of pruning's extraction. Just replace the occurrence of the function f by the function g, the second occurrence of g is erased by the pruning whereas the first is kept. As it is impossible to type g with both $Prop$ and Set, Paulin's extraction can't erased it.

Another remark about this method comes from the fact that it is the user which determines the extraction procedure. It is him who chooses his variables being of type Set or $Prop$. Now it would be fine if the system was able to extract alone subterms without informative content.

The advantage of this approach is that it is valid for languages of higher order with dependent types as the Calculus of Constructions.

3 A new approach of extraction by marking

3.1 Motivations and brief description

As the former approaches have been shown to be not satisfactory, we have to find another approach to extend the results obtained. The idea of marking comes naturally : we will indicate by a new attribute whether the term is useful for the computation or not. This idea was already present in Boerio's algorithm (see [2]).

As one can already deduce from earlier studies, the marking should verify some requirements. It has to be orthogonal to the typing mechanism, as Paulin's method reveals that typing and informative content are two different things. It has also to allow to rely on the contextual feature of the informativity of a term. Indeed, we have seen that the typing is too static to express the extraction's optimization features. This last remark suggests the creation of *weakening* rules, which transform a term with positive content into a new one with negative content.

Moreover, one has to has clarify the part of the constraints introduced to mark the term by expressing them inside the typing rules.

3.2 Application to a second order system F_2^m

In the rest of the paper, we will denote by Λ_m the smallest language formed by the following rules.

We define the sets :

$$Label = \{r, c\}$$

$$const = \{Prop, Type\}$$

Let $\mathcal{V}$ be a set of variables. In the rest of the paper, the indexes $i, j, k, l \ldots$ stand indistinctly for r or c and we write N^i for (N, i). Moreover, we define the order relation $r \leq c$ on the set $Label$. The language Λ_m contains :

- $Const = const \times Label$
- $V = \mathcal{V} \times Label$
- Applications:
$$(M^i \quad N^j)^k with \quad M^i, N^j \in \Lambda_m$$
- Abstractions:
$$([x^i : M^j]N^k)^l \quad with \quad x^i \in V \quad and \quad M^j, N^k \in \Lambda_m$$
- Products:
$$((x^i : M^j)N^k)^l \quad with \quad x^i \in V \quad and \quad M^j, N^k \in \Lambda_m$$

We will write $x :^i$ for $x^i : A^i$.

An environment is a finite list of bindings $x_1 :^{i_1} A_1, x_2 :^{i_2} A_2, \ldots, x_n :^{i_n} A_n$, where x_j is a variable. $A_j \in \Lambda_m$ and $i_j$is an element of $Label$.

We next define the well formation rules of term in Λ_m. These rules allow us to define a second order language, so polymorphic, that is a language in which one can abstract over types and . We will call this language F_2^m.

Judgments are of two different kinds. If Γ is an environment, we have: Γ is valid and $\Gamma \vdash M \in N$, where Γ is an environment, M and N are two terms of Λ_m and this is to be read " the term M is well formed of type N in the environment Γ".

Inference Rules :

We will take the convention that for every marking variable $:i, j, k, \ldots$ we will have the order relation: $i \geq i' \geq i'' \geq i''' \ldots$

Moreover, to have a more readable paper we will use the following syntactic sugars.

- $\forall X^i$ shall be read as $(X :^i Prop)$.
- ΛX^i shall be read as $[X :^i Prop]$.
- $mark(occ(x, A))$ will design the set of mark of the occurrences of the variable x in A.
  $sup(mark(occ(x, A)))$ will be the "usual" operator, except when the set $mark(occ(x, A))$ is empty. In this case the value is not defined. Of course r is the good choice but, for greater generality, we will let the choice to the user(see theorem 5).

Here are the inference rules of this second order language:

- Environments:

$$(1)\ \frac{}{[]\ \ is\ \ valid}$$

$$(2)\frac{\Gamma \vdash B^i \in Prop^i \quad x\ \ doesn't\ \ appear\ \ in\ \ \Gamma}{\Gamma, x :^{i'} B\ \ \ is\ \ \ valid}$$

$$(3)\frac{\Gamma\ \ is\ \ \ valid \quad X\ \ does\ \ not\ \ appear\ \ in\ \ \Gamma}{\Gamma, X :^i Prop\ \ is\ \ \ valid}$$

- Hypotheses:

$$(4)\frac{\Gamma\ \ is\ \ \ valid \quad x :^i B\ \ appears\ \ in\ \ \Gamma}{\Gamma \vdash x^{i'} \in B^{i'}}$$

$$(5)\frac{\Gamma\ \ is\ \ \ valid \quad X :^i Prop\ \ appears\ \ in\ \ \Gamma}{\Gamma \vdash X^{i'} \in Prop^{i'}}$$

- Type formation:

$$(6)\frac{\Gamma \vdash A^i \in Prop^i \quad \Gamma \vdash B^j \in Prop^j}{\Gamma \vdash (A^i \to B^j)^j \in Prop^j}$$

$$(7)\frac{\Gamma, X :^i Prop \vdash A^j \in Prop^j \quad i' = sup(mark(occ(X, A)))}{\Gamma \vdash (\forall X^{i'}.\ A^j)^j \in Prop^j}$$

- Abstractions:

$$(8)\frac{\Gamma, x :^i A \vdash M^j \in N^j \quad \Gamma \vdash N^j \in Prop^j \quad i' = sup(mark(occ(x, M)))}{\Gamma \vdash ([x :^{i'} A]M^j)^j \in (A^{i'} \to N^j)^j}$$

$$(9)\frac{\Gamma, X :^i Prop \vdash t^j \in T^j \quad \Gamma \vdash T^j \in Prop^j \quad i' = sup(mark(occ(X, t)))}{\Gamma \vdash (\Lambda X^{i'} t^j)^j\ \ \in\ \ (\forall X^{i'}.\ T^j)^j}$$

- Applications.

$$(10)\frac{\Gamma \vdash t_1^j \in (A^i \to B^j)^j \quad \Gamma \vdash t_2^i \in A^i}{\Gamma \vdash (t_1^j\ t_2^i)^j \in B^j}$$

$$(11)\frac{\Gamma \vdash t^j \in (\forall X^i.\ B^j)^j \quad \Gamma \vdash T^i \in Prop^i}{\Gamma \vdash (t^j\ T^i)^j \in (B^j[X^i := T^i])^j}$$

One can already makes some remarks.

- Marking and typing are not directly linked. For a given term, one can give several different markings. Meanwhile the marking respects some rules described in typing rules.
- There are two different kinds of marks. The ones used to label variables only depend on the context, whereas the ones used to label a term are *inherited* from the subterms it contains.
- One can only weaken a mark at the introduction of variable level (rules (4) and (5)). This is clearly needed if we consider the previous remark.
- The links built in Boerio's algorithm have been translated within the typing rules.

To exploit this system we have to define a new β–reduction :

Definition 7 (β–reduction in a marked language) *there are two cases :*

1. *If the argument is marked with r, we have a redex which looks like* $(([x :^r A]M^i)^i N^r)^i$, *and the inference rules on well formation for terms impose that every occurrence of* x *in* M *is marked* r. *We define :*

$$(([x :^r A]M^i)^i \ N^r)^i \rightarrow_\beta M^i[c^r := N^r]$$

2. *If the argument is marked with c, we only know that there is at least one occurrence of the binded variable marked with c while the others can be marked with r. We have a problem to instantiate both kind of variables. But we know that the weakening rules enable us to consider every term marked with c as the same term but marked r (one just have to mark with r every introduction of a variable during the building of the term). It leads to the following definition :*

$$(([x :^c A]M^i)^i N^c)^i \rightarrow_\beta (M^i[x^c := N^c][x^r := \overline{N}^r])^i$$

where $\overline{N}^r$ *represent the term* N *in which every atom has been marked with* r.

This definition looks reasonable because the following property holds:

Property 5 (Invariance of marking under β–reduction) *Let* M^i *be a well formed term of* F_2^m, *we have the following property.*

$$M^i \rightarrow_\beta M'^i$$

Proof:

Let consider a term M^i. We show, by structural induction that substituting any subterm of M^i it by another one with the same mark does not change the mark of M.

This is trivially verified for variables. Indeed the only subterm is the variable itself and so it will keep the same mark.

Suppose that $M \equiv (t_1^i \ \ t_2^j)^j$. Then, if we substitute the subterm t_1^i by q^i the term will keep the same mark. It is also the case if we substitute q^j to t_2^j.

The other cases are treated in the same way.

□

3.2.1 Order introduced on terms by marking

We now define an order on well formed terms of Λ_m . This order is generated by the order relation previously defined on the set *Label* set. We say that if two terms have the same syntactical tree (without looking at the marks), the one which has the longest number of 'c' is the bigger.

This leads to the following definition (by structural induction):

- Ground cases.

