A formalization of Static Analyses in System F

Frederic Prost

Laboratoire de l'Informatique du Parallélisme

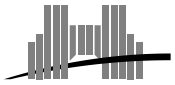
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON
n° 8512

A formalization of Static Analyses in System F

Frédéric PROST

January 99

Research Report N° 1999-07



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

A formalization of Static Analyses in System F

Frédéric PROST

January 99

Abstract

In this paper, we propose a common theoretical framework for type based static functional analyses. The aim is to study the relationships between typing and program analysis.
We present a variant of Girard's System F called $F^{\Pi}_{\leq:}$. We prove some basic properties of $F^{\Pi}_{\leq:}$ such as strong normalization, Church-Rosser property, subject reduction etc. We show how $F^{\Pi}_{\leq:}$ can be used to formalize various program analyses like binding time and dead code, and to encompass previous analyses both in expressivness (often only simply typed calculi are considered) and power (more information can be found on some programs).
$F^{\Pi}_{\leq:}$ features polymorphism as well as subtyping *at the level* of universe extending a previous authors work where only universe polymorphism (on a simply typed calculus). was considered

Keywords: Type Theory, Pruning, Static Analysis, Polymorphism, Subtyping

Résumé

Dans ce papier,nous proposons un cadre théorique générique pour l'analyse statique de programmes fonctionnels. Le but poursuivi est l'étude des relations entre typage et analyse statique de programmes.
Nous présentons une variante du système F de Girard : $F^{\Pi}_{\leq:}$. Nous montrons que ce système satisfait les propriétés standards de normalisation forte, de confluence et d'auto-réduction. Nous utilisons ensuite ce système pour formaliser différentes analyses statiques (comme l'analyse des temps de liaison ou bien la recherche de code mort). Nous montrons aussi que ce système permet de simuler de précédentes analyses de la littérature, et même de les dépasser à la fois en expressivité (souvent ne sont considérés que des calculs simplements typés) et en puissance (plus d'informations peuvent être déterminées sur un programme donné).
$F^{\Pi}_{\leq:}$ est caractérisé par la présence simultanée de sous-typage et de polymorphisme *au niveau* des univers. Ce travail étend de précédents travaux de l'auteur où seul le polymorphisme (de plus sur un calcul simplement typé) était considéré.

Mots-clés: Théorie des types, Analyse statique, Polymorphisme, Sous-typage

A formalization of Static Analyses in System *F*

Frédéric Prost

LIP, Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon
Cedex 07 (France)
email: Frederic.Prost@ens-lyon.fr

Abstract In this paper, we propose a common theoretical framework for type based static functional analyses. The aim is the study of relationships between typing and program analysis.
We present a variant of Girard's System F called $F^{\Pi}_{\leq:}$. We prove basic properties of $F^{\Pi}_{\leq:}$ such as strong normalization, Church-Rosser, subject reduction etc. We show how $F^{\Pi}_{\leq:}$ can be used to formalize various program analyses like binding time and dead code, and to encompass previous analyses both in expressivness (often only simply typed calculi are considered) and power (more information can be inferred).
$F^{\Pi}_{\leq:}$ features polymorphism as well as subtyping *at the level* of universe extending previous author work where only universe polymorphism (on a simply typed calculus) was considered

1 Introduction

The aim of this paper is to provide a theoretical framework for static typed functional analyses. Static functional analyses such as dead code or binding time are important to perform valuable program optimizations. Many type inference based systems have been proposed for those analyses. This work is an attempt to provide a uniform approach for type inference based systems.

1.1 Static functional program analysis

Two approaches are used for static program analysis in the functional programming world: semantic based approach, via abstract interpretation (see [CC77, Hun91]), and inference based approach. In this paper, we focus on inference based approach. It has become more and more popular during the early '90s. Many annotated type systems have been proposed, see for instance [Hei95, Sol95, NSN94, HM94, DP98, TJ92], for many analyses like Binding Time, Strictness, Dead Code, Control Flow etc. Common features can be found in all these systems. A typed programming language is given. The type system is then modified by the addition of annotations on types or term constructors. Those annotations denote semantical properties of the programs (for a survey see the introduction of [Sol95]). The general picture of this approach is as follows: the programmer writes a typed program, then his program is automatically re-typed

under a slightly different type system which includes annotations. In this setting, typing information is meant to be useful for the analysis.

Curry-Howard isomorphism can be used to build programs from proof of their specifications. Nevertheless, code produced this way contain a lot of useless parts from an algorithmic point of view. Indeed, a program just contain what is worthy to implement an algorithm. On the contrary a proof formalizes a great deal of information which is not needed to compute the final result. For instance a proof of the Euclidean division gives for any integers a, b a couple q, r which verify $a = bq + r$ and $r \leq b - 1$. From a computational point of view only q, r are valuable, their properties might be seen as dead code.

The work of C. Paulin (see [Pau89, PM89]) might be seen as a forerunner of type based systems for this kind of dead code analysis. In this work a system to extract F_ω programs from Calculus of Constructions (CC for short) proofs is developed. From a programming language point of view, there is no difference between CC and F_ω. Dependent types of CC can only be used to reason about program properties. Therefore, a syntactical difference between F_ω and CC parts of a term is introduced. It is done by the duplication of the calculus: one part, typed on universe *Prop* is used to denote purely logical parts while the other one typed on universe *Spec* denotes computational parts of the term (parts which may be typed on F_ω). The *extraction* process consists in the erasure of parts typed on *Prop*. It can also be seen as dead code removal.

A major drawback of typed systems lies in their lack of flexibility. For instance consider:

$$p = (\lambda x^{\mathcal{N}}.(+ \;\; (\lambda y^{\mathcal{N}}.5 \;\; x) \;\; x) \;\; 4)$$

One may like to prove that the first occurrence of x is dead, hence typed on *Prop* (if we adopt [PM89] conventions). The problem comes from the second occurrence of x which is alive and hence should be typed on *Spec*. Now, since a term must have a single type and since the analysis must be conservative, x has to be typed on *Spec*. Eventually, in order to keep type consistency, the first occurrence of x cannot be proved dead. A method to overcome this problem, following [Ber93], is to introduce a dummy term $\emptyset$, which is the single inhabitant of a dummy type $\mathcal{U}$ ($\mathcal{N}$ might be seen has the type of naturals built over *Spec* and $\mathcal{U}$ the type of naturals built over *Prop*). The term:

$$p' = (\lambda x^{\mathcal{N}}.(+ \;\; (\lambda y^{\mathcal{U}}.5 \;\; \emptyset) \;\; x) \;\; 4)$$

is well typed. p' can be proved equivalent to p (see [Ber93]), and p' is an improved version of p since it contains less code.

Limitations of type inference based systems come from redexes: $(\lambda f.t \;\; t')$. Variable f must be of the same type that t', but may be used in different situations in t, and each of this situation . Consider for instance:

$$p_1 = (\lambda f^{\mathcal{N}\to\mathcal{N}}.(g \;\; f \;\; (f \;\; 4)) \;\; \lambda x^{\mathcal{N}}.3)$$

where g is variable of type $(\mathcal{N} \to \mathcal{N}) \to (\mathcal{N} \to \mathcal{N})$. Dead code analysis shows that f is a constant function, therefore that 4 is dead. Now, following [Ber93], if

we replaced dead code by the dummy constant, we would obtain the optimized program:

$$p'_1 = (\lambda f^{\mathcal{U}\to\mathcal{N}}.(g \;\; f \;\; (f \;\; \emptyset)) \;\; \lambda x^{\mathcal{U}}.3)$$

p'_1 is ill typed since f is the argument of g and so should be of type $\mathcal{N} \to \mathcal{N}$ while f is of type $\mathcal{U} \to \mathcal{N}$.
To overcome these limitations several paths have been explored. In particular, subtyping (see. [BB95]), ML polymorphism (see [DHM95, Pro97]), conjunctive types (see [DP98]) have been tried out to relax constraints imposed by inference based systems.

1.2 Multi universe system

We take as programming language Girard's System F, later just called F. This choice is relevant from our theoretical point of view since via impredicative encoding, there is no need to define constants. Naturals, booleans, product types etc. can be built inner the system.

We modify F by the introduction of two different universes from which types may be built. Following C. Paulin, those two universes may be seen as *Prop* and *Spec*. In the following of the paper we will rather use the notation $\bot$ and $\top$ for *Prop* and *Spec*, since our use of those two universes is rather different from the one of [Pau89]. The originality of our system lies on two points:

1. We define an inclusion relation between the two universes, namely $\bot{\leq:}\top$, from which we derive a subtyping relation on types.
2. We introduce a notion of *universe variable*, which is a refinement of [Pro97] properties variables. We develop a polymorphism à la ML on universes.

A key point that the reader should keep in mind is that $F^{\Pi}_{\leq:}$ has two different kinds of polymorphism: one is on types (just as in F), the other one is on universes. To enlighten this notion consider for instance $\mathcal{N} = \Pi X.(X \to X) \to X \to X$, the impredicative encoding of natural type. In $F^{\Pi}_{\leq:}$ a type variable X represents all types *of a given universe*. We would encode the type of naturals this way: $\mathcal{N}^u = \Pi X : u.(X \to X) \to X \to X$, where u is an universe. As we have already said universes may be $\bot$, $\top$ or a universe variable α. Therefore, there is naturally a universe variable binder. In $F^{\Pi}_{\leq:}$ it is $\forall$. The scheme $\forall\alpha.\Pi X : \alpha.(X \to X) \to X \to X$, denotes a type which may be instantiated either to $\mathcal{N}^{\top}$ or to $\mathcal{N}^{\bot}$.
It seems sensible that, due to subtyping, there may be constraints between universe variables. Consider the standard subtyping rule:

$$\frac{t : A_1 \to A_2 \quad t' : A'_1 \quad A'_1{\leq:}A_1}{(t \;\; t') : A_2}$$

If we simply abstract over universe variables that occurs in A_1, A_2, A'_1, we "forget" that $A'_1{\leq:}A_1$. This oversight can lead to type inconsistency if the type scheme is badly instantiated. Therefore, to overcome this problem, we introduce a notion of guarded type scheme: $\forall\alpha.C \Rightarrow (A)$ might be seen as the type A where universe α is abstracted and such that any valid instantiation must verify the constraint set C.

