Extending Pruning Techniques to Polymorphic Second Order λ-Calculus

Luca Boerio

Dipartimento di Informatica, Universita' di Torino
Corso Svizzera 185, 10149 Torino, Italy
lucab@di.unito.it

Abstract. Type theory and constructive logics allow us, from a proof of a formula, to extract a program that satisfies the specification expressed by this formula. Normally, programs extracted in this way are inefficient. Optimization algorithms for these programs have been developed. In this paper we show an algorithm to optimize programs represented by second order typed λ-terms. We prove also that the simplified programs are observational equivalent to the original ones.

1 Introduction

Constructive logics can be used to write the specifications of programs as logic formulas to be proved. Writing programs as constructive proofs of such formulas, is a good attempt to automate programming and program verification. We can extract executable code from constructive proofs, using the Curry-Howard isomorphism of formula-as-types ([8]) or the notion of realizability ([11] and [1]). Following these ideas, some tools have been introduced to help programmers in developing proofs or for the automatic extraction of programs, for example COQ, LEGO, NUPRL ([5] and [6]).
Automatic program extraction has a great problem: often, the extracted code is not efficient. Many attempts have been done to develop methods for the automatic erasing of redundant parts from these programs.
For example, in intuitionistic logic, the formula $\exists x.A(x)$ is interpreted as a pair, whose first element is a term t and whose second element is a proof of A(t). Generally, only the value t of x is needed as program extracted. The proof of A(t), from a computational point of view, is meaningless. Such a proof is not useful for the computation of result. It is only useful to prove that the program meets the specification but this is of use only once and not at every run of the program.
Several algorithms have been defined for erasing redundant code (e.g. Beeson [1] and Mohring [9] that use manual techniques to label and remove redundant code, Takayama [10] wich designed an automatic technique that motivated our work). A variant of them are pruning techniques of S. Berardi ([2], [3] and [4]).
Berardi's idea is that, inside the tree representation of an expression, we can find subtrees that are useless for computing the final result. Such parts can be removed in the same way as dead branches in a tree are pruned. Of course the problem is how to find useless subtrees.
In this paper, we show an algorithm to find useless subterms in computations. These subterms can safely be replaced by dummy constants, leading to equivalent terms. The main result of this paper, in particular, is that for each type A and term t of type A, there exists a unique term t' of type A, of minimum length, such that t' is

observational equivalent to t. We'll describe the algorithm that receiving t, gives back this simplest term.
We'll follow the technique used in [4], by defining two order relations '$\leq$', one for types and one for terms. Such relations are an attempt to formalize the previous ideas. $A \leq B$ means that A is simpler than B. In a similar way, if t and u are terms, $t \leq u$ means that t is a simplified version of u. This simpler term has the same input/output behaviour of the original one if the type doesn't change.
In section 2 we introduce some bare notions about the system in which we write our programs. In section 3 we formally define the pruning relation $\leq$. In section 4 we see that simplifying a term doesn't alter its operational meaning. In section 5 we see that, given a term t, there exists a minimum term t' equivalent to t. In section 6 we show an algorithm to find minimum terms. In section 7 we have conclusions and possible developments.

2 The System F_2

We begin this section with the definition of our system. It is essentially Girard's system F (see [7]), extended with constants for the monomorphic natural numbers and special constants unit and Unit.
The intended purpose of Unit is to denote the type with only one element, denoted by unit. Every expression that we consider useless for the computation of the final result will be substituted by these constants.
Now the formal definitions.

Definition 1.
(i) The language of types of F_2 is inductively defined by
$T ::= \text{Unit} \mid \text{Nat} \mid X \mid T \text{ -> } T \mid \forall X.T$
where X denotes a type variable, and Nat the constant for natural number type.
(ii) The language of pseudoterms is inductively defined
$t ::= \text{unit} \mid 0 \mid S \mid \text{rec} \mid x \mid \lambda x{:}T.t \mid (t\ t) \mid \Lambda X.t \mid t[T]$
where 0 and S are the constants for constructing the primitive natural number while rec is the primitive polymorphic constant for the primitive recursion over Nat; x denotes a term variable and $\Lambda X.t$ and $t[T]$ denote type abstraction and type application.
(iii) A context is any finite set of ordered pairs, whose first components are term variables, having pairwise distinct names, and whose second components are types.
(iv) If Γ is a context of the form $\{<x_1{:}T_1>, \dots, <x_n{:}T_n>\}$ $\text{dom}(\Gamma)$ is $\{x_1, \dots, x_n\}$.

As usual, for type system "a la Church", we have rules for well-formedness of terms. A judgement of the form $\Gamma \mid\!- t : T$, means that the pseudoterm t is a well formed term of type T in the context Γ.