- For constants: $Prop^r \leq Prop^c$
- For variables: $x^r \leq x^c$

- Induction cases on types.
  - If M, M', N' and N are types : $M' \to N' \leq M \to N$if $M' \leq M$ and $N' \leq N$.
  - If X, X' are type variables and M M' are types then $\forall X'M' \leq \forall XM$ if $X' \leq X$ and if $M' \leq M$.

- Induction cases on terms.
  - If t_1, t'_1, t_2 et t'_2 are terms then $(t'_1 \ \ t'_2) \leq (t_1 \ \ t_2)$ if $t'_1 \leq t_1$ and $t'_2 \leq t_2$.
  - If t, t' are terms, T and T' types then $(t'[T']) \leq (t[T])$ if $t' \leq t$ and $T' \leq T$.
  - If M, M' are terms, A, A' types, then $[x : A']M' \leq [y : A]M$ if $A' \leq A$ and if $M'[x := z] \leq M[y := z]$ where z is a fresh term variable (this is to avoid problems linked to α–conversion).
  - If M and M' are terms then $\Lambda XM' \leq \Lambda YM$ if $M'[X := Z] \leq M[Y := Z]$ where Z is a fresh terms variable (it is to avoid problems linked to α–conversion)

3.2.2 Flexibility of F_2^m

We now show that our system is flexible by simulating the other extraction procedures. These different simulations are naturally obtained by playing on two parameters. The first parameter concerns the initial conditions, it corresponds to the initial marking of Boerio's algorithm. The second parameter deals with the way the extraction is defined. In Boerio, for example, one can remove a subtree only if *all* atoms of this subtree are marked with r while it is not the case, for Paulin.

• Simulation of Paulin's Extraction

Definition 8 *The extracting function Ex_P is inductively defined by :*

- $Ex_P(x^c) = \overline{x}$
- $Ex_P(X^c) = \overline{X}$
- $Ex_P((A^r \to B^j)^j) = Ex_P(B^j)$
- $Ex_P((A^c \to B^j)^j) = Ex_P(A^c) \to Ex_P(B^j)$
- $Ex_P((\forall\ X^r.A^j)^j) = \forall\ \overline{X}.Ex_P(A^j)$
- $Ex_P((\forall\ X^c.A^j)^j) = Ex_P(A^j)$
- $Ex_P([x :^r A]M^j)^j) = Ex_P(M^j)$
- $Ex_P([x :^c A]M^j)^j) = [\overline{x} : Ex_P(A)]Ex_P(M^j)$
- $Ex_P((t^j \ \ T^r)^j) = Ex_P(t^j)$
- $Ex_P((t^j \ \ T^c)^j) = (Ex_P(t^j) \ \ Ex_P(T^c)$
- $Ex_P((t^j \ \ u^r)^j) = Ex_P(t^j)$
- $Ex_P((t^j \ \ u^c)^j) = (Ex_P(t^j) \ \ Ex_P(u^c))$

Where $\overline{x}$ and $\overline{X}$ are fresh term and type variables. One can easily extend this definition to environments.

Theorem 2 *If $\Gamma \vdash M^c \in N^c$ is provable in F_2^m then $Ex_P(\Gamma) \vdash Ex_P(N^c) \in Ex_P$ in F^2.*

Proof:

To prove this theorem we look what is going on by induction on the term structure.

The ground case, the case of variables, is clearly verified. Consider the case of a variable marked by c. This the only case to be considered as extraction only works on c marked terms. We have $Ex_P(v^c) = \overline{v}$ which is a well formed term in F_2. Indeed, as the variable is marked c, its declaration in the context is of the kind $v :^c V$, and so in the extracted context appears $\overline{v} : V$, hence the announced result.

For induction cases we have the following options :

- Suppose: $\Gamma \vdash ([x :^i A]M^j)^j \in (A^i \to B^j)^j$. Two cases are to be considered:

  1. $i = r$. In this case we have $Ex_P(([x :^i A]M^j)^j) = Ex_P(M^j)$ and $Ex_P((A^i \to B^j)^j) = Ex_P(B^j)$. So it is sufficient to show that $Ex_P(\Gamma) \vdash Ex_P(M^j) \in Ex_P(B^j)$, which is exactly the induction hypothesis.
  2. $i = c$. Then we have $Ex_P(([x :^i A]M^j)^j) = [\overline{x} : Ex_P(A^i)]Ex_P(M^j)$ and $Ex_P((A^i \to B^j)^j) = Ex_P(A^i) \to Ex_P(B^j)$. One has to show that : $Ex_P(\Gamma) \vdash [\overline{x} : Ex_P(A^i)]Ex_P(M^j) \in Ex_P(A^i) \to Ex_P(B^j)$. One uses the induction hypothesis which says that : $Ex_P(\Gamma, x :^c A) \vdash Ex_P(M^j) \in Ex_P(B^j)$. In another terms $Ex_P(\Gamma), x : Ex_P(A^c) \vdash Ex_P(M^j) \in Ex_P(B^j)$. This last remark shows that one can apply the abstraction rule in F_2, hence the result.

- Now if we have : $\Gamma \vdash (t^i \ \ u^j)^i \in T^i$, with $\Gamma \vdash t^i \in (U^j \to T^i)^i$ and $\Gamma \vdash u^j \in U^j$. We have the two following subcases to treat:

  1. $j = r$. We have $Ex_P((t^i \ \ u^j)^i) = Ex_P(t^i)$ and one has to show that $Ex_P(\Gamma) \vdash Ex_P(t^i) \in Ex_P(T^i)$. But by induction hypothesis, one has : $Ex_P(\Gamma) \vdash Ex_P(t^i) \in Ex_P(U^j \to T^i)$ and as $Ex_P(U^j \to T^i) = Ex_P(T^i)$ one has the researched result.
  2. $j = c$. Now, one has $Ex_P((t^i \ \ u^j)^i) = (Ex_P(t^i)Ex_P(u^j))$ and one wants to show that $Ex_P(\Gamma) \vdash (Ex_P(t^i) \ \ Ex_P(u^j)) \in Ex_P((U^j \to T^i)^i)$. we know by induction hypothesis that $Ex_P(\Gamma) \vdash Ex_P(t^i) \in Ex_P(U^j \to T^i)$ and also that : $Ex_P(\Gamma) \vdash Ex_P(u^j) \in Ex_P(U^j)$. Now one has $Ex_p(U^j \to T^i)^i) = Ex_p(U^j) \to Ex_P(T^i)$. Hence by applying the application rule of F_2, it comes $Ex_P(\Gamma) \vdash (Ex_P(t^i) \ \ Ex_P(u^j)) \in Ex_p(U^j) \to Ex_P(T^i)$ in other words $Ex_P(\Gamma) \vdash (Ex_P(t^i) \ \ Ex_P(u^j)) \in Ex_P((U^j \to T^i)^i)$, which demonstrates the result.

All other cases, abstraction of types, and application of types to terms are treated in the same way.

□

Theorem 3 *One can simulate in F_2^m the extraction procedure proposed by C. Paulin for the Calculus of Construction restricted to the second order.*

Proof:

To the type $Prop$ of the Calculus of Constructions one associates $Prop^r$ and to Set one associates $Prop^c$.

Now, one only has to show that every term with empty content in Paulin's system is marked with an r in our system. It is simply done by induction on formation of empty content terms.

- Ground case. If $M = Prop$ in Paulin's system then by definition it will be translated in $Prop^r$ in our system.

- If M is a term of empty content in Paulin's system and translated to a term with a mark c in our system then:

  - $((x : N)M)$ has an empty content, and is translated by $((x :^i N)M^r)^r$ and so marked with r.
  - $([x : N]M)$ has an empty content, and is translated by $([x :^i N]M^r)^r$ and so marked with r.

– $(M \quad N)$ has an empty content and is translated by $(M^r \quad N^i)^r$.

Hence, every term with an empty content can be translated in our system. It shows that one can simulate Paulin's extraction in F_2^m.

□

• **Simulation of Berardi–Boerio's Extraction**

The first thing to do is to define a new extraction function. Indeed, for Boerio, a term can only be replaced by $\emptyset$ if it has *all* of its atoms marked with an r. This leads to the following definition.