1.3 Overview

The type system $F^{\Pi}_{\leq:}$, and its basic properties, are given in section 2. In section 3, we informally discuss the semantics behind $\top$ and $\bot$ universes. Then, in section 4 we see how $F^{\Pi}_{\leq:}$ may be used to formalize program analyses such as dead code and binding time. Finally in section 5 we relate our work to previous ones and conclude.

2 $F^{\Pi}_{\leq:}$ system

In this section we present a system to reason about type inference based systems for static analyses. It is a multi universe F. The different universes will be used to denote different semantical properties.

2.1 Syntact

We start by defining the syntactic categories of $F^{\Pi}_{\leq:}$:

Universe variables: Universe variables are

$$\alpha, \beta$$

elements of an infinite set of universe variables $\mathcal{V}$. **Universes:** The set $\mathcal{U}$ of $F^{\Pi}_{\leq:}$ universes is given by the grammar

$$u ::= \top \mid \bot \mid \alpha$$

We define the relation $\leq:$ between universes. $u_1 \leq: u_2$ always hold except when $u_1 = \top$ and $u_2 = \bot$.

Type variables: Type variables noted

$$X, Y, Z$$

are elements of an infinite set of type variable $\mathcal{T}$.

Types: The pre-types of $F^{\Pi}_{\leq:}$ are given by

$$A, B ::= X \mid A \to B \mid \Pi X : u.A$$

Guarded Type Scheme: Guarded Type schemes are given by

$$\sigma ::= A \mid \forall \alpha.C \Rightarrow (\sigma)$$

where C is a set $\{u_1 \leq: u_2; \ldots; u_n \leq: u_{n+1}\}$ of inequalities on $\mathcal{U}$. By extension we say that C holds if for each i in $\{1 \ldots n\}$, $u_i \leq: u_{i+1}$ holds. Since quantifiers and constraint sets may only occur at the top level of guarded type schemes, we do not distinguish between $\sigma = \forall \alpha.C \Rightarrow (\forall \beta.C' \Rightarrow (A))$ and $\sigma' = \forall \beta.C' \Rightarrow (\forall \alpha.C \Rightarrow (A))$. Thus, for σ and σ' we use the only notation

$$\forall \alpha, \beta.C \cup C' \Rightarrow (A).$$

Terms: The pre-terms of $F^{\Pi}_{\leq:}$ are given by

$$t, t' ::= x\langle S\rangle \mid (t\ \ t') \mid (t\ \ A) \mid \lambda x : A.t \mid \lambda X : u.t \mid \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t'$$

where $\mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t'$ might be seen as a synonym for the term $(\lambda x : A.t\ \ t)$, and S is a substitution from universe variables towards $\mathcal{U}$ elements.

Substitutions: $F^{\Pi}_{\leq:}$ substitutions are finite mappings from universe variables to universes:

$$S ::= [\alpha_1 \mapsto u_1, \ldots, \alpha_n \mapsto u_n]$$

where α_i are pairwise distinct. $D(S)$ is the domain of S; it is the set of α_i. $Im(S)$ is the image of S; it is the set of u_i. A substitution S extends naturally to types and terms as follows:

$$[\ldots; \alpha \mapsto u; \ldots](\alpha) = u$$

$$S(\Pi X : u.A) = \Pi X : S(u).S(A)$$

$$S(X) = X \qquad S(A \to B) = S(A) \to S(B)$$

$$S(x\langle S'\rangle) = x\langle S' \circ S\rangle \qquad S((t\ \ t')) = (S(t)\ \ S(t'))$$

$$S(\lambda x : A.t) = \lambda x : S(A).S(t) \quad S(\lambda X : u.t) = \lambda X : S(u).S(t)$$

$$S((t\ \ A)) = (S(t)\ \ S(A))$$

$$S(\mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t') = \mathsf{let}\ x : S(A)\ \mathsf{be}\ S(t)\ \mathsf{in}\ S(t')$$

A substitution S preserves a constraint set $C = \{u_1 \leq: u_2; \ldots; u_n \leq: u_{n+1}\}$ if $S(C) = \{S(u_1) \leq: S(u_2); \ldots; S(u_n) \leq: S(u_{n+1})\}$ holds.

Contexts: A *context* is an ordered sequence given by

$$\Gamma ::= \emptyset \mid \Gamma, x : A \mid \Gamma, X : u$$

Judgments: We define three judgments:

$$J ::= \Gamma \vdash \mid \Gamma \vdash A : u \mid \Gamma \vdash t : A$$

The first has to be read "Γ is a well formed context", the second "Under the context Γ, the type A is of universe u" and the third "under the context Γ, the term t is of type A".

Typing rules: Typing rules are lists of judgment given by

$$TR ::= \frac{J_1 \ldots J_n}{J_{n+1}}$$

where $J_1 \ldots J_n$ are premises and J_{n+1} the conclusion.

Derivation tree: are trees of typing rules. They are given by

$$\Xi ::= TR \mid \frac{\Xi_1 \dots \Xi_n}{TR}$$

where premises $J_1 \dots J_n$ of TR are equal to $\Xi_1 \dots \Xi_n$ conclusions. We write: $\Xi \rhd \Gamma \vdash t : A$, to mean that Ξ is a derivation tree having the conclusion $\Gamma \vdash t : A$.

FV is the set of free type variables and FUV is the set of free universe variables. For instance, $FUV(\forall\beta,\gamma.C \Rightarrow (\Pi X : \alpha.X \to Y \to X \to Z)) = \alpha$ and $FV(\forall\beta,\gamma.C \Rightarrow (\Pi X : \alpha.X \to Y \to X \to Z)) = \{Y, Z\}$.
$Univ(A)$ (resp. $Univ(t)$) denotes the set of universes occurring in A (resp. t). For instance $Univ(\lambda X : u.t) = \{u\} \cup Univ(t)$.
If t is a term, by $t[t_1; \dots; t_n]$, we mean that t_i are disjoint occurrences of subterms of t. $t[t'_1; \dots; t'_n]$ denotes the literal replacement of $t_1, \dots, t_n$ by $t'_1, \dots, t'_n$.

Following ML type scheme instantiation and generic instantiation (see [CDDK86] for instance), we define two relations between guarded type schemes.

Definition 2.1 (instantiation)
A guarded type scheme σ' is called an instance of a guarded type scheme σ if there exists a substitution S for free universe variables such that:

$$\sigma' = S(\sigma)$$

Definition 2.2 (Generic instantiation)
A guarded type scheme $\sigma' = \forall\beta_1, \dots, \beta_m.C' \Rightarrow (A'_0)$ is called a generic instance *of a guarded type scheme $\sigma = \forall\alpha_1, \dots, \alpha_n.C \Rightarrow (A_0)$ if there exists a substitution S such that:*

1. *$D(S) \subseteq \{\alpha_1, \dots, \alpha_n\}$,*
2. *$S(C) = C'$ and S preserves C,*
3. *$S(A) = A'$.*

If $\sigma' = A'$ (hence $m = 0$), we write $\sigma \overset{S}{\rightsquigarrow} A'$.

$F^{\Pi}_{\leq:}$ is closely related to F. The main difference between F and $F^{\Pi}_{\leq:}$ is that F types are built over a single universe. There is a natural surjection $|.|$ from $F^{\Pi}_{\leq:}$ to F. It is inductively defined as follows:

- **Types:** $|X| = X$, $|A \to A'| = |A| \to |A'|$, and $|\Pi X : u.A| = \Pi X.|A|$.
- **Terms:** $|x\langle S\rangle| = x$, $|(t\ \ t')| = (|t|\ \ |t'|)$, $|(t\ \ A)| = (|t|\ \ |A|)$, $|\lambda x : A.t| = \lambda x : |A|.|t|$, $|\lambda X : u.t| = \lambda X.|t|$, and $|\mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t'| = (\lambda x : |A|.|t|\ \ |t'|)$.

This surjection induces an equivalence relation on $F^{\Pi}_{\leq:}$ types and terms: we say that types A, A' (resp. terms t, t') have the same **skeleton** and write $A \simeq A'$ (resp. $t \simeq t'$), if $|A| = |A'|$ (resp. $|t| = |t'|$).

2.2 Type inference system

In this section we discuss $F^{\Pi}_{\leq:}$ typing rules. They differ from F typing rules on the following points:

- Types may be built on different universes.
- Based on basic constraints over universes a subtyping relation between types is defined.
- By means of universe variable substitutions, types are related via instantiations and generic instantiations.

We first define the subtyping relation $\leq:$ between types which is an extension of $\leq:$ to types.

Definition 2.3 (Subtyping)
$\leq:$ *between types is defined as follows:*

$$[Ref] X \leq: X$$

$$[\rightarrow] \quad \frac{A'_1 \leq: A_1 \quad A_2 \leq: A'_2}{A_1 \rightarrow A_2 \leq: A'_1 \rightarrow A'_2} \qquad [\Pi] \quad \frac{u_2 \leq: u_1 \quad A_1[Y := X] \leq: A_2[Z := X]}{\Pi Y : u_1.A_1 \leq: \Pi Z : u_2.A_2}$$

where X is a fresh type variable (it allows $\Pi X : u.X \leq: \Pi Y : u.Y$).

Let $A \leq: A'$, we define $\mathsf{CSet}(A \leq: A')$ as the set of universes constraints required to satisfy $A \leq: A'$. $\mathsf{CSet}(.)$ is defined by induction on the form of A and A':
$\mathsf{CSet}(X \leq: X) = \emptyset$
$\mathsf{CSet}(A_1 \rightarrow A_2 \leq: A'_1 \rightarrow A'_2) = \mathsf{CSet}(A'_1 \leq: A_1) \cup \mathsf{CSet}(A_2 \leq: A'_2)$
$\mathsf{CSet}(\Pi X : u_1.A_1 \leq: \Pi Y : u_2.A_2) = \{u_2 \leq: u_1\} \cup \mathsf{CSet}(A_1 \leq: A_2)$

Well-formed types and well-formed context are mutually inductively defined. Rules are given in Fig. 1.

Well-formed terms are defined by rules given in Fig. 2. A derivation tree Ξ is valid only if it is built with valid judgments.