Definition 2.
Let $\Sigma = \{<0{:}\text{Nat}>, <S{:}\text{Nat}\rightarrow\text{Nat}>, <\text{rec}{:}\forall X.\text{Nat}\rightarrow X\rightarrow(\text{Nat}\rightarrow X\rightarrow X)\rightarrow X>\}$,
t, t_1 and t_2 be pseudoterms, x a term variable, X a type variable, T, A and B types and Γ a context. The term formation rules are:

(Unit) $\Gamma \mid\!- \text{unit} : \text{Unit}$ (Const) $\Gamma \mid\!- c : A$ (with $<c{:}A> \in \Sigma$)

(Var) $\Gamma \mid\!- x : T$ (if $<x{:}T> \in \Gamma$)

$$(\to \mathrm{I})\ \frac{\Gamma \cup \{<x:A>\} \mid\!\!- t : B}{\Gamma \mid\!\!- \lambda x{:}A.t : A\to B} \qquad (\to \mathrm{E})\ \frac{\Gamma \mid\!\!- t_1 : A\to B \quad \Gamma \mid\!\!- t_2 : A}{\Gamma \mid\!\!- t_1 t_2 : B}$$

$$(\forall\ \mathrm{I})\ \frac{\Gamma \mid\!\!- t : T}{\Gamma \mid\!\!- \Lambda X.t : \forall X.T}\ (X \text{ not free in } \Gamma) \qquad (\forall\ \mathrm{E})\ \frac{\Gamma \mid\!\!- t : \forall X.T}{\Gamma \mid\!\!- t[A] : T[X:=A]}$$

We assume the standard convention for the associativity of operators and for the priorities. We call *atom* any atomic type, constant or variable, and *symbol* any term constant and term variable. We use α, β, γ, ... to denote atoms and s, s', s", ... to denote symbols. Generally, we use upper case for types and lower case for terms. The symbol '$\equiv$' expresses syntactical equality between expressions.
With FV(e) we indicate the free (type and term) variables of e, while with $FV_1(e)$ and $FV_2(e)$ we respectively denote the free type variables and free term variables of e. As usual, an expression is called *closed* if it hasn't free (type and term) variables. A *substitution* σ is a function from variables to expressions of the language such that, if x is a term variable of type A, $\sigma(x)$ is a term of type A and, if X is a type variable, $\sigma(X)$ is a type. We denote substitutions with σ, τ, ... The effect of a substitution over an expression is the replacement of all occurrences of free variables with the corresponding expressions as indicated in the substitution itself. We suppose implicit renaming of bound variables to avoid capture of free variables. A *closed substitution* is a substitution σ such that $\sigma(x)$ is closed for any variable (type and term) $x \in \mathrm{dom}(\sigma)$.
If Γ is a context, a substitution σ is said to be a Γ-substitution, if $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(\sigma)$.
Let e be any expression. We denote $T_2 = \{T \mid T \text{ is a type of } F_2\}$, $T_2^0 = \{T \mid T \text{ is a closed type of } F_2\}$, $\Lambda_2 = \{t \mid t \text{ is a terms of } F_2\}$, $\Lambda_2^0 = \{t \mid t \text{ is a closed terms of } F_2\}$, $E_2 = T_2 \cup \Lambda_2$ and $E_2^0 = T_2^0 \cup \Lambda_2^0$.

The system has β and η rules for terms and types application. We suppose α rules implicit.

Definition 3.
(i) If t,u and f are terms and A a type the elementary reduction rules for F_2 are:

(β)	$(\lambda x{:}A.t)(u)$	$\to_\beta$	$t[x:=u]$
(η)	$\lambda x{:}A.(f\ x)$	$\to_\eta$	f (with x not free in f)
(B)	$(\Lambda X.t)[A]$	$\to_B$	$t[X:=A]$
(H)	$\Lambda X.(f\ [X])$	$\to_H$	f (with X not free in f)

Together with these standard rules there are other ones for the constant rec; if a is a term of type A and f a term of type Nat$\to$A$\to$A then:

(rec0)	rec [A] 0 a f	$\to_{rec0}$	a
(recS)	rec [A] (S n) a f	$\to_{recS}$	f n (rec [A] n a f)

We write $\to_{rec}$ for $\to_{rec0} \cup \to_{recS}$.

(ii) Let $r \in \{\beta, B, \eta, H, rec\}$. We use: $\to_r$ also for the contextual closure of the elementary reduction rule $\to_r$; $\to_1$ for the union $\to_\beta \cup \to_B \cup \to_\eta \cup \to_H \cup \to_{rec}$; $\to_n$ for exactly n steps of $\to_1$; $\to_{\leq n}$ for at most n steps; $\to_{\geq n}$ for at least n steps. $\to_r^*$ is used for any finite number of steps of reduction r (included 0). We use $=_r$ for

conversion w.r.t. reduction r.We indicate $=_\beta \cup =_B$ with $=_{\beta B}$ and $\to_\beta \cup \to_B$ with $\to_{\beta B}$, while $=_F$ denotes conversion respect all reduction rules together.
We assume the standard definitions of redex and normal form.

An important notion for the analysis of programs is observational equality $=_{obs}$. We can say that two programs are equal from an observational point of view if no observation or test is able to distinguish them. Formally we have:

Definition 4.
Let t, u be two terms such that $\Gamma \mid\!- \ t : T$ and $\Gamma \mid\!- \ u : T$. We define:
(i) t and u closed $\Rightarrow$ $\mid\!- t =_{obs} u$ iff ($\forall$closed f s.t. $\mid\!- f : T\to Nat$) $f(t) =_{\beta B} f(u)$
(ii) any t, u $\Rightarrow$ $\Gamma \mid\!- t =_{obs} u$ iff ($\forall\sigma$ closed Γ-substitution) $\mid\!- \sigma(t) =_{obs} \sigma(u)$

We can consider every closed function f:T→Nat as an observation, aiming to discover any difference in the behaviour of t and u. If it is not possible with observations to find a difference between the two terms, we say that they are observationally equivalent.