Definition 9 *The extracting function Ex_B is inductively defined by :*

- $Ex_B(x^c) = \overline{x}$
- $Ex_B(X^c) = \overline{X}$
- $Ex_B((A^r \to B^j)^j) =$
  - *If every atom of A^r is marked r then $Ex_B(B^j)$*
  - *Else $(Ex_B(A^r) \quad Ex_B(B^j))$*
- $Ex_B((A^c \to B^j)^j) = Ex_B(A^c) \to Ex_B(B^j)$
- $Ex_B((\forall\ X^r.A^j)^j) = Ex_B(A^j)$
- $Ex_B((\forall\ X^c.A^j)^j) = \forall \overline{X} Ex_B(A^j)$
- $Ex_B([x :^r A]M^j)^j) =$
  - *If every atom of A^r is marked r then $\forall\ \overline{X}.Ex_B(M^j)$*
  - *Else $[\overline{x} : Ex_B(A)]Ex_B(M^j)$*
- $Ex_B([x :^c A]M^j)^j) = [\overline{x} : Ex_B(A)]Ex_B(M^j)$
- $Ex_B((t^j \quad T^r)^j) =$
  - *If every atom of T^r is marked r then $Ex_B(t^j)$*
  - *Else $(Ex_B(t^j) \quad Ex_B(T^r)$*
- $Ex_B((t^j \quad T^c)^j) = (Ex_B(t^j) \quad Ex_B(T^c)$
- $Ex_B((t^j \quad u^r)^j) =$
  - *If every atom of u^r is marked r then $Ex_B(t^j)$*
  - *Else $(Ex_B(t^j) \quad Ex_B(u^r)$*
- $Ex_B((t^j \quad u^c)^j) = (Ex_B(t^j) \quad Ex_B(u^c))$

Here too, one extends easily this notion for environments.

Theorem 4 *If $\Gamma \vdash M^c \in N^c$ is provable in F_2^m then $Ex_P(\Gamma) \vdash Ex_P(N^c) \in Ex_P$ in F^2.*

Proof:

The proof is almost the same as the one used to prove the theorem 2. One just has to check that terms are uniformly marked with r or not. This condition does not influences the well formation of terms. Moreover, every term extracted with Boerio's extraction is also extracted in the same way by Paulin's extraction, hence the result.

□

Actually erasing subterms labeled by an r is like identifying terms of the form $(t\ \emptyset)$ to t, $U \to B$ to B, $B \to U$ to U in Boerio's system.

This shows well that to the function $Simplify$ we give a corresponding function which is Ex_B.

We are now going to exploit this correspondence to prove the following theorem.

Theorem 5 *Let M be a saturated marking, in Boerio's sense, of a term t. One can derive $\Gamma \vdash t^j \in T^j$ in F_2^m in such a way that t^j has the same marks on its atoms than the one associated by M to t in F_2.*

Proof:

First let us recall the definition of a saturated marking. This is a marking which is both canonical (a subterm marked by an r has all of its atoms marked by r) and consistent (the extraction of subterms marked by r is done and the extracted term is well formed).

The theorem 4 already proves us that a marking in F_2^m is always consistent.

It is easy to see that one can always build a canonical marking in F_2^m from a given marking. Indeed, since a subtree is marked with an r one just has to instantiate the following rule of F_2^m

$$(5)\frac{\Gamma \ \ is \ \ valid \quad X :^i Prop \ \ appears \ \ in \ \ \Gamma}{\Gamma \vdash X^{i'} \in Prop^{i'}}$$

to build the subtree. This rule can be read as :

$$(5)\frac{\Gamma \ \ is \ \ valid \quad X :^i Prop \ \ appears \ \ in \ \ \Gamma}{\Gamma \vdash X^{r} \in Prop^{r}}$$

hence every of the subtree will be marked with an r.

So we can forget about the problem of canonicity.

Next one has to show that any consistent marking in F_2 can be realized in our marking system.

Let us prove it by absurdity. Suppose that we have a term t^c, marked in such a way that we cannot build its marking in our system although $Ex_B(t^c)$ is well formed with respect to F_2, or that we can find a lower marking.

So, there exists at least a node in the proof tree which does not satisfy the constraints generated by F_2^m. Let us consider the different possibilities. The node can be:

- A term application formation node. In our system we have:

$$(10)\frac{\Gamma \vdash t_1^j \in (A^i \to B^j)^j \quad \Gamma \vdash t_2^i \in A^i}{\Gamma \vdash (t_1^j\ t_2^i)^j \in B^j}$$

  As the marking has to be impossible to build in our system, the node must have the following shape:

$$\frac{\Gamma \vdash t_1^j \in (A^i \to B^j)^j \quad \Gamma \vdash t_2^k \in A^k}{\Gamma \vdash (t_1^j\ t_2^k)^j \in B^j}$$

  with $k \neq i$. Now, let us extract this subterm. By our definition of extraction we have:

  - $Ex_B((t_1^j\ t_2^k)^j)) = Ex_B(t_1^j)$ if $k = r$, and $(Ex_B(t_1^j)\ \ Ex_B(t_1^j))$ otherwise.
  - $Ex_B((A^i \to B^j)^j) = Ex_B(A^i) \to Ex_B(B^j)$ if $i = r$, and Ex_{B^j} otherwise.

  Now one can easily check that, whether $k = r$ or c, the extracted term will not be well formed.

- A type application formation node. In our system we have:

$$(11)\frac{\Gamma \vdash t^j \in (\forall X^i.\ B^j)^j \qquad \Gamma \vdash T^i \in Prop^i}{\Gamma \vdash (t^j\ T^i)^j \in (B^j[X^i := T^i])^j}$$

  if we don't use those constraints, one can show as above that the extracted term will not be well formed.

- An abstraction over terms node. In our system we have:

$$(8)\frac{\Gamma, x :^i A \vdash M^j \in N^j \qquad \Gamma \vdash N^j \in Prop^j \qquad i' = sup(mark(occ(x, M)))}{\Gamma \vdash ([x :^{i'} A]M^j)^j \in (A^{i'} \to N^j)^j}$$

  Now, to find a marking impossible to build in our system the node must looks like :

$$\frac{\Gamma, x :^i A \vdash M^j \in N^j \qquad \Gamma \vdash N^j \in Prop^j}{\Gamma \vdash ([x :^{i'} A]M^j)^j \in (A^{i'} \to N^j)^j}$$

  such that $i' \leq sup(mark(occ(X, N)))$. As we have a two element domain this leads to the unique solution $i' = r$.

  Indeed, if $i' = c$ while there is no occurrence of x in M. So $mark(occ(x, M)) = \emptyset$ in F_2^m and we can choose $i' = c = sup(\emptyset)$, obtaining the same marking.

  So, let us consider the case where $i' = r$ whereas there exists an occurrence of x in M^j which is marked c. By extraction, we will erase the binder $[x :^i A]$ but not the occurrence of x in M, hence M will not be well formed after extraction as it will contain a variable x which has no binder and no definition in the context.

- Abstraction over types. The proof is the same as above.

Hence by absurdity we have proved the result.

□

This theorem is crucial for understanding of what marking is. It expresses thatin order to have a coherent marking i.e. *saturated*, the marks have to verify certain constraints, precisely those constraints that are encoded in the typing rules of F_2^m.

Actually, the last theorem proves than in our system, we can *at worse* obtain the same results than in the F_2 with pruning. Indeed, if we restrict F_2^m to the terms having a saturated marking we will exactly find the F_2 system studied by Boerio. In particular we can prove for this restriction of F_2^m all what was proved in F_2 by Boerio.

To optimize a term t of type T, it suffices to find the greatest lower bound of $CLE_m(t) = \{t' | t' \leq t and \Gamma' \vdash t' \in T \quad and \quad \Gamma' \leq \Gamma\}$ in the restriction of F_2^m to saturated terms. For doing that, we follow the method proposed by Boerio in [3]. The initial marking consists to set all marks to a fresh variable of mark, except the ones of the global term and of the environments (terms of the environments are uniformly marked by the same mark everywhere) which are marked with c. After that one just has to follow the typing rules and generate constraints between the mark variables in the tree.

Let us consider a small example. Suppose we have:

$$\frac{\Gamma \vdash t_1^i \in (A^j \to B^k)^l \qquad \Gamma \vdash t_2^m \in A^n}{\Gamma \vdash (t_1^o\ t_2^p)^q \in B^h}$$

Then if we follow the associated typing rules, we can find that $i = l = o = q = h$ and that $j = m = n = p$. This way one can make the final c mark flows from the root to the rest of the term.

Let us next develop a larger example to see how one can simulate Boerio's algorithm.

Example 4 *Consider the following term* $t \equiv (([x:X][y:X]y\ \ u)\ \ v)$. *We will show how to simplify it using the* F_2^m *system. Suppose* $\Gamma \equiv X :^c Prop, u :^c X, v :^c X$. *We begin by giving its derivation tree in* F_2*:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; y:X \vdash y \in X \quad \Gamma; y:X \vdash y \in X}{\Gamma \vdash [y:X]y \in X \to X} \quad \Gamma; x:X \vdash x \in X}{\Gamma \vdash [x:X][y:X]y \in (X \to (X \to X))} \quad \Gamma \vdash u \in X}{\Gamma \vdash ([x:X][y:X]y\ \ u) \in X \to X} \quad \Gamma \vdash v \in X}{\Gamma \vdash (([x:X][y:X]y\ \ u)\ \ v) \in X}$$

Where Γ *represents the following environment:*$X : Prop, u : X, v : X$.