In Fig. 2, C denotes any set of inequalities on $\mathcal{U}$.
The intuition behind the function $\mathtt{Gen}(.)$, is more or less the same than the one behind ML generalization. The idea is that the most general type of a term is obtained by abstraction of its free variables (here universe variables). Because of subtyping, we cannot simply abstract over free universe variables, since there may be constraints between universe variables that may be not verified by random instantiations. Consider the following derivation:

$$\frac{\dfrac{\Gamma \vdash t : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2} \quad \Gamma \vdash x : \mathcal{N}^{\alpha_3} \quad \mathcal{N}^{\alpha_3} \leq: \mathcal{N}^{\alpha_1}}{\Gamma \vdash (t \;\; x) : \mathcal{N}^{\alpha_2}}}{\vdash \lambda t : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2}.\lambda x : \mathcal{N}^{\alpha_3}.(t \;\; x) : \mathcal{N}^{\alpha_2}}$$

where $\Gamma = t : \mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2}, x : \mathcal{N}^{\alpha_3}$, and $\mathcal{N}^u = \Pi X : u.(X \rightarrow X) \rightarrow X \rightarrow X$. The most general type induced by this derivation, would be likely $\forall \alpha_1, \alpha_2, \alpha_3.(\mathcal{N}^{\alpha_1} \rightarrow \mathcal{N}^{\alpha_2}) \rightarrow \mathcal{N}^{\alpha_3} \rightarrow \mathcal{N}^{\alpha_2}$. However this scheme would lead

– **Contexts:**

$$[Ax]\ \frac{}{\emptyset \vdash}$$

$$[\Gamma T]\ \frac{\Gamma \vdash \quad X \notin FV(\Gamma)}{\Gamma, X : u \vdash}\quad [\Gamma t]\ \frac{\Gamma \vdash \sigma : u \quad x \notin FV(\Gamma)}{\Gamma, x : \sigma \vdash}$$

– **Types:**

$$[\forall C]\ \frac{\Gamma \vdash \sigma : u \quad \alpha \notin FUV(\Gamma)}{\Gamma \vdash \forall \alpha . C \Rightarrow (\sigma) : u}$$

$$[\rightarrow C]\ \frac{\Gamma \vdash A : u \quad \Gamma \vdash A' : u'}{\Gamma \vdash A \rightarrow A' : u'}\qquad [Tvar]\ \frac{\Gamma, X : u \vdash}{\Gamma, X : u \vdash X : u}$$

$$[Tadd]\ \frac{\Gamma \vdash A : u \quad \Gamma, X : u' \vdash}{\Gamma, X : u' \vdash A : u}\quad [tadd]\ \frac{\Gamma \vdash A : u \quad \Gamma, x : A' \vdash}{\Gamma, x : A' \vdash A : u}$$

$$[\Pi C]\ \frac{\Gamma, X : u \vdash A : u'}{\Gamma \vdash \Pi X : u . A : u'}$$

Figure1. Rules for well-formed context and types

$$[Hyp]\ \frac{\Gamma, x : \sigma \vdash \quad \sigma \overset{S}{\rightsquigarrow} A}{\Gamma, x : \sigma \vdash x\langle S \rangle : A}\qquad [add]\ \frac{\Gamma \vdash t : A \quad \Gamma, x : \sigma \vdash}{\Gamma, x : \sigma \vdash t : A}$$

$$[\rightarrow I]\ \frac{\Gamma, x : A' \vdash t : A}{\Gamma \vdash \lambda x : A' . t : A' \rightarrow A}\qquad [\Pi I]\ \frac{\Gamma, X : u \vdash t : A}{\Gamma \vdash \lambda X : u . t : \Pi X : u . A}$$

$$[\rightarrow E]\ \frac{\Gamma \vdash t_1 : A_2 \rightarrow A_1 \quad \Gamma \vdash t_2 : A_2' \quad A_2' \leq : A_2}{\Gamma \vdash (t_1 \ \ t_2) : A_1}$$

$$[\Pi E]\ \frac{\Gamma \vdash t : \Pi X : u . A \quad \Gamma \vdash A' : u' \quad u' \leq : u}{\Gamma \vdash (f \ \ A') : A[X := A']}$$

$$[Poly]\ \frac{\Xi \rhd \Gamma \vdash t : A \quad \Gamma, x : \mathtt{Gen}(\Xi) \vdash t' : A'}{\Gamma \vdash \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'}$$

Figure2. $F^{\Pi}_{\leq:}$ Typing rules

to type inconsistency since the substitution $S = [\alpha_1 \mapsto \top, \alpha_3 \mapsto \bot]$ would produce the term $\lambda t : \mathcal{N}^{\top} \rightarrow \mathcal{N}^{\alpha_2} . \lambda x : \mathcal{N}^{\bot} . (t \ \ x)$, which is not derivable from Fig. 2 rules.

$\mathtt{Gen}$ is defined in a mutual inductive way with the term typing judgments: if $\Xi \rhd \Gamma \vdash t : A$, then

$$\mathtt{Gen}(\Xi) = \begin{cases} \forall \alpha_1, \ldots, \alpha_n.\mathsf{CSet}(\Xi) \Rightarrow (A) \quad FUV(A) \setminus FUV(\Gamma) = \{\alpha_1, \ldots, \alpha_n\} \\ A \text{ if n=0} \end{cases}$$

where $\mathsf{CSet}(\Xi)$ is the extension of constraint set to derivation trees. It is the set constraints generated by subtyping assumptions in Ξ. More precisely:

$$\mathsf{CSet}\left(\frac{\Xi \rhd \Gamma, x : \sigma \vdash \quad \sigma \overset{S}{\rightsquigarrow} A}{\Gamma, x : \sigma \vdash x\langle S \rangle : A}\right) = S(C)$$

if $\sigma = \forall \alpha_1, \ldots, \alpha_n.C \Rightarrow (A_0)$.

$$\mathsf{CSet}\left([add] \quad \frac{\Xi \rhd \Gamma \vdash t : A \quad \Gamma, x : \sigma \vdash}{\Gamma, x : \sigma \vdash t : A}\right) = \mathsf{CSet}(\Xi)$$

$$\mathsf{CSet}\left(\frac{\Xi \rhd \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B}\right) = \mathsf{CSet}(\Xi)$$

$$\mathsf{CSet}\left(\Xi = \frac{\Xi \rhd \Gamma, X : u \vdash t : A}{\Gamma \vdash \lambda X : u.t : \Pi X : u.A}\right) = \mathsf{CSet}(\Xi)$$

$$\mathsf{CSet}\left(\frac{\Xi \rhd \Gamma \vdash t_1 : A_2 \to A_1 \quad \Xi' \rhd \Gamma \vdash t_2 : A_2' \quad A_2' \leq : A_2}{\Gamma \vdash (t_1 \ \ t_2) : B}\right)$$
$$= \mathsf{CSet}(\Xi) \cup \mathsf{CSet}(\Xi') \cup \mathsf{CSet}(A' \leq : A)$$

$$\mathsf{CSet}\left(\frac{\Xi \rhd \Gamma \vdash t : \Pi X : u.A \quad \Xi' \rhd \Gamma \vdash A' : u' \quad u' \leq : u}{\Gamma \vdash (t \ \ A') : A[X := A']}\right)$$
$$= \mathsf{CSet}(\Xi) \cup \mathsf{CSet}(\Xi') \cup \{u' \leq : u\}$$

$$\mathsf{CSet}\left(\frac{\Xi \rhd \Gamma \vdash t : A \quad \Xi' \rhd \Gamma, x : \mathtt{Gen}(\Xi) \vdash t' : A'}{\Gamma \vdash \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'}\right) = \mathsf{CSet}(\Xi) \cup \mathsf{CSet}(\Xi')$$

The mutually inductive definition of $\mathtt{Gen}(\Xi)$ with term typing rules is sound since derivation trees are finite and recursive calls are done on strictly smaller derivation trees.

We say that a substitution S preserves a derivation tree $\Xi \rhd \Gamma \vdash t : A$ if S preserves $\mathsf{CSet}(\Xi)$.

The relation between F and $F^{\Pi}_{\leq:}$ is extended to typing derivation. We denote F judgments by $\vdash_F$. F typing rules are simply obtained from $F^{\Pi}_{\leq:}$ rules by erasing all universes. We have the following trivial fact:

Fact 2.1 *If $\Gamma \vdash t : A$ is a valid derivation tree of $F^{\Pi}_{\leq:}$ then $|\Gamma| \vdash_F |t| : |A|$.*

Fact 2.2 *Conversely, for a given F well formed term, t_F, there exists many well formed $F^{\Pi}_{\leq:}$ terms t', such that $|t'| = t_F$. One can remark that "many" means at least 2: there are the extreme cases when universes used are either all $\top$ or all $\bot$.*

2.3 System properties

We define a notion of reduction in this calculus. The only noticeable feature is the reduction of let redexes. When a let binding is used, variables are introduced together with a substitution. During the reduction this substitution is applied to the term that substitutes the bound variable.

Definition 2.4 (Reductions) *The $F^{\Pi}_{\leq:}$ β-reduction is defined as follows:*

$$(\lambda x : A.t \;\; t') \rightarrow_\beta t[x\langle S\rangle := S(t')]$$

$$(\lambda X : b.t \;\; A) \rightarrow_\beta t[X := A]$$

We write $=_\beta$ the reflexive, transitive closure of $\rightarrow_\beta$, and $\rightarrow^*_\beta$ the transitive closure of $\rightarrow_\beta$.

We now study basic properties of $\rightarrow_\beta$ on $F^{\Pi}_{\leq:}$ well formed terms. First we make a straightforward comment regarding relations between $F^{\Pi}_{\leq:}$ reductions and F β-reductions.

Fact 2.3 *Let t be a well formed $F^{\Pi}_{\leq:}$ term. If $t \rightarrow_\beta t'$ then $|t| \rightarrow_\beta |t'|$ in F.*

Theorem 2.1 (Strong Normalization) *$\rightarrow_\beta$ is strongly normalizing on $F^{\Pi}_{\leq:}$ terms.*

Proof: Fact 2.3 and F Strong Normalization implies directly $F^{\Pi}_{\leq:}$ S.N. □

The Church-Rosser property is not as easy to prove. Indeed, in general substitutions do not commute, hence no simple adaptation of F proof should be expected. However, substitutions in well formed terms satisfy an important property: if $x\langle S\rangle$ and $y\langle S'\rangle$ occur in a well formed term t with $y \neq x$. Then, if S, S' are not the identity, it means that x, y are bound by a `let` expression. Now, after a correct renaming of bound variables (which are exactly the Domain of substitutions S, S'), we can ensure $D(S) \cap D(S') = \emptyset$. Hence S, S' commute.