Definition 5.
Let **T** be a theory on F_2. If t and u are terms:
(i) we write $\Gamma \mid\!- t =_T u$ if $(t,u) \in T$ and $FV_2(t) \cup FV_2(u) \subseteq dom(\Gamma)$;
(ii) with $\Gamma \not\vdash t =_T u$ we mean that $\Gamma \mid\!- t =_T u$ doesn't hold;
(iii) a theory **T** is consistent if there exist two terms t, u of same type and context, such that $\Gamma \not\vdash t =_T u$;
(iv) a theory **T** is a maximum theory if it is consistent and for each consistent theory **T'** and for each pair of term t and u, we have $\Gamma \mid\!- t =_{T'} u \Rightarrow \Gamma \mid\!- t =_T u$.

Lemma 1.
Observational equivalence is a maximum equational theory for F_2.
The proof of lemma 1 was developed by Statman and written in an unpublished manuscript of 1986.

Now some properties of F_2.

Proposition 1.
System F_2 is Church-Rosser and strongly normalizing.
For the proof see e.g. [7].

A term is said ***algebraic*** if it consists only of: constants of kind $f{:}\beta$ or $f{:}\alpha_1\to\ldots\to\alpha_n\to\beta$, where $\beta, \alpha_1, \ldots, \alpha_n$ are atomic types, term variables of the form $x{:}\alpha$, where α is an atom, and applications.

Proposition 2.
Closed algebraic terms of type Nat, in normal form, are only of this shape: $S^k(0)$, for some integer k (if k = 0 $S^k(0)$ is 0).
Proof
Easy by induction.

3 Pruning in F_2

In analogy with [4], we define three relations of $\leq$, two on *well formed expressions* of the system and one over context. The first relation is over well formed types. We write $A \leq_T B$, if A and B are types. The second relation is defined over contexts. We write $\Gamma \leq_{ctx} \Gamma'$ if Γ and Γ' are contexts. The third one is defined on well formed terms. We write $t\ {}_{\Gamma;A}{\leq}_{\Delta;B}\ u$, if t and u are terms s.t. $\Gamma \vdash t : A$ and $\Delta \vdash u : B$.

Definition 6.
Let t, t_1, t_2, u_1, u_2 be any terms, A, B, C, D, T any types, c any costant, x, y, X, Y any term and type variables, Γ and Γ' any contexts. The pruning relations "$\leq$" for types and terms are inductively defined by two deduction systems:
(i) The system $\leq_T$

(Unit) $\mathrm{Unit} \leq_T T$ (Nat) $\mathrm{Nat} \leq_T \mathrm{Nat}$ (VAR) $X \leq_T X$

$$(\rightarrow)\ \frac{A_1 \leq_T A_2 \quad B_1 \leq_T B_2}{A_1 \rightarrow B_1 \leq_T A_2 \rightarrow B_2} \qquad (\forall)\ \frac{A[X:=Z] \leq_T B[Y:=Z]}{\forall X.A \leq_T \forall Y.B}\ (*)$$

(ii) We say that $\Gamma \leq_{ctx} \Gamma'$ iff $(\forall x,A)\ ((\langle x{:}A\rangle \in \Gamma) \Rightarrow (\exists B)\ (\langle x{:}B\rangle \in \Gamma'$ and $A \leq_T B))$.

(iii) The system ${}_{\Gamma;A}{\leq}_{\Delta;B}$

$$(\mathrm{unit})\ \frac{\Gamma \leq_{ctx} \Delta}{\mathrm{unit}\ {}_{\Gamma;\mathrm{Unit}}{\leq}_{\Delta;T}\ t} \qquad (\mathrm{const})\ \frac{\Gamma \leq_{ctx} \Delta}{c\ {}_{\Gamma;A}{\leq}_{\Delta;A}\ c}$$

$$(\mathrm{var})\ \frac{\Gamma \leq_{ctx} \Delta}{x\ {}_{\Gamma;A}{\leq}_{\Delta;B}\ x} \quad \text{where } \langle x{:}A\rangle \in \Gamma \text{ and } \langle x{:}B\rangle \in \Delta$$

$$(\lambda)\ \frac{A \leq_T B \quad t_1[x:=z]\ {}_{\Gamma;C}{\leq}_{\Delta;D}\ t_2[y:=z]}{\lambda x{:}A.t_1\ {}_{\Gamma';A\rightarrow C}{\leq}_{\Delta';B\rightarrow D}\ \lambda y{:}B.t_2}\ (*)$$

where $\Gamma = \Gamma' \cup \{\langle z{:}A\rangle\}$ and $\Delta = \Delta' \cup \{\langle z{:}B\rangle\}$