Now, we have to assign initial marks to start the algorithm. For that we begin to décorate *our proof tree to its root. We only know that the final type and term and the variables in the final environment have to be conserved, and hence, marked with a* c. *It gives the following derivation tree :*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; y:X \vdash y \in X \quad \Gamma; y:X \vdash y \in X}{\Gamma \vdash [y:X]y \in X \to X} \quad \Gamma; x:X \vdash x \in X}{\Gamma \vdash [x:X][y:X]y \in (X \to (X \to X))} \quad \Gamma \vdash u \in X}{\Gamma \vdash ([x:X][y:X]y\ \ u) \in X \to X} \quad \Gamma \vdash v \in X}{\Gamma \vdash (([x:X][y:X]y\ \ u)\ \ v)^c \in X^c}$$

Now using the rule :

$$(10)\ \dfrac{\Gamma \vdash t_1^j \in (A^i \to B^j)^j \qquad \Gamma \vdash t_2^i \in A^i}{\Gamma \vdash (t_1^j\ \ t_2^i)^j \in B^j}$$

we can deduce the following decoration:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; y:X \vdash y \in X \quad \Gamma; y:X \vdash y \in X}{\Gamma \vdash [y:X]y \in X \to X} \quad \Gamma; x:X \vdash x \in X}{\Gamma \vdash [x:X][y:X]y \in (X \to (X \to X))} \quad \Gamma \vdash u \in X}{\Gamma \vdash ([x:X][y:X]y\ \ u)^c \in (X^i \to X^c)^c} \quad \Gamma \vdash v^i \in X^i}{\Gamma \vdash (([x:X][y:X]y\ \ u)\ \ v)^c \in X^c}$$

where i *is a mark variable. One can already notice that this is the same as the one used to mark the first occurrence of* X *in* $(X^i \to X^c)^c$. *It is here that the realization of the* links *used in Boerio's algorithm is done in our system with marking. Still using the same rule one can lift the mark on the left premise to finally get:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; y:X \vdash y \in X \quad \Gamma; y:X \vdash y \in X}{\Gamma \vdash [y:X]y \in X \to X} \quad \Gamma; x:X \vdash x \in X}{\Gamma \vdash ([x:X][y:X]y)^c \in (X^j \to (X^i \to X^c)^c)} \quad \Gamma \vdash u^j \in X^j}{\Gamma \vdash ([x:X][y:X]y\ \ u)^c \in (X^i \to X^c)^c} \quad \Gamma \vdash v^i \in X^i}{\Gamma \vdash (([x:X][y:X]y\ \ u)\ \ v)^c \in X^c}$$

Where j *is a new mark variable. Now to progress in the tree on has to apply the following rule:*

$$(8)\ \dfrac{\Gamma, x :^i A \vdash M^j \in N^j \qquad \Gamma \vdash N^j \in Prop^j \qquad i' = sup(mark(occ(x, M)))}{\Gamma \vdash ([x :^{i'} A]M^j)^j \in (A^{i'} \to N^j)^j}$$

We then get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; y:X \vdash y \in X \quad \Gamma; y:X \vdash y \in X}{\Gamma \vdash ([y:X]y)^c \in (X^i \to X^c)^c} \quad \Gamma; x:X \vdash x^j \in X^j}{\Gamma \vdash ([x:X][y:X]y)^c \in (X^j \to (X^i \to X^c)^c)^c} \quad \Gamma \vdash u^j \in X^j}{\Gamma \vdash ([x:X][y:X]y\ \ u)^c \in (X^i \to X^c)^c} \quad \Gamma \vdash v^i \in X^i}{\Gamma \vdash (([x:X][y:X]y\ \ u)\ \ v)^c \in X^c}$$

Finally we obtain:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma; y : X \vdash y^i \in X^i \quad \Gamma; y : X \vdash y^c \in X^c}{\Gamma \vdash ([y : X]y)^c \in (X^i \to X^c)^c} \quad \Gamma; x : X \vdash x^j \in X^j}{\Gamma \vdash ([x : X][y : X]y)^c \in (X^j \to (X^i \to X^c)^c)^c} \quad \Gamma \vdash u^j \in X^j}{\Gamma \vdash ([x : X][y : X]y \;\; u)^c \in (X^i \to X^c)^c} \quad \Gamma \vdash v^i \in X^i}{\Gamma \vdash (([x : X][y : X]y \;\; u) \;\; v)^c \in X^c}$$

One can notice that variables x and y have no marks in their declaration in the context : we recall that by convention everything in the context is marked with c.

From the rule (8) it follows that $i = c$ and one has to replace every occurrences of i by c in our tree. As the j variable does not have any constraints and that we seek for the minimal marking we just have to choose it equal to r. We obtain the following marked term $(([x :^r X][y :^c X]y^c \;\; u^c)^c \;\; v^r)^c$, which , after extraction gives $([y : X]y \;\; u)$. Which was the result expected.

This very simple example shows how inference rules with marks can simulate the link of Boerio.

- **Simulation of extraction with subtyping**

The problem which has given birth to the subtyping was the fact that some optimizations were not allowed because they would lead to ill typed terms. In our system, this is not the case. Indeed, as the marking is independent from the typing mechanism, such problems are avoided.

Thus, we are able to simulate the subtyping mechanism simply by choosing the Ex_P extraction. It represents an extension to the second order of the work done by Berardi and Boerio in [3].

Theorem 6 *One can simulate the pruning technique with subtyping in F_2^m.*

Proof:

One only has to check that there is, for each Ω–type of Berardi–Boerio's system, a corresponding type in F_2^m.

The first thing to do is to find an equivalent for Ω. Clearly N^r, where N is the predefined constant representing integers, is the good choice.

Now it is clear that for each Ω–type one can give a term representing it in F_2^m. Indeed, an Ω–type is of kind $A \to O$ or $O \times O$. By induction, it is clear that we can build such types in F_2^m.

One just has to add for each Ω–type T a declaration of the form $d_T :^r T$ one which corresponds to the d_O constants in Berardi–Boerio's system.

Now one has to find the correspondence for application rule. Let us recall that this rule is, for more precision refer to section 2.1.4, of the form :

$$\dfrac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A' \quad A' \subseteq_\Omega A}{\Gamma \vdash (f \;\; a) : B}$$

while in our system one has the following application rule :

$$(10)\,\dfrac{\Gamma \vdash t_1^j \in (A^i \to B^j)^j \qquad \Gamma \vdash t_2^i \in A^i}{\Gamma \vdash (t_1^j \;\; t_2^i)^j \in B^j}$$

Now, let us look at the definition of $\subseteq_\Omega$:

$$N \subseteq_\Omega \Omega$$

$$N \subseteq_\Omega N \qquad\qquad \Omega \subseteq_\Omega \Omega$$

$$\dfrac{B_1 \subseteq_\Omega A_1 \quad B_1 \subseteq_\Omega A_1}{A_1 \to A_2 \subseteq_\Omega B_1 \to B_2} \qquad \dfrac{B_1 \subseteq_\Omega A_1 \quad B_1 \subseteq_\Omega A_1}{A_1 \times A_2 \subseteq_\Omega B_1 \times B_2}$$

If we follow our simulation, we have the following definition of $\subseteq_\Omega$ in our system:

$$N \subseteq_\Omega N^r$$

$$N \subseteq_\Omega N^c \qquad N^r \subseteq_\Omega N^r$$

$$\frac{B_1 \subseteq_\Omega A_1 \quad B_1 \subseteq_\Omega A_1}{A_1 \to A_2 \subseteq_\Omega B_1 \to B_2} \qquad \frac{B_1 \subseteq_\Omega A_1 \quad B_1 \subseteq_\Omega A_1}{A_1 \times A_2 \subseteq_\Omega B_1 \times B_2}$$

One has to check that the typing rule (10) is equivalent to the one given for Boerio–Berardi system.

Clearly, as we have the same type, only marks are changing. In both parts of an expression of the form $\subseteq_\Omega B$ we can build a derivation in F_2^m such that $\Gamma \vdash f^i \in (A^j \to C^i)^j$ if there exists a derivation of $\Gamma \vdash f^i \in (B^j \to C^i)^j$

□

We have shown that our approach of extraction is relevant since it allows to represent every known technique. On this basis we are going to extend Berardi–Boerio's approach to higher level. The natural language to consider is F_ω, because it is a higher order language simpler than the calculus of Construction as there are no dependences from low level to high level.

3.3 Marking in a higher order language : F_ω^m

3.3.1 Presentation of F_ω system

The F_ω system has been introduced by J.–Y. Girard in [6]. It is more powerful than F_2. It presents some features which make it the *natural* system to generalize Boerio and Berardi's technic.

- The first point is the expressivity of the system. F_ω allows to express all programs and it has the same computational power as the Calculus of Constructions.
- The second point concerns its hierarchical structure. The language is structured in such a way that objects of an higher level cannot have an influence on objects situated below them. This makes the task easier.
- The third important feature comes from *how* are structured the levels of objects in F_ω. Their structure is such that, apart few remarks, they strongly remind languages of the first and second order. The idea is to re-apply to different levels what we already know.