Theorem 2.2 (Church Rosser) *Let t, t_1, t_2 be well formed $F^{\Pi}_{\leq:}$ terms such that $t \rightarrow_\beta t_1$ and $t \rightarrow_\beta t_2$, then there exist a term t_3 such that $t_1, t_2 \rightarrow^*_\beta t_3$.*

Proof: As we have seen substitutions commute, hence we can apply F result (see for instance [GLT89]). □

We now study typing derivation properties. We first address the question of typing judgment stability through universe substitutions.

Theorem 2.3 *Let $\Gamma \vdash A : u$, and S a substitution. Then $S(\Gamma) \vdash S(A) : S(u)$.*

Proof: An easy induction. □

Theorem 2.4 (Stability of Typing judgments) *Let $\Xi \rhd \Gamma \vdash t : A$ be a valid derivation tree of $F^{\Pi}_{\leq:}$, and S a substitution preserving Ξ, then*

$$S(\Gamma) \vdash S(t) : S(A)$$

is a valid $F^{\Pi}_{\leq:}$ judgment.

Proof: The proof is naturally done by structural induction, and only three rules need to be carefully checked:

- $[Hyp]$: Let $\Xi \rhd \Gamma \vdash x\langle S_x \rangle : A$. One has $\sigma \overset{S_x}{\rightsquigarrow} A$ for some $\sigma = \forall \alpha_1, \ldots, \alpha_n.C \Rightarrow (A_x)$ and $x : \sigma \in \Gamma$, such that $A = S_x(A_x)$ (with S_x preserving C). Assuming a correct renaming of bound variables, it is safe to suppose that $D(S) \cap \{\alpha_1, \ldots, \alpha_n\} = \emptyset$. We define substitution S_{unk} such that $D(S_{unk}) = \{\alpha_1, \ldots, \alpha_n\}$ and $S_{unk}(\alpha_i) = S(S_x(\alpha_i))$. Therefore:
  - $S_{unk}(S(\alpha_i)) = S_{unk}(\alpha_i) = S(S_x(\alpha_i))$ for $\alpha_i \in \{\alpha_1, \ldots, \alpha_n\}$.
  - $S_{unk}(S(\beta)) = S(\beta) = S(S_x(\beta))$ for any $\beta \notin \{\alpha_1, \ldots, \alpha_n\}$.

  Thus, it is clear that $S_{unk}(S(A_x)) = S(S_x(A_x)) = S(A)$. Moreover, it is clear that S_{unk} preserves $S(C)$. Indeed, S_x preserves C, hence $S_x(C)$ holds and since S is Ξ consistent it is also $S_x(C)$-consistent. Thus, $S(A)$ is an instance of $S(\sigma) = \forall \alpha_1, \ldots, \alpha_n.S(C) \Rightarrow (S(A_0))$.
- $[\rightarrow E]$ Let $\Xi \rhd \Gamma \vdash (t_1\ t_2) : B$. One must have $\Gamma \vdash t_1 : A_2 \rightarrow A_1$, $\Gamma \vdash t_2 : A'_2$ and $A'_2 \leq: A_2$. Now by induction hypothesis we have that $S(\Gamma) \vdash S(t_1) : S(A_1) \rightarrow S(A_2)$, $S(\Gamma) \vdash S(t_2) : S(A'_2)$. Hence one has only to prove that $S(A'_2) \leq: S(A_2)$, which is straightforward since S preserves Ξ (thus all inequalities remains true relatively to S).
- $[Poly]$ Let $\Xi \rhd \Gamma \vdash \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'$. One has

$$\frac{\Xi' \rhd \Gamma \vdash t : A \qquad \Gamma; x : \mathtt{Gen}(\Xi') \vdash t' : A'}{\Gamma \vdash \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'}$$

  where $\mathtt{Gen}(\Xi') = \forall \alpha_1, \ldots, \alpha_n.C \Rightarrow (A)$, with $\{\alpha_1, \ldots, \alpha_n\} \subseteq FUV(A) \setminus FUV(\Gamma)$. Let $\beta_1, \ldots, \beta_n$ be out of the scope of S and not free in Γ. Assume $S_1 = S \circ [\alpha_i \mapsto \beta_i]$. Then, $S_1(\Gamma) = S(\Gamma)$. Now we apply the induction hypothesis yielding $S(\Gamma) \vdash S(t) : S(A)$ and $S_1(\Gamma); x : S_1(\mathtt{Gen}(\Xi')) \vdash S_1(t') : S_1(A')$. It is easy to check that $\{\beta_1, \ldots, \beta_n\} = FUV(S_1(A)) \setminus FUV(S_1(\Gamma))$. Thus $S_1(\mathtt{Gen}(\Xi')) = \forall \beta_1, \ldots, \beta_n.S_1(C) \Rightarrow (S_1(A)) = S(\forall \alpha_1, \ldots, \alpha_n.C \Rightarrow (A)) = S(\mathtt{Gen}(\Xi'))$. Now since $S(\Gamma) = S_1(\Gamma)$ the induction hypotheses induce that $S(\Gamma) \vdash S(t) : S(A)$ and $S(\Gamma); x : S(\mathtt{Gen}(\Xi')) \vdash S(t') : S(A')$. One has just to apply the rule $[Poly]$ to conclude.

□

It is clear that the subject reduction property does not hold for $F^{\Pi}_{\leq:}$. Indeed, the term $t = (\lambda x : \mathcal{N}^{\perp}.x\ \ y)$ is well formed of type $\mathcal{N}^{\perp}$ under a context Γ with $y : \mathcal{N}^{\top} \in \Gamma$, but $t \rightarrow_{\beta} y$ and $\Gamma \vdash y : \mathcal{N}^{\perp}$ is not a valid judgment. Nevertheless, a weaker version of subject reduction holds.

Theorem 2.5 ("Weak" Subject Reduction) *Let $\Gamma \vdash t : A$ be a valid judgment and $t \rightarrow_\beta t'$, then $\Gamma \vdash t' : A'$ with $A' \leq : A$ is a valid judgment.*

Proof: The proof is done by showing substitution lemma for each redexes.

Lemma 2.1 *1. Suppose that $\Gamma \vdash t_0 : A'_0$, $\Gamma, x : A_0 \vdash t_1 : A_1$ and $A'_0 \leq : A_0$, then $\Gamma \vdash t_1[x := t_0] : A'_1$ and $A'_1 \leq : A_1$.*
2. Suppose that $\Gamma \vdash A_0 : u'_0$, $\Gamma, X : u_0 \vdash t : A$ and $u'_0 \leq : u_0$, then $\Gamma \vdash t[X := A_0] : A'$ and $A' \leq : A$.
3. Suppose that $\Xi_0 \rhd \Gamma \vdash t_0 : A_0$, $\Gamma, x : \mathtt{Gen}(\Xi_0) \vdash t_1 : A_1$ then $\Gamma \vdash t_1[x\langle s\rangle := s(t_0)] : A'_1$.

Proof:

1. It is done by induction on the form of t. We only treat the non trivial cases.
   - If $t_1 = y$. We have two cases, either $y \neq x$ or $y = x$. In the first case, since $t_1[x := t_0] = y = t_1$ we have $\Gamma \vdash t_1[x := t_0] : A_1$ which concludes. In the other case, if $t_1 = x$ then $t_1[x := t_0] = t_0$. By hypothesis we know that $\Gamma \vdash t_0 : A'_0$ with $A'_0 \leq : A_0$. Now since $\Gamma, x : A_0 \vdash t_1 : A_1$ and since $t = x$, one must have $A_0 = A_1$. Thus the result.
   - If $t_1 = \lambda y : A_y.t_{11}$, then $A_1 = A_y \rightarrow A_{11}$ and $\Gamma, x : A_0, y : A_y \vdash t_{11} : A_{11}$. Now, by induction hypothesis, we have $\Gamma, y : A_y \vdash t_{11}[x := t_0] : A'_{11}$ and $A'_{11} \leq : A_{11}$. Applying rule $[\rightarrow I]$ we obtain: $\Gamma \vdash \lambda y : A_y.t_{11}[x := t_0] : A_y \rightarrow A'_{11}$. Since $A_y \rightarrow A'_{11} \leq : A_y \rightarrow A_{11}$ we conclude.
   - If $t_1 = \lambda Y : u.t_{11}$, this case is handled similarly to the previous one.
   - If $t_1 = (t_{11}\ \ t_{12})$, then $\Gamma, x : A_0 \vdash t_{11} : A_{12} \rightarrow A_1$ and $\Gamma, x : A_0 \vdash t_{12} : A_{22}$ and $A_{22} \leq : A_{12}$. Now, we apply induction hypothesis on both branches and deduce
     (a) $\Gamma \vdash t_{11}[x := t_0] : A'_{12} \rightarrow A_{11}$ with $A_{12} \leq : A'_{12}$ and $A_{11} \leq : A_1$.
     (b) $\Gamma \vdash t_{12}[x := t_0] : A'_{22}$ and $A'_{22} \leq : A_{22}$.
     So we have $A'_{22} \leq : A_{22} \leq : A_{12} \leq : A'_{12}$, therefore we can apply rule $[\rightarrow E]$ and get $\Gamma \vdash (t_{11}[x := t_0]\ \ t_{12}[x := t_0]) : A_{11}$. Since $A_{11} \leq : A_1$, it concludes.
   - If $t_1 = (t_{11}\ \ A_{11})$, this case is handled similarly to the previous one.
   - If $t_1 = \mathsf{let}\ y : A_{11}\ \mathsf{be}\ t_{11}\ \mathsf{in}\ t_{12}$. Then $\Xi_{11} \rhd \Gamma, x : A_0 \vdash t_{11} : A_{11}$ and $\Gamma, x : A_0, y : \mathtt{Gen}(\Xi_{11}) \vdash t_{12} : A_1$. Now we apply the induction hypothesis on both branches and deduce:
     (a) $\Xi'_{11} \rhd \Gamma \vdash t_{11}[x := t_0] : A'_{11}$ and $A'_{11} \leq : A_{11}$.
     (b) $\Gamma, y : \mathtt{Gen}(\Xi_{11}) \vdash t_{12} : A'_1$ and $A'_1 \leq : A_1$.
     Clearly $\mathtt{Gen}(\Xi_{11}) \rightsquigarrow \mathtt{Gen}(\Xi'_{11})$, thus we can apply the rule $[Poly]$ to get the desired result.
2. as for point 1.
3. Also by induction hypothesis.