$$(\mathrm{app})\ \frac{t_1\ {}_{\Gamma;A\rightarrow C}{\leq}_{\Delta;B\rightarrow D}\ t_2 \quad u_1\ {}_{\Gamma;A}{\leq}_{\Delta;B}\ u_2}{t_1u_1\ {}_{\Gamma;C}{\leq}_{\Delta;D}\ t_2u_2}$$

$$(\Lambda)\ \frac{t_1[X:=Z]\ {}_{\Gamma;A}{\leq}_{\Delta;B}\ t_2[Y:=Z]}{\Lambda X.t_1\ {}_{\Gamma;\forall X.A}{\leq}_{\Delta;\forall Y.B}\ \Lambda Y.t_2}\ (*) \qquad (\mathrm{APP})\ \frac{t_1\ {}_{\Gamma;\forall X.A}{\leq}_{\Delta;\forall Y.B}\ t_2 \quad C \leq_T D}{t_1[C]\ {}_{\Gamma;A[X:=C]}{\leq}_{\Delta;B[Y:=D]}\ t_2[D]}$$

(*) In expressions involving binders we have to introduce fresh variables z and Z in order to have the relation invariant up to α conversion. Without this caution we would have for example, $\forall X.X \leq_T \forall X.X$, but not $\forall X.X \leq_T \forall Y.Y$.

To simplify the notation, in the rest of the paper we write only $\leq$, both for $\leq_T$ and for ${}_{\Gamma;A}{\leq}_{\Delta;B}$. Since these two relations are defined on different domains, it will be clear from the context of which relation we are talking about.

The relation $\leq$ is a formalization of the idea of simplification of a term. When an expression or a part of an expression is useless at the aim of the computation of the final result, we can replace it with one of the two special costants unit and Unit. So we have a simpler expression.
To better understand how pruning an expression we state:

Proposition 3.
(i) if A and B are types s.t. $A \leq B$ then
- either $A \equiv B$
- or $A \equiv$ Unit
- or A is obtained from B by replacing some proper subtypes of B with Unit.

(ii) if t_1 and t_2 are terms s.t. $t_1 \leq t_2$ then
- either $t_1 \equiv t_2$
- or $t_1 \equiv$ unit
- or t_1 is obtained from t_2 by replacing some proper subterms of t_2 with unit
- or t_1 is obtained from t_2 by replacing some types B, in type applications, with some types A s.t. $A \leq B$
- or t_1 is obtained from t_2 by replacing the type B of some of its term variables with a type A s.t. $A \leq B$.

All these assertions are up to α conversion.

Remember always that the relation $\leq$ is defined on well formed expression. So if we write $e_1 \leq e_2$ we implicitly assume that e_1 and e_2 are well formed.

Proposition 4.
The pruning relation, $\leq$, is an order relation.

4 Pruning and Observational Equality

In this section we'll see the main theorem of this paper: pruning is compatible with observation equivalence.

Lemma 2.
If t and u are closed terms in normal form of type Nat, such that $t \leq u$, then $t \equiv u$.
Proof
As in [4].

Lemma 3.
Let x a term variable. Let A and A' types s.t. $A \leq A'$. Let t, t', u and u' terms such that $\Gamma \cup \{<x:A>\} \mid\!- t : B$, $\Gamma' \cup \{<x:A'>\} \mid\!- t' : B'$, $\Gamma \mid\!- u : A$, $\Gamma' \mid\!- u' : A'$, for some Γ, Γ', B, B', and $t \leq t'$, $u \leq u'$. Then $t[x:=u] \leq t'[x:=u']$
Proof
By induction on t.

Lemma 4.
Let t, t', u' be terms. If $t \leq t'$ and $t' \rightarrow_n u'$ with only β reduction steps, then $t \rightarrow_{\leq n} u$ for some term $u \leq u'$.
Proof
By induction on n, using the previous lemma and the definition of $\leq$.

Similar properties hold for type substitutions and B reductions.

Lemma 5.
(i) If A, A', T and T' are types and X is a type variable, such that $A \leq A'$ and $T \leq T'$ then $A[X:=T] \leq A'[X:=T']$.
(ii) If t and t' are terms, T and T' are types and X is a type variable, such that $t \leq t'$ and $T \leq T'$ then $t[X:=T] \leq t'[X:=T']$.
(iii) Let t, t' and u' be any terms. If $t \leq t'$ and $t' \rightarrow_n u'$ with only B reduction steps, then $t \rightarrow_{\leq n} u$ for some term $u \leq u'$.

From the previous lemmas the following result follows.

Lemma 6.
Let t, t' and u' be any terms. If $t \leq t'$ and $t' \rightarrow_n u'$, with only β or B reduction steps, then $t \rightarrow_{\leq n} u$ for some term $u \leq u'$.

Now we are ready for the main theorem about the pruning and the observationality.

Theorem 1.
Let t and u terms. If $\Gamma \mid\!- \ t : A$, $\Gamma \mid\!- \ u : A$ and $t \leq u \Rightarrow \Gamma \mid\!- t =_{obs} u$.
Proof
Using the previous lemmas, the proof is a case analysis of possible contexts and types for t and u.

5 The Minimum Pruning of a Term

We have seen that the pruning relation is an order relation. We can show that for each set of expressions of F_2 there exists the greatest lower bound w.r.t. $\leq$.