We will note Λ_F the language of F_ω. In this language we can distinguish three kinds of objects. Orders, operators, terms. These three kinds of objects are defined in three successive levels beginning with orders followed by operators, which can depend on orders and finally terms which can depend on both orders and operators. We will talk about *level* to designate orders, operators or terms.

More precisely, we have :

- Orders (level 2): The constant $Data$ is an order and if A and B are orders, then $A \Rightarrow B$ is also an order.
- Operators (level 1): If σ, τ are operators, X an operator variable and A an order then $\sigma \to \tau$ (functions), $(\sigma \ \ \tau)$ (applications), $(X : A)\sigma$ (products) and $[X : A]\sigma$ (abstraction) are operators.
- Terms(level 0) : If s, t are terms and x a term variable (and by taking again the previous conventions) then $(s \ \ t)$, $(s \ \ \sigma)$, $[x : \sigma]s$, $[X : A]s$ are also terms.

An order context is a finite sequence of pairs (X, A), where X is an operator variable and A an order.

Typing rules

As the notion of *order* is purely syntactical, we begin to define the relation $\Sigma \vdash \sigma \in A$, which means "the operator σ is well formed of type A in the order context Σ.

The typing rules for operators are :

$$\frac{if \quad X : A \quad appears \quad in \quad \Sigma}{\Sigma \vdash X \in A}$$

$$\frac{\Sigma \vdash \sigma \in Data \quad \Sigma \vdash \tau \in Data}{\Sigma \vdash \sigma \to \tau \in Data}$$

$$\frac{X : A, \Sigma \vdash \sigma \in Data}{\Sigma \vdash (X : A)\sigma \in Data}$$

$$\frac{X : A, \Sigma \vdash \sigma \in B}{\Sigma \vdash [X : A]\sigma \in A \Rightarrow B}$$

$$\frac{\Sigma \vdash \sigma \in A \Rightarrow B \quad \Sigma \vdash \tau \in A}{\Sigma \vdash (\sigma \quad \tau) \in B}$$

For terms we have also to define the notion of environment. We want that the declaration of variables precedes the declaration of operators. So, we introduce the following definition for environments:

- The empty sequence [] is a valid environment and the associated order context $[]_o$ is the empty sequence.
- If Γ is a valid environment, X an operator variable which is not in Γ and A an order then $\Gamma, X : A$ is a valid environment and $(\Gamma, X : A)_o = \Gamma_o, X : A$
- If Γ is a valid environment, x a term variable that does not appear in Γ and if there is a derivation $\Gamma_o \vdash \sigma \in Data$ where$(\Gamma, x : \sigma)_o = \Gamma_o$ then $\Gamma, x : \sigma$ is a valid environment and.

One now defines a typing relation for terms which will be noted $\Gamma \vdash t \in \sigma$ with Γ a valid environment, t a term and σ an operator.

The typing rules for terms :

$$\frac{x : \sigma \quad appears \quad in \ \Gamma}{\Gamma \vdash x \in \sigma}$$

$$\frac{\Gamma \vdash t \in \sigma \to \tau \quad \Gamma \vdash u \in \sigma' \quad \sigma =_{\beta\eta} \sigma'}{\Gamma \vdash (t \quad u) \in \tau}$$

$$\frac{\Gamma, x : \sigma \vdash t \in \tau}{\Gamma \vdash [x : \sigma]t \in \sigma \to \tau}$$

$$\frac{\Gamma \vdash t \in (X : A)\sigma \quad \Gamma_O \vdash \tau \in A}{\Gamma \vdash (t \quad \tau) \in \sigma[X := \tau]}$$

$$\frac{\Gamma, X : A \vdash t \in \sigma}{\Gamma \vdash [X : A]t \in (X : A)\sigma}$$

3.3.2 Informal semantics of marking in this system

Our aim is to extend the techniques of the Turin school to F_ω^m while keeping our approach of marking. We have to think about what meaning have the marking and optimization on higher orders.

The first observation which comes to thought is that F_ω^m is divided in three partsm, which are hierarchically comparable. The Level 2 (orders) does not raise big problems : to be an order is a too much syntactic feature to search any optimization. Level one (operators) is already more interesting. At a first glance, one could think that the language generated by it is a simply typed language (akin of Gödel $\mathcal{T}$ system), like the one studied by Berardi. Nevertheless, one rule shows us it is not exactly the case :

$$\frac{\Sigma \vdash \sigma \in Data \quad \Sigma \vdash \tau \in Data}{\Sigma \vdash \sigma \to \tau \in Data}$$

Indeed, the $Data$ type is *absorbent* with respect to operators of kind $\to$. This means that, if we wanted to apply directly Berardi's technique we would have a bad surprise. To detect the informative parts of a term, Berardi uses the fact that one has just to conserve the final type of the term. " I serve for the building of the final type" is the only information needed to find where we can prune or not. Now, it is clear that this criterion is no longer enough in this system. Indeed, if σ, τ are of type $Data$, then $\sigma \to \tau$ and τ will have the same type if we follow the typing rule. So, the question is the contribution to the final type of $\sigma \to \tau$? If this rule didn't exist, one would have almost exactly Berardi's system. Actually, as we will see later, one just have to define a suitable equivalence relation to solve the problem.

Level 0, the one of terms, is alike, apart few details as a second order system. The only notable differences are coming from the creation of the context. One can introduce a term variable only if the type of this variable is a well formed operator. So, when we are in a leaf of the tree and this leaf corresponds to the introduction of a variable, one has to build *another* proof tree for the well formation of the operator designing the type of the variable. The second *bridge* between those two levels comes from the following rule :

$$\frac{\Gamma \vdash t \in \sigma \to \tau \quad \Gamma \vdash u \in \sigma' \quad \sigma =_{\beta\eta} \sigma'}{\Gamma \vdash (t \ \ u) \in \tau}$$

For this last rule, one has to go to the operator's language to prove the equivalence between the two operators. This means that, modulo the last remarks, we are in a second order system, in which one can apply the already known methods to optimize terms.

Now, to optimize a term in F_ω, while keeping our point of view of marking, one has to solve the problems raised by the operators of kind $\sigma \to \tau$, and the equivalence between two marked operators. We propose, in the next section, a language F_ω^m which enables to solve these problems.

3.3.3 The F_ω^m language

Syntax of F_ω^m

For the creation of F_ω^m language we follow the general ideas already developed for F_2. For uniformisation reasons, we decided to mark orders, despite the remarks raised in the last section which have clearly shown that we cannot expect optimization for those terms.

As usual, in the rest of the text, the indices $i, j, k, \ldots$ will indistinctly represent r or c. One find again the main lines of F_ω language.

- Orders: The constants $Data^i$ are orders and if A^i and B^j are orders, then $(A^i \Rightarrow B^j)^j$ is also an order.
- Operators: If σ^i, τ^j are operators, X^k an operator variable and A^l an order then $(\sigma^i \to \tau^j)^m$ (functions), $(\sigma^i \ \ \tau^j)^m$ (applications), $(X^k : A^l)\sigma^m$ (products) and $[X^k : A^l]\sigma^i$ (abstraction) are operators.
- Terms : if s^n, t^o are terms and x^p a term variable (and by taking again the previous conventions) then $(s^n \ \ t^o)^m$, $(s^n \ \ \sigma^i)^m$, $([x^p : \sigma^i]s^n)^m$, $([X^k : A^l]s^n)^m$ are also terms.

The syntax is the same than in F_ω, apart the fact that all subterms are marked with a r or a c. One still has to define typing rules for this language. One naturally extend those marking notions to the formation of order context, and of environment for terms. The formation rules are the same as preceding ones, and one simply adds marks.

Optimization in F_ω^m

We have pointed out two points in the discussion on optimization in the last section. Let us see how we can handle them. First we will denote Op the language defined by the typing rules on operators in F_ω.

For the first point, we will simplify the language Op in such a way that we will obtain a simply marked language.

Definition 10 *The Tr function is inductively defined by :*

$$\begin{array}{lcl} Tr(\sigma \to \tau) & = & Tr(\tau) \\ Tr([X:A]\sigma) & = & [X:A]Tr(\sigma) \\ Tr((X:A).\sigma) & = & (X:A).Tr(\sigma) \\ Tr((\sigma \ \ \tau)) & = & (Tr(\sigma) \ \ Tr(\tau)) \end{array}$$

We will denote by $\sim$ the equivalence relation generated by Tr function, which is to say:

$$\sigma \sim \tau \iff Tr(\sigma) = Tr(\tau)$$

One can already observe that $\overline{\sigma \to \tau} = \overline{\tau}$, where $\overline{\tau}$ represents the equivalence class of τ. This way a particular representant of the class of an object formed by many $\to$, is the rightmost situated subterm.