□

Now, the proof of the proposition reduces to a simple induction on redexes. For each case one has just to apply the correct statement of lemma 2.1. □

To prove theorem 2.6, we develop a new typing system $\vdash_\sharp$. The main differences between $\vdash$ and $\vdash_\sharp$ lies in context building. In $\vdash_\sharp$ we consider Π binders as universe variable binders. It not allows to share universe variables: all universe variables introduced are unique. In order to enforce this uniqueness we introduce marks in context statements. We define new syntactic categories

- **Marks:** Marks are given by the grammar

$$m ::= \circ \mid \bullet$$

- **Statements:** A statement is the triplet formed by a mark, a type and a universe.

$$s ::= mA : u$$

- **Marked contexts:** Marked contexts are lists of statements:

$$\Delta = \emptyset \mid \Delta; mA : u$$

Marked contexts may be seen as lists of well formed types. In these lists, all types are marked either with $\circ$ or $\bullet$. $\circ$ denotes a "free" type, i.e. a type that has never be used in a judgment, whereas $\bullet$ denotes a type that has already be used in a judgment.

The idea is the following. Suppose that you want to introduce variable x of type $\mathcal{N} \to \mathcal{N}$ in context. The most general way to do it is introduce it with the type $\mathcal{N}^{\alpha_1} \to \mathcal{N}^{\alpha_2}$ with $\alpha_1 \neq \alpha_2$. To do this, we record in marked context what types have been built. $\circ$ will denote free types of the context that may be used to build a type. $\bullet$ denotes types having already been used.
We define an equivalence relation between marked contexts: $\overset{\bullet}{=}$. $\Delta_1 \overset{\bullet}{=} \Delta_2$, means that Δ_1 and Δ_2 are equivalent except for marks.

- $\emptyset \overset{\bullet}{=} \emptyset$,
- if $\Delta_1 \overset{\bullet}{=} \Delta_2$ then $\Delta_1; \circ A : b \overset{\bullet}{=} \Delta_2; \circ A : b$,
- if $\Delta_1 \overset{\bullet}{=} \Delta_2$ then $\Delta_1; \bullet A : b \overset{\bullet}{=} \Delta_2; \circ A : b$.

We now define an operator $\wedge$ between two marked contexts. This operator build the inf of two contexts.

- $\emptyset \wedge \emptyset = \emptyset$.
- $(\Delta; \circ A : u) \wedge (\Delta'; \circ A : u) = \Delta \wedge \Delta'; \circ A : u$.
- $(\Delta; \bullet A : u) \wedge (\Delta'; \bullet A : u) = \Delta \wedge \Delta'; \bullet A : u$.
- $(\Delta; \circ A : u) \wedge (\Delta'; \bullet A : u) = \Delta \wedge \Delta'; \bullet A : b$.
- $(\Delta; \bullet A : u) \wedge (\Delta'; \circ A : u) = \Delta \wedge \Delta'; \bullet A : b$.

With $\Gamma \oplus \Delta$ we denote the concatenation of a context Γ and a marked context Δ. In Fig. 3, we give context formation rules for $\vdash_\sharp$.

Four rules change from $\vdash$ context formation rules:

1. $[\Gamma\sharp]$: Type variables are introduced in context with fresh *universe variables*. Thus no $\bot$ nor $\top$ may occur in well formed contexts.

– **Contexts:**

$$[Ax\sharp]\ \frac{}{\emptyset \oplus \emptyset \vdash_\sharp} \qquad [\Gamma T\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp \quad \begin{array}{c} X \notin FV(\Gamma) \\ \alpha \notin FUV(\Gamma \cup \Delta) \end{array}}{\Gamma, X : \alpha \oplus \Delta \vdash_\sharp}$$

$$[\Gamma t\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp \sigma : u \quad x \notin \Gamma}{\Gamma, x : \sigma \oplus \Delta \vdash_\sharp}$$

– **Types:**

$$[Tvar\sharp]\ \frac{\Gamma, X : b \oplus \Delta \vdash_\sharp}{\Gamma, X : b \oplus \Delta \vdash_\sharp X : b}$$

$$[\forall C\sharp]\ \frac{\Gamma \oplus \Delta \vdash \sigma : u \quad \alpha \notin FUV(\Gamma)}{\Gamma \oplus \Delta \vdash \forall \alpha . C \Rightarrow (\sigma) : u}$$

$$[\rightarrow C\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp A : u \quad \Gamma \oplus \Delta' \vdash_\sharp A' : u' \quad \Delta \overset{\bullet}{=} \Delta'}{\Gamma \oplus \Delta \wedge \Delta' \vdash_\sharp A \rightarrow A' : u'}$$

$$[\Delta\bullet]\ \frac{\Gamma \oplus \Delta, \circ A : u, \Delta' \vdash_\sharp}{\Gamma \oplus \Delta, \bullet A : u, \Delta' \vdash_\sharp A : u}$$

$$[\Delta\circ]\ \frac{\Gamma, X : u \oplus \Delta \vdash_\sharp A : u'}{\Gamma \oplus \Delta, \circ \Pi X : u . A : u' \vdash_\sharp}$$

$$[Tadd\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp A : u \quad \Gamma, X : u' \oplus \Delta \vdash_\sharp}{\Gamma, X : u' \oplus \Delta \vdash_\sharp A : u}$$

$$[tadd\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp A : u \quad \Gamma, x : A' \oplus \Delta \vdash_\sharp}{\Gamma, x : A' \oplus \Delta \vdash_\sharp A : u}$$

Figure3. $\vdash_\sharp$ context formation rules

2. $[\rightarrow C\sharp]$: This rule weakens marked context by taking the $\wedge$ of the two marked contexts used to build both parts of an arrow type.
3. $[\Delta\circ]$: It denotes the use of a type. Before its use the type was marked "free" (with a $\circ$ mark) and after it is marked with $\bullet$ because this type has occurred in the right side of $\vdash_\sharp$. Once marked with $\bullet$ the type can no longer be used in future derivations.
4. $[\Delta\bullet]$: This rule allows the introduction of new types in the marked context. It is the only way to build product types.

We now show that $\vdash_\sharp$ formation context is a special case of $\vdash$ formation context.

Proposition 2.1 *for all type context Δ and context Γ*

$$\Gamma \oplus \Delta \vdash_\sharp \Longrightarrow \Gamma \vdash$$

$$\Gamma \oplus \Delta \vdash_\sharp A : \alpha \Longrightarrow \Gamma \vdash A : \alpha$$

Proof: The proof is done by mutual induction on the form of the context and on the form of the type derived.

For context formation:

- If $\Gamma = \emptyset$, it is straightforward.
- If $\Gamma = \Gamma', X : \alpha$. Then by rule $[\Gamma T\sharp]$, we know that X does not occur in Γ'. By induction hypothesis, we know that $\Gamma' \vdash$, thus $\Gamma', X : \alpha \vdash$ by rule $[\Gamma T]$.
- If $\Gamma = \Gamma', x : \sigma$. By $[\Gamma t\sharp]$, we know that x does not occur in Γ and $\Gamma \oplus \Delta \vdash_\sharp \sigma : u$. By induction we have that $\Gamma \vdash \sigma : u$, and we can apply rule $[\Gamma t]$.

For type formation:

- If $\Gamma \oplus \Delta \vdash_\sharp X : u$. It means that $X : u$ occurs in Γ, which by induction hypothesis is such that $\Gamma \vdash$. Thus $\Gamma \vdash X : u$.
- If $\Gamma \oplus \Delta \vdash_\sharp \Pi X : u.A : u'$. One can note that the only to introduce a Π binders comes from the rule $[\Delta\circ]$. Moreover, if $\Gamma \oplus \Delta \vdash_\sharp \Pi X : u.A : u'$, then the rule $\Delta\bullet$ must have been applied. Therefore the $\vdash_\sharp$ derivation tree has the following form :

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, X : u \oplus \Delta \vdash_\sharp A : u'}{\Gamma \oplus \Delta, \circ \Pi X : u.A : u' \vdash_\sharp}}{\vdots}}{\Gamma, \Delta, \circ \Pi X : u.A : u', \Delta' \vdash_\sharp}}{\Gamma, \Delta, \bullet \Pi X : u.A : u', \Delta' \vdash_\sharp \Pi X : u.A : u'}$$

  We apply the induction hypothesis on $\Gamma, X : u \oplus \Delta \vdash_\sharp A : u'$, thus $\Gamma, X : u \vdash A : u'$ and we conclude by the application of rule $[\Pi I]$.
- If $\Gamma \oplus \Delta \vdash_\sharp A \to A' : u$, it is straightforward from the application of the induction hypothesis on both branches A, A'.
- If $\Gamma \oplus \Delta \vdash_\sharp \forall \alpha.C \Rightarrow (\sigma) : u$. Then from rule $[\forall C\sharp]$ we have $\Gamma \oplus \Delta \vdash_\sharp \sigma : u$. We can apply the induction hypothesis to conclude (since α does not occur in Γ).

□

$\vdash_\sharp$ is a constructive way to build contexts. Indeed, for a given F context, there exists only one (up to universe substitution) $\vdash_\sharp$ context.