Definition 7.
We inductively define the function inf : $E_2 \times E_2 \rightarrow E_2$
(i) for each pair of types,
(ii) for each pair of terms

(i) Let T_1, T_2, A, A_1, A_2, B, B_1 and B_2 be any types, α any atom, X, Y any type variables. We have:
-$\inf(\alpha,\alpha) = \alpha$
-$\inf(A_1 \rightarrow B_1, A_2 \rightarrow B_2) = \inf(A_1,A_2) \rightarrow \inf(B_1,B_2)$
-$\inf(\forall X.T_1, \forall Y.T_2) = \forall Z.\inf(T_1[X:=Z], T_2[Y:=Z])$ (Z fresh type variable)
else
-$\inf(A,B) = \text{Unit}$
(ii) Let t , t', t_1, t_2, u, u', u_1 and u_2 be any terms, c any constant, X, Y any type variables, A, B any types, x, y any term variables then:
-$\inf(c,c) = c$
-$\inf(x,x) = x$
-$\inf(\lambda x{:}A.t', \lambda y{:}B.u') = \lambda z{:}\inf(A,B).\inf(t'[x:=z], u'[y:=z])$ (where z is a fresh term variable)
-$\inf(t_1 t_2, u_1 u_2) = \inf(t_1,u_1)\inf(t_2,u_2)$
-$\inf(\Lambda X.t', \Lambda Y.u') = \Lambda Z.\inf(t'[X:=Z], u'[Y:=Z])$ (where Z is a fresh type variable)
-$\inf(t'[A], u'[B]) = \inf(t',u')[\inf(A,B)]$

else
-inf(t,u) = unit

All these assertions are up to α renaming.

Extending these definitions to contexts we have:
Definition 8.
For each pair of contexts exists the $\inf_{cxt}$ and, if Γ and Δ are contexts, we define
$\inf_{ctx}(\Gamma,\Delta) = \{<x:C> \mid <x:A> \in \Gamma, <x:B> \in \Delta \text{ and } C = \inf(A,B)\}$

Now we can state, without proof
Lemma 7.
(i) If e_1 and e_2 are both types or both terms and Γ_1 and Γ_2 are contexts then $\inf(e_1,e_2)$ is the g.l.b. of e_1 and e_2 w.r.t. $\leq$ while $\inf_{ctx}(\Gamma_1,\Gamma_2)$ is the g.l.b. of Γ_1 and Γ_2 w.r.t. $\leq_{ctx}$.
(ii) The sets T_2 and Λ_2 are lower semilattices w.r.t. $\leq$.

As consequence, we have the theorem which puts in relation deduction with pruning.

Theorem 2.
If t and u are terms s.t. Γ |- t : A and Δ |- u : B and there exists a term z s.t. $t \leq z$ and $u \leq z$, then
$\inf_{cxt}(\Gamma,\Delta)$ |- inf(t,u) : inf(A,B)

Now we introduce two structures useful for the optimization algorithm that we'll define in the next section.

Definition 9.
For each term t, s. t. Γ |- t : T we define:
(i) LE(t) = $\{t' \mid t' \leq t\}$
(ii) CLE(t) = $\{t' \mid t' \leq t \text{ and } \Gamma' \vdash t' : T \text{ and } \Gamma' \subseteq \Gamma\}$

If t is a term, the set LE(t) (*less equal* t) is the set of all the terms that we can obtain from t replacing some parts by the special constants, while the set CLE(t) (*contestual less equal*) is the set of simplified version of t with same type and context and so equivalent to t itself.

Proposition 5.
Given any term t we have:
(i) LE(t) is a finite complete lower semilattice w.r.t. $\leq$
(ii) CLE(t) is a sub semilattice of LE(t)
Proof
(i) A simple consequence of lemma 7.
(ii) Easy, generalizing theorem 2 and remembering that every term t' $\in$ CLE(t) has the same type of t and has a context included in Γ.

Definition 10.
Let t any term. We denote the minimum element of CLE(t) as FL_2(t).

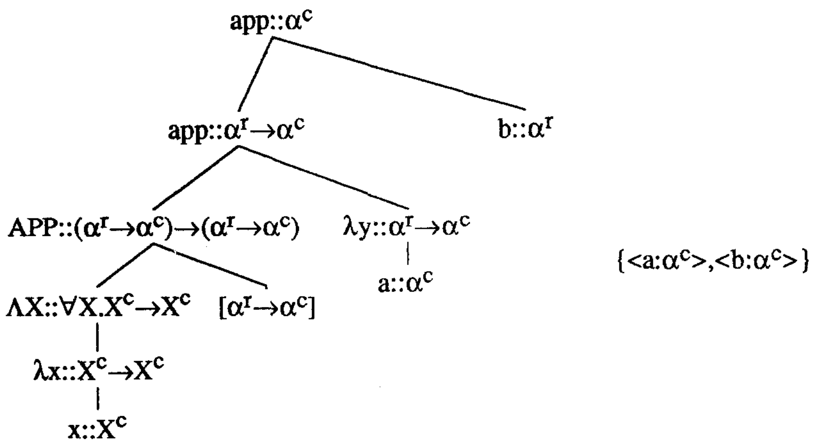
Proposition 6.
$Fl_2(t) =_{obs} t$.
Proof
From theorem 2, since $Fl_2(t) \leq t$.