Theorem 7 *The system generated by $Op/\sim$ is a simply typed λ–calculus.*

So, one can apply the results precedently obtained for the first order in $Op/\sim$.

To obtain an equivalent to F_2^m marked system, one has just to manage with equivalence classes in a way to avoid to *forget* some informative subterms. In clear one has to force the terms of kind $\sigma \to \tau$ to be marked in the same way on the right and on the left of the arrow unless the user specifies the contrary. *Thus this remark does not play its role at the logical level (i.e. in typing rules) but during the development of the marking algorithm ! It has to be understood as a marking strategy.*

One obtains the following rules :

$$\frac{if \ \ X :^i A \ \ appears \ \ in \ \ \Sigma}{\Sigma \vdash X^{i'} \in A^{i'}}$$

$$\frac{\Sigma \vdash \sigma^j \in Data^j \quad \Sigma \vdash \tau^i \in Data^i}{\Sigma \vdash (\sigma^j \to \tau^i)^i \in Data^i}$$

$$\frac{X :^i A, \Sigma \vdash \sigma^j \in Data^j \quad i' = sup(mark(occ(X,\sigma)))}{\Sigma \vdash ((X :^{i'} A).\sigma^j)^j \in Data^j}$$

$$\frac{X :^i A, \Sigma \vdash \sigma^j \in Data^j \quad i' = sup(mark(occ(X,\sigma)))}{\Sigma \vdash ([X :^{i'} A]\sigma^j)^j \in (A^{i'} \Rightarrow B^j)^j}$$

$$\frac{\Sigma \vdash \sigma^i \in (A^j \Rightarrow B^i)^i \quad \Sigma \vdash \tau^j \in A^j}{\Sigma \vdash (\sigma^i \ \ \tau^j)^i \in B^i}$$

For the second point, one has to define an equivalence between marked operators. Remember that types represent set of terms. Now the question is not "when a marked term is equivalent to another marked term" but " when two sets of terms are equivalent". From this point of view, the answer comes more easily. When you realize that a type marked with r means that it has no informative content you are getting very close to the solution.

Knowing the semantics of the marking (Keep what is marked with c and erase what is marked with r) the researched equivalence is easy to find. What we seek is the following:

- If the mark is r, we can forget about marks and only care about standard $\beta\eta$ equivalence.
- In the case when the mark is c, we have to check both standard $\beta\eta$ equivalence and also the compatibility of marks inside the terms.

All these remarks lead to the following definition of equivalence between operators. We will note σ^* the F_ω term corresponding to the marked term σ^i. σ^* is simply obtained by erasing all marks in σ^i.

Definition 11 *We inductively define $=_m$ by :*

$$\frac{\sigma^i \to_\beta \tau^i}{\sigma^i =_m \tau^i}$$

$$\frac{}{\sigma^i =_m \sigma^i} \quad \frac{\sigma^i =_m \tau^i}{\tau^i =_m \sigma^i} \quad \frac{\sigma^i =_m \tau^i \quad \tau^i =_m \rho^i}{\sigma^i =_m \rho^i}$$

$$\frac{\sigma^c =_m \tau^c}{([X :^i A]\sigma^c)^c =_m ([X :^i A]\tau^c)^c} \qquad \frac{\sigma^r =_m \tau^r}{([X :^i A]\sigma^r)^r =_m ([X :^j A]\tau^r)^r}$$

$$\frac{\sigma^c =_m \tau^c}{((X :^i A).\sigma^c)^c =_m ((X :^i A)\tau^c)^c} \qquad \frac{\sigma^r =_m \tau^r}{((X :^i A).\sigma^r)^r =_m ((X :^j A)\tau^r)^r}$$

$$\frac{\sigma^c =_m \tau^c \quad \rho^i =_m \varphi^i}{(\rho^j \to \sigma^c)^c =_m (\varphi^j \to \tau^c)^c} \qquad \frac{\sigma^r =_m \tau^r \quad \rho^r =_m \varphi^r}{(\rho^j \to \sigma^r)^r =_m (\varphi^k \to \tau^r)^r}$$

$$\frac{\sigma^c =_m \tau^c \quad \rho^i =_m \varphi^i}{(\sigma^c \quad \rho^i)^c =_m (\tau^c \quad \varphi^i)^c} \qquad \frac{\sigma^r =_m \tau^r \quad \rho^i =_m \varphi^i}{(\sigma^r \quad \rho^j)^r =_m (\tau^r \quad \varphi^k)^r}$$

One easily checks the following properties.

Property 6 *Let σ^i, τ^j be marked terms. The following facts are verified :*

1. $\sigma^i =_m \tau^i \Rightarrow \sigma^* =_{\beta\eta} \tau^*$
2. $\sigma^* =_{\beta\eta} \tau^* \Rightarrow \sigma^r =_m \tau^r$

From this equivalence relation, one obtains the typing rules for terms:

$$\frac{x :^i \sigma \quad appears \quad in \ \Gamma}{\Gamma \vdash x^{i'} \in \sigma^{i'}}$$

$$\frac{\Gamma \vdash t^i \in (\sigma^j \to \tau^i)^i \quad \Gamma \vdash u^j \in \sigma'^j \quad \sigma^j =_m \sigma'^j}{\Gamma \vdash (t^i \quad u^j)^i \in \tau^i}$$

$$\frac{\Gamma, x :^i \sigma \vdash t^j \in \tau^j \quad i' = sup(mark(occ(x, \sigma)))}{\Gamma \vdash ([x :^{i'} \sigma]t^j)^j \in (\sigma^{i'} \to \tau^j)^j}$$

$$\frac{\Gamma \vdash t^i \in ((X :^j A)\sigma^i)^i \quad \Gamma_O \vdash \tau^j \in A^j}{\Gamma \vdash (t^i \quad \tau^j)^j \in \sigma^i([X^j := \tau^j])^i}$$

$$\frac{\Gamma, X :^i A \vdash t^j \in \sigma^j \quad i' = sup(mark(occ(X, t)))}{\Gamma \vdash [X :^{i'} A]t^j \in ((X :^{i'} A).\sigma^j)^j}$$

These rules are almost the same as ones developed in F_2^m. Only the definition of the $\beta\eta$–equivalence and the context formation differ from a second order language.

Now, we prove the validity of our marking system by defining a new extracting function : EX_{F_ω}.

Definition 12 *The function EX_{F_ω} is inductively defined as follows.:*

- *On terms:*
  - $EX_{F_\omega}(x^c) = \overline{x}$

- $EX_{F_\omega}((t^j \quad u^r)^j) = EX_{F_\omega}(t^j)$
- $EX_{F_\omega}((t^j \quad u^c)^j) = (EX_{F_\omega}(t^j) \quad EX_{F_\omega}(u^c))$
- $EX_{F_\omega}(([x :^r \sigma]t^j)^j) = EX_{F_\omega}(t^j)$
- $EX_{F_\omega}(([x :^c \sigma]t^j)^j) = [x : EX_{F_\omega}(\sigma^c)]EX_{F_\omega}(t^j)$
- $EX_{F_\omega}((t^j \quad \tau^r)^j) = EX_{F_\omega}(t^j)$
- $EX_{F_\omega}((t^j \quad \tau^c)^j) = (EX_{F_\omega}(t^j) \quad EX_{F_\omega}(\tau^c))$
- $EX_{F_\omega}(([X :^r \sigma]t^j)^j) = EX_{F_\omega}(t^j)$
- $EX_{F_\omega}(([X :^c \sigma]t^j)^j) = [X : EX_{F_\omega}(\sigma^c)]EX_{F_\omega}(t^j)$

- *On operators:*
  - $EX_{F_\omega}(X^c) = \overline{X}$
  - $EX_{F_\omega}((\sigma^r \to \tau^i)^i) = EX_{F_\omega}(\tau^i)$
  - $EX_{F_\omega}((\sigma^c \to \tau^i)^i) = EX_{F_\omega}(\sigma^c) \to EX_{F_\omega}(\tau^i)$
  - $EX_{F_\omega}(([X :^r A]\sigma^j)^j) = EX_{F_\omega}(\sigma^j)$
  - $EX_{F_\omega}(([X :^c A]\sigma^j)^j) = [X : EX_{F_\omega}(A^c)]EX_{F_\omega}(\sigma^j)$
  - $EX_{F_\omega}(((X :^r A)\sigma^j)^j) = EX_{F_\omega}(\sigma^j)$
  - $EX_{F_\omega}(((X :^c A)\sigma^j)^j) = (X : EX_{F_\omega}(A^c))EX_{F_\omega}(\sigma^j)$
  - $EX_{F_\omega}((\tau^j \quad \sigma^r)^j) = EX_{F_\omega}(\tau^j)$
  - $EX_{F_\omega}((\tau^j \quad \sigma^c)^j) = (EX_{F_\omega}(\tau^j) \quad EX_{F_\omega}(\sigma^c))$

- *On orders:*
  - $EX_{F_\omega}(Data^c) = Data$
  - $EX_{F_\omega}((O^r \Rightarrow P^j)^j) = EX_{F_\omega}(P^j)$
  - $EX_{F_\omega}((O^c \Rightarrow P^j)^j) = EX_{F_\omega}(O^c) \Rightarrow EX_{F_\omega}(P^j)$

Theorem 8 *If $\Gamma \vdash t^c \in \tau^c$ is derivable in F^m_ω then $EX_{F_\omega}(\Gamma) \vdash EX_{F_\omega}(t^c) \in EX_{F_\omega}(\tau^c)$ is derivable in F_ω.*

Proof:

We prove this theorem by a structural induction on orders, operators and terms.