Proposition 2.2 *Let $\Gamma \oplus \Delta \vdash_\sharp$, $\Gamma' \oplus \Delta' \vdash_\sharp$ and $\Gamma' \simeq \Gamma$. Then, there exists S, S' such that $S(\Gamma) = \Gamma'$ and $S'(\Gamma') = \Gamma$.*

$$[Hyp\sharp]\ \frac{\Gamma, x:\sigma \oplus \Delta \vdash_\sharp \quad \sigma \overset{S}{\rightsquigarrow} A}{\Gamma, x:\sigma \oplus \Delta \vdash_\sharp x\langle S\rangle : A}$$

$$[add\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp t : A \quad \Gamma, x:\sigma \oplus \Delta \vdash_\sharp}{\Gamma, x:\sigma \oplus \Delta \vdash_\sharp t : A}$$

$$[\rightarrow I\sharp]\ \frac{\Gamma, x:A \oplus \Delta \vdash_\sharp t : A'}{\Gamma \oplus \Delta \vdash_\sharp \lambda x:A.t : A \rightarrow A'}$$

$$[\Pi I\sharp]\ \frac{\Gamma, X:\alpha \oplus \Delta \vdash_\sharp t : A}{\Gamma \oplus \Delta \vdash_\sharp \lambda X:\alpha.t : \Pi X:\alpha.A}$$

$$[\rightarrow E\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp t_1 : A_1 \rightarrow A_2 \quad \Gamma \oplus \Delta \vdash_\sharp t_2 : A' \quad A' \leq: A_1}{\Gamma \oplus \Delta \vdash_\sharp (t_1 \ \ t_2) : A_2}$$

$$[\Pi E\sharp]\ \frac{\Gamma \oplus \Delta \vdash_\sharp t : \Pi X:u.A \quad \Gamma \oplus \Delta \vdash_\sharp A' : u' \quad \Pi X:u.A \leq: \Pi X:u'.A}{\Gamma \oplus \Delta \vdash_\sharp (t \ \ A') : A[X := A']}$$

$$[Poly\sharp]\ \frac{\Xi \rhd \Gamma \oplus \Delta \vdash_\sharp t : A \quad \Gamma, x : \mathtt{Gen}(\Xi) \oplus \Delta \vdash_\sharp t' : A'}{\Gamma \oplus \Delta \vdash_\sharp \mathsf{let}\ x : A\ \mathsf{be}\ t\ \mathsf{in}\ t' : A'}$$

Figure4. $\vdash_\sharp$ Typing rules

Proof: straightforward since universes introduced are all fresh universe variables. □

We define the judgment $\Gamma \oplus \Delta \vdash_\sharp t : A$ to be red, under the context Γ and marked context Δ, t is well formed of type A. The rules are given in Fig. 4

Proposition 2.3

- *Suppose that $\Gamma \oplus \Delta \vdash_\sharp t : A$ then $\Gamma \vdash t : A$ is a valid derivation.*
- *Let $\Gamma \oplus \Delta \vdash_\sharp$. Then $\Gamma \vdash t : A$ implies $\Gamma \oplus \Delta \vdash_\sharp t : A$.*

Proof: From proposition 2.1, we now that if Γ is a well formed context for $\vdash_\sharp$ then Γ is a well formed context for $\vdash$. Now, $\vdash_\sharp$ typing rules (see Fig 4) are exactly the same than $\vdash$ typing rules, except the fact that a marked context occurs in $\vdash_\sharp$ typing rules. Since marked context does not play any role in typing rules we have the result. □

Propositions 2.1 and 2.3, show that $\vdash_\sharp$ is a restriction of $\vdash$. Indeed, except for context formation, $\vdash_\sharp$ is equivalent to $\vdash$. $\vdash_\sharp$ context formation has the following properties:

– only universe variable are used in valid derivation (i.e. $\vdash_\sharp$ derivations are analyses),
– closed types are built on distinct universe variables.

Proposition 2.4 *1. Let $\Gamma \vdash$, then there exists Γ_{gen} such that for some marked context Δ and $\Gamma_{gen} \oplus \Delta \vdash_\sharp$ there exists a substitution S and $S(\Gamma_{gen}) = \Gamma$.*
2. Let $\Gamma \simeq \Gamma'$ be well formed $\vdash$ contexts, there exists a context Γ_{gen}, substitutions S, S', type context Δ such that $\Gamma_{gen} \oplus \Delta \vdash_\sharp$ and $S(\Gamma_{gen}) = \Gamma$ and $S'(\Gamma_{gen}) = \Gamma'$.

Proof:

1. It is done by induction on the form of Γ.
   – If Γ is empty, it is straightforward.
   – If $\Gamma = \Gamma', X : u$. By induction we have that $\Gamma'_{gen} \oplus \Delta \vdash_\sharp$ and $S(\Gamma'_{gen}) = \Gamma'$. Now, $X \notin \Gamma'$ implies $X \notin \Gamma'_{gen}$. Therefore if $\alpha \notin FUV(\Gamma') \cup FUV(\Delta)$, $\Gamma', X : \alpha \oplus \Delta \vdash_\sharp$. $S[\alpha \mapsto u](\Gamma', X : \alpha) = \Gamma$.
   – If $\Gamma = \Gamma', x : A$. By induction we have that $\Gamma'_{gen} \oplus \Delta \vdash_\sharp$ and $S(\Gamma'_{gen}) = \Gamma'$. Now, by proposition 2.1 we have that $\Gamma'_{gen} \vdash$. By theorem 2.3, $\Gamma'_{gen} \vdash A' : u'$ and $S(A') = A$ $S(u') = u$. Thus, by proposition 2.3, $\Gamma_{gen'} \oplus \Delta' \vdash A' : u'$. Hence the result.
2. It is a simple application of the first point and proposition 2.2.

□

We are now ready to prove the following theorem.

Theorem 2.6 (Most General Derivation) *Suppose that $\Gamma_F \vdash_F t_F : A_F$. Then, there exists a* **most general derivation** *(MGD for short) such that : $\Gamma_{gen} \oplus \Delta \vdash_\sharp t_{gen} : A_{gen}$, with $|\Gamma_{gen}| = \Gamma_F$, $|t_{gen}| = t_F$ and $|A_{gen}| = A_F$.*

Proof:

Suppose that $\Gamma_F \vdash_F t_F : A_F$. We know that there exists $\Gamma \vdash t : A$ such that $|\Gamma| = \Gamma_F$, $|t| = t_F$ and $|A| = A_F$ from fact 2.2. Now, from propositon 2.4 the result follows immediately.

□

The implications of theorem 2.6 are clear. Suppose given t_F, a well formed F term. We know from the theorem that there exists a valid derivation of some term t_{gen} for $\vdash_\sharp$ such that $|t_{gen}| = t_F$. It is easy to see that from propositions 2.3,2.1 and 2.4, and due to the stability of typing judgments via substitutions that any valid derivation of $\Gamma \vdash t : A$ in $F^{\Pi}_{\leq:}$ is such that $S(\Gamma_{gen}) = \Gamma$, $S(t_{gen}) = t$ and $S(A_{gen}) = A$.

3 Behind the scene

In this section, we provide the intuitions that led to the definition of a multi universe typing system. We also informally discuss how $F^{\Pi}_{\leq:}$ can be used to formalize static analyses.

One way of interpreting types is to think of them as partial equivalence relation over an untyped domain D. These interpretations, called PER interpretation, combines a way of interpreting untyped lambda terms as elements of some set D with a way of interpreting types as PER on D (see [Mit90]). The underlying idea is to see the PER interpreting a type A as a typed equality between terms of A. Thus, in standard PER interpretations, the partial equivalence considered are restriction of the equality over the domain of the type. Now, imagine that instead of equality, types are interpreted as restrictions of the trivial relation which relates all elements to all. Under this interpretation it would be impossible to distinguish a term of another one. We claim that types build on universe $\top$ (we will talk about **positive** types) are interpreted in a standard way whereas types build on universe $\bot$ (we will talk about **negative** types) are interpreted as trivial PER. This idea was used both for dead code analysis [BB95] and binding time analysis [Hun91], in simply typed calculi.

The reason why $F^{\Pi}_{\leq:}$ may be used for program analyses becomes now clear. Suppose that a term t is a function which takes an argument of a negative type to give back a result of positive type. It means, under the previous semantic, that *whatever* is the input of t, its result will remain unchanged. In other words t is a function constant on its argument.

The originality of our approach lies in the use of universe variables. The idea, already present in [Pro97], is that the sharing of universe variables allows to represent constraints across the term. Thus, the most general type of a term might be seen as a description of the minimal requirements for an analysis to be sound. Once done, the analysis schema can be instantiated to several analyses. In the following section we consider dead code and binding time analyses.

4 Program Analysis

In this section we show how $F^{\Pi}_{\leq:}$ can be used for various static analyses. We only consider closed terms, called programs. We start by the definition of an observational equivalence between $F^{\Pi}_{\leq:}$ programs. We introduce *simplified* terms. They are terms whom negative subterms have been replaced by dummy constants. Then, we study relationships between observational and simplification relation. We show how dead code and binding time can be formalized and proved correct in our setting. Finally, we give some examples of program analyzes.

4.1 Observational equality and Simplified terms

We define **observational equivalence**, which serves as semantics of $F^{\Pi}_{\leq:}$ programs. Two programs are equivalents if no observation (i.e. any program that takes programs as input and give back a boolean) is able to distinguish them.

We define $\mathcal{B}\mathrm{ool}^u \stackrel{def}{=} \Pi X : u.X \to X \to X$, the impredicative encoding of boolean type. We also define the two inhabitants of this type:

$$\mathsf{True}^u \stackrel{def}{=} \lambda X : u.\lambda x_1 : X.\lambda x_2 : X.x_1$$

and

$$\mathsf{False}^u \stackrel{def}{=} \lambda X : u.\lambda x_1 : X.\lambda x_2 : X.x_2$$

Definition 4.1 (observational equivalence) *Let t_1, t_2 be two programs such that $\vdash t_1 : A$ and $\vdash t_2 : A$, then*

$$t_1 =_{obs} t_2 \iff \textit{for all closed } t \textit{ of type } A \to \mathcal{B}ool^{\top} \quad (t\ \ t_1) =_\beta (t\ \ t_2)$$

For any negative type A we introduce a dummy constant $\mathtt{d_A}$.

$$[\mathsf{d}]\ \ \frac{\Gamma \vdash \mathtt{A} : \bot}{\Gamma \vdash \mathtt{d_A} : \mathtt{A}}$$

It is easy to check that the extended system shares the same properties than $F^{\Pi}_{\leq:}$.

We now introduce a simplification relation based on dummy terms.

Definition 4.2 *The simplification relation $\sqsubseteq$ on terms is inductively defined on well formed terms as follows:*

- $\mathtt{d_A} \sqsubseteq t$, *for any well formed term of negative type A.*
- $x\langle S\rangle \sqsubseteq x\langle S\rangle$.
- $\lambda x : A.t_1 \sqsubseteq \lambda y : A.t_2$ *if* $t_1[x := z] \sqsubseteq t_2[y := z]$.
- $\lambda X : u.t_1 \sqsubseteq \lambda Y : u.t_2$ *if* $t_1[X := Z] \sqsubseteq t_2[Y := Z]$.
- $(t_1\ \ t'_1) \sqsubseteq (t_2\ \ t'_2)$ *if* $t_1 \sqsubseteq t_2$ *and* $t'_1 \sqsubseteq t'_2$.
- $(t_1\ \ A) \sqsubseteq (t_2\ \ A)$ *if* $t_1 \sqsubseteq t_2$.
- $\mathsf{let}\ x : t_1\ \mathsf{be}\ A\ \mathsf{in}\ t'_1 \sqsubseteq \mathsf{let}\ x : t_2\ \mathsf{be}\ A\ \mathsf{in}\ t'_2$, *if* $t_1 \sqsubseteq t_2$ *and* $t'_1 \sqsubseteq t'_2$.