6 An Algorithm for Finding Minimum Prunings

In this section we show an algorithm to find the term $FL_2(t)$, defined in last section. In order to compute Fl_2 we use the technique of C. Mohring based on marking of types (see [9]). A mark is a label that we put over an atomic type, constant or variable. Marks are of two kind: 'r' and 'c'. The former means that a term or subterm whose type has such mark is redundant, i.e. useless for the computation of final result. Instead the latter mark means that the expression *may be* useful for the computation of the result. When an expression is entirely marked we can remove the parts marked 'r'. We let parts marked 'c' while we replace redundant types by Unit and the corresponding (redundant) terms by unit. So there is a strong correspondence between Mohring's manual technique of type labelling and our extended syntax with constants denoting useless expression. The marking technique will be used to compute a 'minimum' marking that corresponds to a 'minimum' term (with respect to the pruning relation). The underlying computational structure is the abstract syntax tree of a term. We use the *fully decorated tree*, in which associated with every node there is the type of the subterm individuated by this node. Now, we introduce some definitions and terminology about trees and markings.

Definition 11.
Let t be a term such that $\Gamma \mid\!- t : T$
(i) The fully decorated tree of t, FDT(t), is the syntax tree of t in which each node that identifies a subterm is decorated by the type of the subterm itself.
(ii) The fully decorated version of $\Gamma \mid\!- t : T$, $FDV(\Gamma \mid\!- t : T)$, is defined as the ordered pair $<\Gamma, FDT(t)>$, formed by the context Γ and the fully decorated tree of t.
(iii) $Atoms(\Gamma,t)$, the set of occurrences of atoms of $FDV(\Gamma \mid\!- t : T)$, is formed by every occurrence of atom in the types of the variables in the context Γ with every occurrence of atom in the types that decorate the nodes of FDT(t) and the occurrences of atoms of applied types in type applications of t.
(iv) Let Lab be the set formed by the two labels 'r' and 'c', namely Lab = {'r', 'c'}. A marking M of $FDV(\Gamma \mid\!- t : T)$ is a map from $Atoms(\Gamma,t)$ to Lab. When we say a marking M of t, we mean the restriction of the map M to FDT(t). Note that we can identify a marking of a FDV with the set $C \subseteq Atoms(\Gamma,t)$ of occurrences of atoms that are marked by 'c'.

We show these concepts with an example
Example.
Let $term_1 \equiv ((\Lambda X.\lambda x{:}X.x)[\alpha\rightarrow\alpha](\lambda y{:}\alpha.a))b$ with α an atom, a and b free term variables of type α. We can deduce $\{<a{:}\alpha>, <b{:}\alpha>\} \mid\!- term_1 : \alpha$. The FDV for this judgement, with an example of marking for it, is:



We call this marking M_1.

Not all the markings are useful or meaningful for the pruning. We single out some kinds of markings for their use in optimization.

Definition 12.
Let t be a term. A marking M of t 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 is completely redundant, i.e. all of its atoms are marked with label 'r'.

Now we define the map that gives the correspondence between marking and pruning.

Definition 13.
Given a fully decorated syntax tree for a term t and a canonical marking M for it, we denote Simplify(M,t) the term obtained replacing every maximum subtype in t, totally marked by 'r', with Unit and every maximum subterm of t, whose type is totally marked by 'r', with unit.

Now, we can define others kinds of markings.

Definition 14.
Let t be a term s.t. $\Gamma \mid\!- t : T$,
(i) A marking M on t is *consistent* iff Simplify(M,t) is a well formed term w.r.t. some context $\Gamma' \leq_{ctx} \Gamma$.
(ii)A marking M is *saturated* iff it is consistent and canonical.

Observe that, to have consistency in a marking of t, any two atoms matched during the typechecking of t have to be marked in the same way.
As already said, there is a strong correspondence between marking and pruning. This correspondence is expressed by the map Simplify. It is an isomorphism from saturated markings and the expressions of the system in which: 1) there are no non-atomic types whose atoms are just Unit; 2) the only term of type Unit is unit. With this choices, we can prove that if t is a term, for each saturated marking M there exists one and only one pruned term $t' \leq t$ such that Simplify(M,t) = t'.

For example the marking M_1 is saturated. Applying Simplify to it, we have Simplify(M_1,term$_1$) ≡ ((ΛX.λx:X.x)[Unit→α](λy:Unit.a))unit that is also the minimal element of CLE(term$_1$), namely FL$_2$(term$_1$).

We can define also an order on markings.

Definition 15.
Let t be a term s.t. Γ |- t : T. Let M and M' be markings for the FDV(Γ |- t : T). We say:
M ≤ M' iff each 'c' assigned by M is also assigned by M'.

We can prove that the two ordering correspond to each other. If t is a term and M, M' markings over t, then M ≤ M' iff Simplify(M,t) ≤ Simplify(M',t).
Obviously, the minimum marking has all 'r'. In this way, the whole term is replaced by unit and we obtain a term which is not equivalent to the original. We look for a term with same type and same context. So we use an *initial marking* M_0 that brought with it these conditions.