On orders it is clear that extraction gives a well formed term. Indeed if an order is marked with c, then at least one $Data$, namely the rightest one, will be conserved by extraction.

For the level of operators, one can re-use what was previously done for F^m_2. For the ground case it works, since extraction on orders works. Now, we can reason by induction. It is clear that for the cases of application and abstraction the result will be verified (the proof is similar to what was done in F^m_2). There are only two cases left.

1. The operator is of the form $(\sigma^i \to \tau^c)^c$, this means that the rule

$$\frac{\Sigma \vdash \sigma^i \in Data^i \quad \Sigma \vdash \tau^c \in Data^c}{\Sigma \vdash (\sigma^i \to \tau^c)^c \in Data^c}$$

   has been applied. By definition, $Ex_{F^m_\omega}(\sigma^i \to \tau^c) = Ex_{F^m_\omega}(\tau^c)$ if $i = r$ and $Ex_{F^m_\omega}(\sigma^i) \to Ex_{F^m_\omega}(\tau^c)$ if $i = c$. In both cases, one can use the induction hypothesis to prove that both terms are of type $Ex_{F^m_\omega}(Data^c) = Data$.

2. The operator is of the form $((X :^{i'} A)\sigma^c)^c$. Here too, the extracted order will be $Data$ and it is clear, from the definition of extraction that whether $i = c$ or r that the extracted term will be of type $Data$ in F_ω

Now one has to prove the final result, i.e. the extracting function on terms gives well formed terms in F_ω.

The ground cases, as the cases of abstraction of operators and terms on terms as well as the application of operators on terms, are verified in the same way than what was done for F_2^m. It remains the case of application of terms on terms. The novelty comes from the utilization of equivalence on types.

Let us consider the term $s \equiv (t^c \ \ u^j)^c$. This means that the rule

$$\frac{\Gamma \vdash t^c \in (\sigma^j \to \tau^c)^c \quad \Gamma \vdash u^j \in \sigma'^j \quad \sigma^j =_m \sigma'^j}{\Gamma \vdash (t^c \ \ u^j)^c \in \tau^c}$$

has been applied. Two subcases are to be considered :

1. $j = r$. $Ex_{F_\omega^m}(s) = Ex_{F_\omega^m}(t^c)$. But,$Ex_{F_\omega^m}((\sigma^r \to \tau^c)^c) = Ex_{F_\omega^m}(\tau^c)^c$. This allows to conclude by induction hypothesis. Indeed this hypothesis states that $Ex_{F_\omega^m}(\Gamma) \vdash Ex_{F_\omega^m}(t^c) \in Ex_{F_\omega^m}((\sigma^r \to \tau^c)^c)$, which can be rewritten $Ex_{F_\omega^m}(\Gamma) \vdash Ex_{F_\omega^m}(t^c) \in Ex_{F_\omega^m}(\tau^c)$, hence the result.
2. $j = c$. We have to show that $Ex_{F_\omega^m}(\Gamma) \vdash Ex_{F_\omega^m}(t^c \ \ u^c) \in Ex_{F_\omega^m}((\sigma^c \to \tau^c)^c)$. We know by definition of extraction that :

   - $Ex_{F_\omega^m}((t^c \ \ u^c)^c) = (Ex_{F_\omega^m}(t^c) \ \ Ex_{F_\omega^m}(u^c))$.
   - $Ex_{F_\omega^m}(\tau^c \to \sigma^c) = Ex_{F_\omega^m}(\tau^c) \to Ex_{F_\omega^m}(\sigma^c)$

   Hence, we have to prove that $Ex_{F_\omega^m}(\Gamma) \vdash (Ex_{F_\omega^m}(t^c) \ \ Ex_{F_\omega^m}(u^c)) \in Ex_{F_\omega^m}(\sigma^c)$ which is clear if we look at the induction hypothesis which are the following :

   - $Ex_{F_\omega^m}(\Gamma) \vdash Ex_{F_\omega^m}(t^c) \in Ex_{F_\omega^m}(\sigma^c \to \tau^c)$
   - $Ex_{F_\omega^m}(\Gamma) \vdash Ex_{F_\omega^m}(u^c) \in Ex_{F_\omega^m}(\sigma'^c)$
   - $\sigma^c =_m \sigma'^c$

   If we can prove that $\sigma^c =_m \sigma'^c$ implies that $Ex_{F_\omega^m}(\sigma^c) =_{\beta\eta} Ex_{F_\omega^m}(\sigma'^c)$ then the case will be proved. This is easy to show. Indeed we have shown in Property 5 that the marking is stable under β–reduction. Thus if $\sigma^c \to_\beta \sigma'^c$, then all subterms marked with r in σ^c will also be marked with an r in σ'^c hence the extraction, which erase terms marked with c, will leave two terms such that $Ex_{F_\omega^m}(\sigma^c) \to_\beta^* Ex_{F_\omega^m}(\sigma'^c)$.

□

3.4 Marking in the Calculus of Constructions

Now, using what was previously done in F_ω we propose a marking system for the calculus of constructions. The difference between F_ω and the calculus of construction comes from the dependent types. So our first aim will be to handle the problems raised by this feature. Until now, we have always performed extraction within the same system. For example in F_2^m, when extraction was done it was giving a new term typable in F_2. The same remark was also valid for F_ω : an extracted term of F_ω^m was an F_ω term. For the Calculus of Construction this is no longer the case. We will extract from the Calculus of Constructions to F_ω. Thus one has to erase these kind of dependences.

The language of our system, *the Calculus of Constructions with marks*, will be Λ_m already defined for F_2^m. Here too, two kinds of judgments will be done. Γ is a valid environment and $\Gamma \vdash M \in N$, which has to be read " M is well formed of type N under the valid environment Γ".

- Environment formation:

$$\frac{}{[] \ \ is \ \ valid}$$

$$\frac{\Gamma \vdash M^i \in \mathcal{K}^i \quad x \in \Lambda \ \ does \ \ appear \ \ in \ \ \Gamma \quad \Gamma \ \ is \ \ valid}{\Gamma, x :^{i'} M \ \ is \ \ valid}$$

- Propositional type formation:

$$\frac{\Gamma \ \ is \ \ valid \quad x :^i N \ \ appears \ \ in \ \ \Gamma \quad N^i \in \Lambda_m}{\Gamma \vdash x^{i'} \in N^{i'}}$$

- Hypotheses:

$$\frac{\Gamma \ \ is \ \ valid}{\Gamma \vdash Prop^i \in Type^i}$$

- Product :
  - Propositional types:

$$\frac{\Gamma \vdash P^i \in Prop^i \quad \Gamma, x :^{i'} P \vdash N^j \in Type^j}{\Gamma \vdash ((x :^r P)N^j)^j \in Type^j}$$

$$\frac{\Gamma \vdash O^i \in Prop^i \quad \Gamma \vdash P^{i'} \in O^{i'} \quad \Gamma, x :^{i''} P \vdash N^j \in Type^j}{\Gamma \vdash ((x :^r P)N^j)^j \in Type^j}$$

$$\frac{\Gamma \vdash P^i \in Type^i \quad \Gamma, x :^{i'} P \vdash N^j \in Type^j \quad i'' = sup(mark(occ(x, N))}{\Gamma \vdash ((x :^{i''} P)N^j)^j \in Type^j}$$

  - Propositional schemes :

$$\frac{\Gamma \vdash P^i \in Type^i \quad \Gamma, x :^{i'} P \vdash N^j \in Prop^j \quad i'' = sup(mark(occ(x, N))}{\Gamma \vdash ((x^{i'} P)N^j)^j \in Prop^j}$$

- Abstraction :
  - Propositional schemes:

$$\frac{\Gamma \vdash N^j \in Type^j \quad \Gamma, x :^i P \vdash M^j \in N^j \quad i' = sup(mark(occ(x, N))}{\Gamma \vdash ([x :^{i'} P]M^j)^j \in ((x :^{i'} P)N^j)^j}$$

$$\frac{\Gamma \vdash N^j \in Prop^j \quad \Gamma, x :^i P \vdash M^j \in N^j}{\Gamma \vdash ([x :^r P]M^j)^j \in ((x :^r P)N^j)^j}$$