The simplification relation states that a term t is a simplified version of a term t' if t is obtained by the substitution of some negative subterms by a dummy constant of the corresponding type.

We now study the behavior of the simplification relation and observational equality.

Lemma 4.1 *If t, t' are well formed terms of type $\mathcal{B}ool^{\top}$ in normal form and $t \sqsubseteq t'$ then $t = t'$.*

Proof: Up to α-conversion the only closed terms of type $\mathcal{B}\mathrm{ool}^{\top}$ in normal form are $\mathsf{True}^{\top}$ and $\mathsf{False}^{\top}$. □

Lemma 4.2 *Suppose $\Xi_0 \rhd \Gamma \vdash t_0 : A_0$, $\Xi'_0 \rhd \Gamma \vdash t'_0 : A_0$, $\sigma_0 = \mathtt{Gen}(\Xi_0)$, $\sigma'_0 = \mathtt{Gen}(\Xi'_0)$, and $t_0 \sqsubseteq t'_0$. We have:*

1. *If* $\Gamma, x : A \vdash t_1 : A_1$, $\Gamma, x : A \vdash t'_1 : A_1$, $t_1 \sqsubseteq t'_1$, $A_0 \leq : A$. *Then,* $t_1[x := t_0] \sqsubseteq t'_1[x := t'_0]$.
2. *If* $\Gamma, X : u \vdash t_1 : A_1$, $\Gamma, X : u \vdash t'_1 : A_1$, $t_1 \sqsubseteq t'_1$, $\Gamma \vdash A : u_A$, *and* $u_A \leq : u$. *Then,* $t_1[X := A] \sqsubseteq t'_1[X := A]$.
3. *If* $\Gamma, x : \sigma_0 \vdash t_1 : A_1$, $\Gamma, x : \sigma'_0 \vdash t'_1 : A_1$, $t_1 \sqsubseteq t'_1$. *Then,* $t_1[x\langle S\rangle := S(t_0)] \sqsubseteq t'_1[x\langle S\rangle := S(t'_0)]$.

Proof:

1. It is an induction on the form of t_1:
   - If $t_1 = d_{A_1}$, straightforward.
   - If $t_1 = y$, then $t'_1 = y$. If $y = x$, then respectively $t_1[x := t_2] = t_2$ and $t_1[x := t'_2] = t'_2$. By hypothesis $t_2 \sqsubseteq t'_2$. Else $y \neq x$ and $t_1[x := t_2] = t_1$ and $t_1[x := t'_2] = t'_1$. Hence the result by $\sqsubseteq$ reflexivity.
   - If $t_1 = (t_{11}\ \ t_{12})$, then $t'_1 = (t'_{11}\ \ t'_{12})$, and result follows immediately from induction hypothesis on t_{11}, t'_{11} and t_{22}, t'_{22}.
   - If $t_1 = (t_{11}\ \ A)$, then $t'_1 = (t'_{11}\ \ A)$, and result follows immediately from induction hypothesis applied.
   - If $t_1 = \lambda x : A_x.t_{11}$, then $t'_1 = \lambda y : A_y.t'_{11}$. Then, the result follows from the application of induction hypothesis on $t_{11}[x := z]$ and $t'_{11}[y := z]$.
   - If $t_1 = \lambda X : u.t_{11}$. Then $t'_1 = \lambda Y : u.t_{11}$. The result follows from the application of the induction hypothesis on t_{11}.
   - If $t_1 = \mathsf{let}\ x : A\ \mathsf{be}\ t_{11}\ \mathsf{in}\ t_{12}$, since types play no role, it is showed as for $t_1 = (\lambda x : A.t_{11}\ \ t_{12})$.
2. It is a simple induction on the form of t_1. Since types do not interfere with $\sqsubseteq$, it is immediate.
3. Since types do not interfere with prune, substitutions neither. Therefore the proof is the same as in point 1.

□

Using the previous lemma, we are able to show that $\sqsubseteq$ and $\rightarrow_\beta$ commutes.

Theorem 4.1 *Let* $t_1 \sqsubseteq t_2$ *be two well formed programs. If* $t_2 \rightarrow_\beta t'_2$ *then, there exists* t'_1 *such that* $t_1 \rightarrow^*_\beta t'_1$, *and* $t'_1 \sqsubseteq t'_2$.

Proof: $\sqsubseteq$ is compatible with term formation. So we can safely only consider terms which are β-redexes. The proof is then a case analysis on the form of the redex. □

As first application of this theorem, we show that well formed programs related by the simplification relation and having type $\mathcal{B}\mathrm{ool}^\top$ have the same normal form.

Theorem 4.2 *Let* $t_1 \sqsubseteq t_2$ *be well formed programs of type* $\mathcal{B}ool^\top$. *Then* $t_1 =_\beta t_2$.

Proof: Since $F^{\Pi}_{\leq :}$ is strongly normalizing let us call t_1^{nf} and t_2^{nf} the respective normal forms of t_1, t_2. By lemma 4.2, we know that $t_1^{nf} \sqsubseteq t_2^{nf}$. Now applying lemma 4.1 we have that $t_1^{nf} = t_2^{nf}$ which implies directly $t_1 =_\beta t_2$. □

4.2 Binding Time Analysis

As stated in [Hun91], "given a description of the parameters in a program that will be known at partial evaluation time, a binding time analysis must determine which parts of the program are dependent solely on these parts (and therefore also known at partial evaluation time)". We show that if a function t of positive type, takes a term of negative type A, then the application of this function to *any term* of type A are observationnaly equivalent. In other words: *t is static relatively to its argument.*

Theorem 4.3 *If t is a program of type $A_1 \to A_2$, with A_1 a negative type and A_2 a positive type. Then, for any t_1, t_2 programs of type A_1, we have $(t\ \ t_1) =_{obs} (t\ \ t_2)$.*

Proof: Clearly, $(t\ \ \mathtt{d}_{\mathtt{A}_1}) \sqsubseteq (t\ \ t_1)$ and $(t\ \ \mathtt{d}_{\mathtt{A}_1}) \sqsubseteq (t\ \ t_2)$.

Now, if A_2 is $\mathcal{B}\mathrm{ool}^{\top}$, we have directly from theorem 4.2 that $(t\ \ \mathtt{d}_{\mathtt{A}_1}) =_{obs} (t\ \ t_1)$ and similarly that $(t\ \ \mathtt{d}_{\mathtt{A}_1}) \sqsubseteq (t\ \ t_2)$. Hence by transitivity of $=_{obs}$, $(t\ \ \mathtt{d}_{\mathtt{A}_1}) \sqsubseteq (t\ \ t_1) =_{obs} (t\ \ \mathtt{d}_{\mathtt{A}_1}) \sqsubseteq (t\ \ t_1)$.

If, A_2 is not $\mathcal{B}\mathrm{ool}^{\top}$. For any closed term t_{obs} of type $A_2 \to \mathcal{B}\mathrm{ool}^{\top}$, we can apply the last reasoning for $(t_{obs}\ \ (t\ \ t_1))$ and $(t_{obs}\ \ (t\ \ t_2))$ which are of type $\mathcal{B}\mathrm{ool}^{top}$. Therefore $(t\ \ t_1) =_{obs} (t\ \ t_2)$ by the definition of observational equivalence. □

Therefore, binding time analysis of a well formed F term t is done as follows in $F^{\Pi}_{\leq:}$. First, using $\vdash_\sharp$ the most general derivation of t is built. Then, all universe variables of known parameters are set to $\top$, and unknown parameters to $\bot$. Using the constraint set of the most general derivation, $\top$ and $\bot$ are propagated through the term.

4.3 Dead Code analysis

Instead of being based on its inputs, as binding time analysis is, Dead code analysis is rather based on its output. Indeed, it aims at finding what part of the program may be removed *without* altering its output. For this analysis, we use [Ber93] principle. In our setting it amounts to show that all negative subterms of a positive program might be seen as dead code.

Theorem 4.4 *Let $t_1 \sqsubseteq t_2$ be two well formed programs of positive type A. Then, $t_1 =_{obs} t_2$.*

Proof: If A is $\mathcal{B}\mathrm{ool}^{\top}$, then theorem 4.2 allows to conclude.

Else if A is not of type $\mathcal{B}\mathrm{ool}^{\top}$. For any program t of type $A \to \mathcal{B}\mathrm{ool}^{\top}$, we have $(t\ \ t_1) \sqsubseteq (t\ \ t_2)$, since $\sqsubseteq$ is compatible with term formation. Now if t_1^{nf} and t_2^{nf} are the respective normal forms of $(t\ \ t_1)$ and $(t\ \ t_2)$, we have by direct application of theorem 4.1 that $t_1^{nf} = t_2^{nf}$ which concludes. □

Theorem 4.4 states that if we replace a negative subterm of a positive program with a dummy constant, the observational behavior of the program does not change. It means that negative subterms are dead since their value has no influence over the observational behavior of the program.

Let t be a well formed program of system F. We can use $F^{\Pi}_{\leq:}$ to detect dead code in t in the following way. Theorem 2.6 states that there exists t_{gen} of type A_{gen}, such that for any $F^{\Pi}_{\leq:}$ term t' such that $|t'| = t$, there exists S'_t and $S_{t'}(t_{gen}) = t'$. Now, one has just to find a substitution S_{dead}, such that S_{dead} preserves the most general derivation tree and $S_{dead}(A_{gen})$ is a positive type. Then, all negative subterms of $S_{dead}(t_{gen})$ are dead and may be replaced with dummy constants.

4.4 Examples

Consider the following F program:

$$p_0 \stackrel{def}{=} \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}.5\ \ x))$$

in which $\mathcal{N} \stackrel{def}{=} \Pi X.(X \to X) \to X \to X$ is the impredicative encoding of natural types, the constant 5 has to be red $\lambda X.\lambda s : X \to X.\lambda z : X.(s\ \ (s \ldots z))$ as the impredicative encoding of this integer, and p a closed term of type $\mathcal{N}$.