Definition 16.
Given a FDV(Γ |- t : T), the initial marking M_0 for it, is the map that assigns 'r' to every atom in Atoms(Γ,t), excluding: 1) the atoms in the type associated to the root, namely *the type of the term*; 2) the atoms in the type of variables inside the context.

This initial marking is canonical but inconsistent. The aim of our algorithm is, to find a minimum consistent marking M' that is greater (respect to ≤) than the initial marking M_0. This operation is called *saturation* of a marking. In this section we call *binder* both λx:A and ΛX in term and pair <x:A> in context.

Now, we can write our optimization algorithm.
Definition 17.
Optimization Algorithm
Input: A term t , a context Γ and a type T such that Γ |- t : T.
Output: FL$_2$(t).
Perform in sequence the following steps:
- Build the FDV(Γ |- t : T).
- Build the initial marking M_0.
- Saturate M_0, obtaining M'.
- Apply Simplify(M',t).

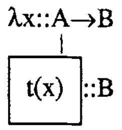
The interesting part of the algorithm is then in the saturation procedure. It consists of two parts. In the first part we build a structure, the adiacence graph that is used to connect the atoms that have to be marked with the same label. So the edges of the graph are a different way to describe consistency condition. If two objects are linked, it means that they have been matched during the type checking procedure. In the second part, the labels 'c', starting from the type T of the term, flow along the paths of the adiacence graph. When this flow is completed the algorithm stops and the resulting marking is consistent.

Definition 18.
Saturation Algorithm
Input: A FDV(Γ |- t : T) and M_0 for it.
Output: The minimum saturated marking M' s.t. $M_0 \leq M'$
- Build the adiacence graph of FDV(Γ |- t : T).
- Propagate the labels to achieve consistency.

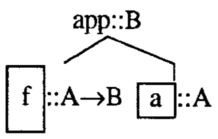
Now we see the algorithm for the construction of the adiacence graph. In this step, we build the structure used for obtaining the consistency in the marking.

Definition 19.
(i) Given a FDV(Γ |- t : T), we call adiacence graph, AG(Γ |- t : T), the graph so defined:
1) The set Nod of nodes is formed by hte union of:
   a) Atoms(Γ,t)
   b) the set of occurrences of term variables of FDT(t)
   c) the set of binders in the FDT(t) and in the context
   d) the set of all types of FDT(t)
2) The set Edg of edges is formed by four kinds of edges connecting different kinds of nodes:
   a) *normal edges* connect pairs of atoms in types
   b) *bind edges* connect free variables with their binders in the context and bound variables with their binders in term
   c) *broadcast edges* link type variables with types
   d) *bind broadcast edges* link types with types in square brackets.
3) Every part of the FDV(Γ |- t : T), matched during the typechecking, is linked by a related edge.
(ii) Given a FDV(Γ |- t : T), its augmented tree, T^{edge}(Γ |- t : T), is obtained from the FDV adding the AG(Γ |- t : T) on it.

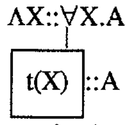
Definition 20.
Adiacence Graph Construction Algorithm
Input: A FDV(Γ |- t : T) and M_0 for it.
Output: The T^{edge}(Γ |- t : T) of FDV.
Perform the following steps:
   1) Link,with a bind edge, every free term variable to its binder in the context Γ
   2) For each λ-abstraction node



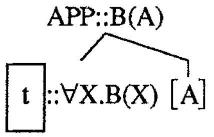
   of the tree, link corresponding atoms in the two occurrences of B with normal edges and link every occurrence of the bound variable x in t to its binder λx with a bind edge.
   3) For each term application node



of the tree, link corresponding atoms in the two occurrences of A and in the two occurrences of B with normal edges.
4) For each Λ-abstraction node



of the tree, link corresponding atoms in the two occurrences of A with normal edges.
5) For each type application node



of the tree, link every occurrence of X in B(X) to the corresponding occurrences of A in B(A) with a broadcast edge. Link corresponding atoms in the two occurrences of B, that are not X or atoms of A, with normal edges. Link every occurrence of A in B(A) to the occurrence of A in [A] with a bind broadcast edge.

Now the second step. With this structure, the consistency of a marking is equivalent to the following conditions:
(1) for each pair of atoms connected with a normal edge they are marked in the same way;
(2) for each free or bound term variable x:A in t, if A has at least one mark 'c' and $<x:A> \in \Gamma$ or $\lambda x:A$ is the binder of x, corresponding atoms in the two occurrences of A are marked in the same way;
(3) for each type variable connected to a type by a broadcast edge, either the variable and the atoms of the type are marked 'r' or the variable and at least one atom of the type are marked 'c';
(4) for each occurrence of type A, connected to a type A in square brackets, by a bind broadcast edge, if its marking contains at least one 'c' then corresponding atoms in the two occurrences of A are marked in the same way.
There is still something to say. In our system the type of the constants cannot be simplified. So there is only a term strictly less then a constant, namely unit. This imply that either a constant is removed or is totally used. So we have to add a fifth case to the procedure, for the treatment of constants.