  - Proofs:

$$\frac{\Gamma \vdash N^j \in Prop^j \quad \Gamma, x :^i P \vdash M^j \in N^j \quad i' = sup(mark(occ(x, N))}{\Gamma \vdash ([x :^{i'} P]M^j)^j \in ((x :^{i'} P)N^j)^j}$$

- Application:
  - Propositional schemes :

$$\frac{\Gamma \vdash M^j \in ((x :^i P)N^j) \quad \Gamma \vdash R^i \in Prop^i \quad P^i =_m Prop^i}{\Gamma \vdash (M^j \ R^i) \in (N^j[x^i/R^i])}$$

$$\frac{\Gamma \vdash M^j \in ((x :^i P)N^j) \quad \Gamma \vdash R^i \in Q^i \quad \Gamma \vdash Q^i \in Prop^i \quad P^i =_m Q^i}{\Gamma \vdash (M^j \ R^r) \in (N^j[x^r/R^r])}$$

– Proofs:

$$\frac{\Gamma \vdash M^j \in ((x :^i P)N^j) \quad \Gamma \vdash R^i \in Q^i \quad \Gamma \vdash Q^i \in Prop^i \quad P^i =_m Q^i}{\Gamma \vdash (M\ R) \in (N[x/R])}$$

- Equality:

$$\frac{\Gamma \vdash M^i \in N^i \quad \Gamma \vdash P^i \in \mathcal{K}^i \quad N^i =_m P^i}{\Gamma \vdash M^i \in P^i}$$

$=_m$ is inductively defined as in F^m_ω by induction :

Definition 13 *we inductively define* $=_m$ *by :*

$$\frac{M^i \to_\beta N^i}{M^i =_m N^i}$$

$$\frac{}{M^i =_m M^i} \quad \frac{M^i =_m N^i}{N^i =_m M^i} \quad \frac{M^i =_m N^i \quad N^i =_m P^i}{M^i =_m P^i}$$

$$\frac{M^c =_m N^c}{([X :^i A]M^c)^c =_m ([X :^i A]N^c)^c} \qquad \frac{M^r =_m N^r}{([X :^i A]M^r)^r =_m ([X :^j A]N^r)^r}$$

$$\frac{M^c =_m N^c}{((X :^i A).M^c)^c =_m ((X :^i A)N^c)^c} \qquad \frac{M^r =_m N^r}{((X :^i A).M^r)^r =_m ((X :^j A)N^r)^r}$$

$$\frac{M^c =_m N^c \quad P^i =_m Q^i}{(M^c\ \ P^i)^c =_m (N^c\ \ Q^i)^c} \qquad \frac{M^r =_m N^r \quad P^i =_m Q^i}{(M^r\ \ P^j)^c =_m (N^r\ \ Q^k)^r}$$

One can notice that the case of dependent types has been dealt by the typing rules. If there is a dependence from a lower level to a higher level, than the term realizing this dependence is marked with an r. So it will be erased by extraction, hence we will obtain a term typable in F_ω by extraction. We define the extraction function as follows :

Definition 14 *The function* EX_{CCm} *is inductively defined by :*

- $EX_{CC^m}(Prop^c) = Prop$
- $EX_{CC^m}(((x :^r M)N^c)^c) = EX_{CC^m}(N^i)$
- $EX_{CC^m}(((x :^c M)N^c)^c) = (x : M)EX_{CC^m}(N^i)$
- $EX_{CC^m}(([x :^r M]N^c)^c) = EX_{CC^m}(N^i)$
- $EX_{CC^m}(([x :^c M]N^c)^c) = [x : M]EX_{CC^m}(N^i)$
- $EX_{CC^m}((M^c\ \ N^r)^c) = EX_{CC^m}(M^c)$
- $EX_{CC^m}((M^c\ \ N^c)^c) = (EX_{CC^m}(M^c)\ \ EX_{CC^m}(N^c))$

Theorem 9 (Validity of EX_{CCm}**)** *If* $\Gamma \vdash M^c \in N^c$ *in* CC^m *then*

$$EX_{CC^m}(\Gamma) \vdash EX_{CC^m}(M^c) \in EX_{CC^m}(N^c)$$

is valid both in the Calculus of Constructions and F_ω*.*

Proof:

The proof is essentially the same as done in section 3.3.3. For the application, abstraction, product, introduction of variables it is exactly similar.

Only the case of type equality is different. But, since it has been shown that if $M^c =_m N^c$ implies that $EX_{CC^m}(M^c) =_{\beta\eta} EX_{CC^m}(N^c)$, which is shown in the proof of the validity of $EX_{F_\omega^m}$, then the first part of the demonstration is ended.

Now, we have still to show that the extracted term is well typed in F_ω. Since we have erased the dependences of inferior types to superior types, we can apply the result shown in [9] to prove that the term is well typed in F_ω.

□

Theorem 10 *One can simulate Paulin's extraction in the calculus of constructions in CC^m.*

Proof:

The simulation will be conducted in the same manner as for the calculus of constructions restricted to the second order. If a term is typed by $Prop$ in Paulin's system, we will type it by $Prop^r$ in CC^m, and if it is typed by Set we will type it by $Prop^c$ in CC^m.

We have already shown that, under these assumptions, the terms marked with an r in our system and terms having an empty content in Paulin's system correspond. Moreover, we know that the dependences are handled correctly. Hence, the result announced.

□

4 Conclusion and future works

We have proposed a new frame to report on problems linked to extraction : the marking.This new approach allows us to unify various approaches known in this domain. By modifying some aspects of a marking system, i.e. the extraction function and the initial conditions, one can simulate all of the extraction techniques. So, it seems that marking is a sound point of view. Moreover, marking systems are flexible. The user can, if he wants to, mark certain parts of the proof tree. The system will then just check that the marking rules can be applied to find a term with such a marking. By using this flexibility and soundness approach we have been able to extend immediately Berardi and Boerio's approach regarding sub–typing to second order.

To higher orders, one could easily find a *semantics* of extraction. The observational equality was a good device to conduct extraction. An interesting work is suggested by the research of such a device for higher orders. Indeed, if we have shown that extraction is valid for higher orders it still remains to find *how* to build a marking satisfying some condition on the extracted term. In [10], Paulin has developed a notion of realizability to encounter this problem. It is maybe a good way to go further in the understanding of marking, but one can also try to develop a semantics based on c.p.o. or another interpretation domain.

One can also remark that we have developed a specific approach for the Calculus of Construction. We have chosen to perform extraction with F_ω as target language. This has led to introduce the rules which explicitly destroy the dependences of low level to high level. One could now study what would happen if we choosed to extract within the Calculus of Constructions : how the mark are going to react ? Are they going to behave like in our proposed system and "destroy by themselves" dependences?

In the future, we will work on the Calculus of Construction with marks to provide a theoretical study which is still to be done. Also we will try to develop *a logical point of view* for various software techniques, as we have done here for marking.

Acknowledgments

This part is dedicated to the shadow workers, who by their numerous and clever remarks or questions have contributed, very broadly to this work. Among them, I would like to cite P. Audebaud and C. Paulin.

Every discussion brings its part of creation. Many people have helped me more than they can imagine, and I am afraid to not really value well their importance.

To all those people I can just say thank you and try to pay them back with words and remarks which will (i hope) bring them the spark they look for, just as they have done for me.

I also would like to thank M. Gengler for his very deep and helpful proofreading.

Bibliography

[1] S. Berardi, *Pruning Simply λ-terms*, Technical Report, Turin University, 1993.

[2] L. Boerio, *Extending Pruning Techniques to Polymorphic Second Order λ-Calculus*, Technical Report, Turin University.

[3] S.Berardi L. Boerio, *Using Subtyping in Program Optimization*, Proc. of TLCA'95, in Lectures Note in Computer Science, 902, Springer-Verlag, 1995.

[4] R.L. Constable et al., *Implementing Mthematics with the Nuprl Proff Development System*, Prentice Hall, 1986.

[5] C. Cornes et al., The Coq Proff Assistant Reference Manual, technical Report INRIA 0177,1995.

[6] J.-Y, Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.*, PhD thesis, Universitè Paris 7, 1972.

[7] C. Goad, *Proofs as description of computation*, Proc. of the 5^{th} Conference on Automated Deduction, Les Arcs, Lectures Notes in Computer Science, 87, Springer-Verlag, 1980.

[8] G. Huet, *A uniform approach to type theory.* Logical foudations of functionnal programing, Addison-Wesley 1990.

[9] C. Paulin, *Extraction de programmes dans le calcul des constructions*, PhD thesis, Unniversité Paris 7, 1989.

[10] C. Paulin, *Informative contents as annotations in the Calculus of Inductive Constructions*, Manuscript 1995.

[11] Y. Takayama, *Extraction of Redundancy-free Programs from Constructive Natural Deduction Proofs*, Journal of Symbolic Computation, 1991, 12, 29-69.