In p_0 the variable x is used in 2 places. Its second occurence is dead since it is the argument of a constant function. Let us consider the following $F^{\Pi}_{\leq:}$ derivation of this program:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash + : \mathcal{N}^{\alpha_0} \to \mathcal{N}^{\alpha_0} \to \mathcal{N}^{\alpha_0}}\quad \dfrac{}{\Gamma \vdash x : \mathcal{N}^{\alpha_0}}}{\Gamma \vdash (+\ \ x) : \mathcal{N}^{\alpha_0} \to \mathcal{N}^{\alpha_0}}\quad \dfrac{\dfrac{\dfrac{}{\Gamma' \vdash 5 : \mathcal{N}^{\alpha_0}}}{\Gamma \vdash \lambda y : \mathcal{N}^{\alpha_1}.5 : \mathcal{N}^{\alpha_1} \to \mathcal{N}^{\alpha_0}}\quad \dfrac{}{\Gamma \vdash x : \mathcal{N}^{\alpha_2}}}{\Gamma \vdash (\lambda y : \mathcal{N}^{\alpha_1}.5\ \ x) : \mathcal{N}^{\alpha_0}}}{\Gamma \vdash (+\ \ x\ \ (\lambda y : \mathcal{N}^{\alpha_1}.5\ \ x)) : \mathcal{N}^{\alpha_0}}\ \Xi}{\vdash \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}^{\alpha_1}.5\ \ x)) : \mathcal{N}^{\alpha_0}}$$

where Ξ denotes a derivation tree for p (we suppose, for the sake of simplicity that Ξ generates no constraints on β). Let us call this program p^*_0. Now consider the following substitution:

$$S = [\alpha_0 \mapsto \top; \alpha_1 \mapsto \bot]$$

If we apply this substitution to the derivation tree of p^*_0, we obtain:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash + : \mathcal{N}^{\top} \to \mathcal{N}^{\top} \to \mathcal{N}^{\top}}\quad \dfrac{}{\Gamma \vdash x : \mathcal{N}^{\top}}}{\Gamma \vdash (+\ \ x) : \mathcal{N}^{\top} \to \mathcal{N}^{\top}}\quad \dfrac{\dfrac{\dfrac{}{\Gamma' \vdash 5 : \mathcal{N}^{\top}}}{\Gamma \vdash \lambda y : \mathcal{N}^{\bot}.5 : \mathcal{N}^{\bot} \to \mathcal{N}^{\top}}\quad \dfrac{}{\Gamma \vdash x : \mathcal{N}^{\alpha_2}}}{\Gamma \vdash (\lambda y : \mathcal{N}^{\bot}.5\ \ x) : \mathcal{N}^{\top}}}{\Gamma \vdash (+\ \ x\ \ (\lambda y : \mathcal{N}^{\bot}.5\ \ x)) : \mathcal{N}^{\top}}\ S(\Xi)}{\vdash \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}^{\bot}.5\ \ x)) : \mathcal{N}^{\top}}$$

Now, since $S(p^*_0)$ is of positive type, and since the second occurence of x in $S(p^*_0)$ is of negative type theorem 4.4 states that it may safely be replaced with

a dummy term. Let $p_{simpl} = \mathsf{let}\ x : \mathcal{N}^{\beta}\ \mathsf{be}\ p\ \mathsf{in}\ (+\ \ x\ \ (\lambda y : \mathcal{N}^{\perp}.5\ \ \mathtt{d}_{\mathcal{N}^{\perp}}))$, then $p_{simpl} \sqsubseteq p_0^*$ and is of type $\mathcal{N}^{\top}$.

This trivial example illustrates well the mechanisms of program analyzes in $F^{\Pi}_{\leq:}$. We left as exercise to the reader the following program:

$$p_1 \stackrel{def}{=} (\lambda F^{(\mathcal{N}\to\mathcal{N}\to\mathcal{N})\to\mathcal{N}\to\mathcal{N}\to\mathcal{N}}.(+\ \ (F\ \ \lambda x^{\mathcal{N}}, y^{\mathcal{N}}.x\ \ m\ \ n)\ \ (F\ \ \lambda x^{\mathcal{N}}, y^{\mathcal{N}}.y\ \ o\ \ q))\ \ \lambda f^{\mathcal{N}\to\mathcal{N}\to\mathcal{N}}, z^{\mathcal{N}}, v^{\mathcal{N}}.(f\ z\ v))$$

where m, n, o, q are any closed term of type $\mathcal{N}$. Types are given in superscript for notational convenience. Using $F^{\Pi}_{\leq:}$ it is possible to prove that n, q are dead code. It is impossible to prove it using [BB95]. Another example worth of consideration is the following one:

$$p_2 \stackrel{def}{=} (\lambda F^{\mathcal{N}\to\mathcal{N}} \to \mathcal{N} \to \mathcal{N}.(F\ \ \lambda y^{\mathcal{N}}.5\ \ m)\ \ \lambda f^{\mathcal{N}\to\mathcal{N}}, x^{\mathcal{N}}.(f\ \ (f\ \ x)))$$

where m is any closed term of type $\mathcal{N}$. Using $F^{\Pi}_{\leq:}$ it is possible to prove that m is dead code while it is impossible to prove it in [Pro97].

5 Conclusion

We have presented a variant of system F in which types are build on different universes. ML-Polymorphism at the level of universe as well as subtyping, derived from a basic inclusion between $\top$ and $\perp$, are available in $F^{\Pi}_{\leq:}$.We have shown the basic properties of this system : Strong Normalization, Confluence, stability of typing through substitutions, Subject Reduction and the existence of a most generic typing derivation. We have also exposed how it can be used to formalize program analyzes (construction of the most generic typing) and studied some applications for dead code and binding time.

Our work differs from earlier studies where polymorphism and/or subtyping are used for program analyzes. First, our underlying type system is a polymorphic λ-calculus, whereas previous works only consider simply typed lambda-calculi (except [Boe94] but there is nor polymorphism no subtyping in it). It makes possible to avoid *had-oc* typing rules for constants such as if then else construct, natural numbers, booleans and basic operations on them (like Successor,+ etc), through their impredicative encoding. Second, previous works are generally linked (excepted [Ber93, BB95, Boe94]) to a given reduction strategy. In our setting only β-reduction is considered. It makes our study valid under any reduction strategy.

From a theoretical point of view, this work provides an abstract approach of type based program analyses. It enlightens the relationships between type inference and program analyses. By few means (different universes and an inclusion rule between universes), we have shown that it is possible to formalize different analyses.

Possible further work includes the definition of a suitable semantics for $F^{\Pi}_{\leq:}$, based on the intuition developed in section 3. There are also no fundamental reasons to limit the number of universes to two. For strictness and totality analsis for instance one could imagin a variant of $F^{\Pi}_{\leq:}$ with three universes. Another direction is to try the idea of variable universes to F_ω or even the Calculus of Constructions. Indeed, the Calculus of Construction is the underlying type system of proof checkers as Coq (see [BBC+96]) and Lego (see [LP92]). These proof checkers allow to build programs as proof of their specifications through *extraction*. Extraction might more or less be seen as dead code elimination (which is not entirely true since extraction provides a proof that the extracted program verifies its specification). It is likely that our dead code analysis would improve existing extractions.

Acknowledgments

I thank Christine Paulin who suggested the inclusion $\bot \leq: \top$, I also thank Stefano Berardi for discussions where he gave me valuable intuitions about the implications of the inclusion for dead code analysis.

References

[BB95] S. Berardi and L. Boerio. Using subtyping in program optimization. In *Proceedings of TLCA'95, LNCS 902*. Spinger-Verlag, 1995.

[BBC+96] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliâtre, H. Herbelin, G. Huet, P. Manoury, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. *The Coq Proof Assistant Reference Manual Version 6.1*. INRIA-Rocquencourt-CNRS-ENS Lyon, December 1996.

[Ber93] S. Berardi. Pruning simply typed λ–terms. Technical report, Turin University, 1993.

[Boe94] L. Boerio. Extending pruning techniques to polymorphic second order λ–calculus. In D. Sanella, editor, *Proceedings of ESOP'94, LNCS 788*, pages 120–134. Springer-Verlag, April 1994.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages* (*POPL '77*), pages 238–252, New York, 1977. ACM Press.

[CDDK86] D. Clément, J. Despeyroux, T. Desperoux, and G. Kahn. A simple applicative language: Mini-ML. Technical Report 529, INRIA-Sophia Antipolis, May 1986.

[DHM95] D. Dussart, F. Henglein, and C. Mossin. Polymorphic recursion and subtype qualifications: Polymorphic binding-time analysis in polynomial. In Alan Mycroft, editor, *SAS'95: 2nd Int'l Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 118–135, Glasgow, Scotland, September 1995. Springer-Verlag.

[DP98] F. Damiani and F. Prost. Detecting and removing dead code using rank-2 intersection. In *International Workshop:"TYPES'96", selected papers, LNCS 1512*. Spinger-Verlag, 1998.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[Hei95] N. Heintze. Control-flow analysis and type systems. In Alan Mycroft, editor, *Proceeding of SAS 1995, LNCS 983*, pages 189–206. Springer-Verlag, 1995.

[HM94] C. Hankin and D. Le Métayer. A type-based framework for program analysis. In *Proceedings of the Static Analysis Symposium, LNCS 864*, pages 380–394. Springer-Verlag, 1994.

[Hun91] S. Hunt. *Abstract Interpretation of functionnal languages : from theory to Practice*. PhD thesis, Department of Computing, Imperial College, London, 1991.

[LP92] Z. Luo and R. Pollack. Lego proof development system : User's manual. Technical Report ECS-LFCS-92-211, University of Edinburgh., 1992.

[Mit90] J.C. Mitchell. A type inference approach to reduction properties and semantics of polymorphic expressions. In G. Huet, editor, *Logical Foundations of Functionnal programming*, pages 195–211. Addison-Wesley, 1990. (Chapter 9).

[NSN94] H.R. Nielson, K.L. Solberg, and F. Nielson. Strictness and totality analysis. In *Static Analysis, LNCS 864*, pages 408–422. Springer-Verlag, 1994.

[Pau89] C. Paulin. *Extraction de programmes dans le calcul des constructions*. PhD thesis, Université Paris 7, January 1989.

[PM89] C. Paulin-Mohring. Extracting F_ω's programs from proofs in the Calculus of Constructions. In *Sixteenth Annual ACM Symposium on Principles of Programming Languages*, Austin, January 1989. ACM.

[Pro97] F. Prost. Using ML type inference for dead code analysis. Research Report RR97-09, LIP, ENS Lyon, France, May 1997.

[Sol95] K. Lackner Solberg. *Annotated Type Systems for Program Analysis*. PhD thesis, Odense University, July 1995.

[TJ92] J.-P. Talpin and P. Jouvelot. The type and effect discipline. In IEEE Computer Society Press, editor, *Proceedings of the 1992 Conference on Logic in Computer Science*, 1992.