Definition 21.
Marking Propagation Algorithm
Input: The $T^{edge}(\Gamma \mid\!- t : T)$ of FDV and the initial marking M_0.
Output: The minimum saturated marking M' s.t. $M_0 \leq M'$.
Repeat one of the following steps **until** no more step can be executed:
(1) Take any normal edge such that at least one atom is marked 'c'. Then mark the other atom 'c' and remove the edge;
(2) or take any bind edge such that the free or bound variable has at least one 'c' in the marking of its type. Let x:A this variable and $<x:A> \in \Gamma$ or $\lambda x:A$ its binder. Then remove the bind edge and link with one normal edge each corresponding atom in the two occurrences of A;
(3) or take a broadcast edge such that either the type variable is marked 'c' or the type has an atom marked 'c'. Then, in the first case, mark last atom of the type

with 'c'. In the second case mark the type variable with 'c'. In both cases remove the edge;
(4) or take a bind broadcast edge such that the type not inside the square brackets has at least one 'c' in its marking. Let A this type. Link each corresponding atom in the two occurrences of A with a normal edge. Remove the bind broadcast edge;
(5) or take a constant c:A such that at least one of its labels is 'c'. Then label with 'c' each atom of A.

This part ends the description of our optimization algorithm. We can prove that saturation algorithm always stops with the minimum consistent marking, namely the one with less 'c' in it, s.t. the type and the context are totally marked with 'c'. Owing to the correspondence between pruning and marking, this leds to the minimal pruning among the ones which respect the type and the context of the original term.

Theorem 3.
Let t be a term s.t. $\Gamma \ |\text{-}\ t : T$. The saturation algorithm computes the saturation of M_0, i.e. the minimal marking M' s.t. $M_0 \leq M'$ and M' is saturated.

Theorem 4.
Let t be any term. Simplify is an isomorphism between saturated markings of t and the set of terms $t' \leq t$, whose sintax respects the rules: 1) don't exist non-atomic types whose atoms are just Unit; 2) the only term of type Unit is unit.

As consequence of the two previous theorems, we have the correctness of our optimization algorithm.

Theorem 5.
Given a term t such that $\Gamma \ |\text{-}\ t : T$ for some type T and context Γ, the optimization algorithm computes $Fl_2(t)$, i.e. the minimal term $t' \leq t$ whose type is T and whose context if $FV_2(t)$.

7 Conclusions and Future Works

In this paper, we have described a first step toward a simplification of expressions in F_2. Every time we find a subexpression useless for the final result, we replace it by a unit. New expressions are shorter but may contain a lot of instances of constant unit. We are developing another step, i.e. the elimination of these instances. In this way we save more time and space during the evaluation of the expressions.
In future, we want also introduce the notion of Harrop type in our system. Informally, a type H is Harrop if all terms of such type are equivalent. So, for each Harrop type H, we can define a canonical constant c_H and replace each term t of type H by this constant.
Another idea is to introduce Kinds in our system. Kinds may be useful for pruning type variables. If a λ-binder in a term binds a term variable x of type A never used in the body of the abstraction, we can prune λx:A to λx:Unit. For a Λ-binder that binds a type variable never used, we can do nothing. Introducing a special Kind UNIT, we can mark this useless abstraction, using this new constant, and in future remove such an abstraction.
Finally, we have to remember that, our techniques are useful above all for automatically generated programs. In such programs there may appear large parts of

redundant code. Normally, for code generated by hand, the best optimizer is the programmer's head.

Acknowledgements

I want to thank my supervisor M. Coppo for his help in recent years and for his valuable remarks and suggestions about this paper. Above all I thank S. Berardi since without his foundational work, his encouragement and discussions about pruning this paper would never have been written.

Bibliography

[1] M. Beeson, *Foundations of Constructive Mathematics*, Berlin, Springer-Verlag, 1985

[2] S. Berardi, *Extensional Equality for Simply Typed λ-calculi*, Technical Report, Turin University, 1993.

[3] S. Berardi, *A canonical Projection between Simply Typed λ-calculi*, Technical Report, Turin University, 1993.

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

[5] R. L. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System*, Prentice-Hall, 1986.

[6] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Pauling-Mohring, B. Werner, *The Coq Proof Assistant- User's Guide*, INRIA - Rocquencourt, 1983.

[7] J.-Y. Girard, *Interpretation Fonctionelle et Elimination des Coupures de l'Arithmetique d'Ordre Superieur*, These de Doctorat d'Etat, Soutenue le 26 Juin 1972.

[8] W.A. Howard, *The Formulae-as-Types notion of Construction*, in 'Essays on Combinatory Logic, Lambda Calculus and Formlism', Eds J. P. Seldin and j. R. Hindley, Accademic Press, 1980.

[9] C. Paulin-Mohring, *Extracting F_ω's Programs from Proofs in the Calculus of Constructions*, In: Association for Computing Machinery, editor, Sixteenth Annual ACM Symposium on Priciples of Programming Languages, 1989.

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

[11] A. S. Troelstra, *Mathematical Investigation of Intuitionistic Arithmetic and Analysis*, Lecture Notes in Mathematics, 344, Springer-Verlag, 1973