A Proof of Strong Normalization for F_2, F_ω and Beyond
(Draft)

∗ †D. McAllester
dam@ai.mit.edu

J. Kučan
kucan@theory.lcs.mit.edu

D. F. Otth
otth@theory.lcs.mit.edu

∗ MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139, U.S.A

MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, U.S.A.

September 20, 1994

---

†Research supported by Loral Federal Systems-Owego, Contract No. 475598

Abstract

We present an evaluation technique for proving strong normalization (SN). We use the technique to give SN proofs for F_2, F-bounded quantification, subtypes, and F_ω. The evaluation technique derives SN as a corollary of the soundness of the typing rules under an appropriate evaluation semantics. The evaluation semantics yields simpler type sets than those used in the earlier SN proofs. The type sets discussed here form a complete lattice under classical union and intersection. The simplified type sets allow a simplified treatment of a variety of type constructors.

1 Introduction

Consider a programming language without any form of iteration or recursion. One might expect that in such an apparently restricted language all programs must terminate. However, it is well known that *untyped* languages with higher order procedures can use self application to simulate recursion. Types change things—in most typed higher order languages self application is not allowed and recursion can not be simulated without a recursion primitive. A related observation is that under most type disciplines all recursion-free and iteration-free programs terminate. But many different type systems have been proposed in the literature and these different systems have different properties. This paper concerns the characterization of those type systems in which all iteration-free and recursion-free programs terminate. The concept of strong normalization provides a well known formal model of termination. A term (program) is strongly-normalizing (SN) if any reduction (simplification) process starting at that term must yield a final value no matter what order is selected for individual simplifications. We are interested in characterizing type disciplines in which all well typed terms have this strongly normalization property.

A well known typing system in which all terms are SN is the Girard/Reynolds second-order polymorphic lambda calculus, more concisely known as the system F_2 [Gir72, Rey74]. All terms typable in F_2 are SN in spite the fact that F_2 can type a limited form of self-application. The original proof of SN for terms typable in F_2 is due to Girard and was given simultaneously with his introduction of the system itself. Since then the proof has been simplified by a variety of authors, e.g., [Tai75, Mit86]. An extensive overview of the SN proofs for F_2 can be found in [Gal90].

The inference rules of F_2 (and other type systems) allow one to derive sequents of the form $\Sigma \triangleright M{:}\,\tau$ where Σ is a set of variable declarations, M is a term, and τ is a type expression. Intuitively, a type denotes a set of values and a formula of the form $M{:}\,\tau$ expresses the statement that the value of the term M is an element of the set denoted by the type τ. A sequent of the form $\Sigma \triangleright M{:}\,\tau$ is *valid* provided that, when free variables are interpreted so as to satisfy the conditions given in Σ, the formula $M{:}\,\tau$ is true.

In this paper we construct a *type inference model* (as defined in [Mit88]) for the system F_2. In other words we are only concerned with validity of the derivable typing sequents of F_2 and not with the validity of equations. Our inference model is a kind of a "term model", since a meaning of a term M is a (usually different) term N called the *value* of M. The types are interpreted as sets of values. With this simple, and as we later argue, natural interpretation, the SN property of F_2 typable terms is a simple corollary of the semantic soundness of the type inference rules.

Earlier presentations of proofs of SN for F_2 have also been based on showing the soundness of the inference rules under an appropriate semantics [Mit86]. However, in earlier proofs the judgment $M{:}\,\tau$ has meant that M, rather than the value of M, is a member of the set denoted by τ. This forces the type sets to be sets of unevaluated terms rather than sets of values. The main innovation here is that types are taken to denote sets of values rather than sets of terms. We call the semantics used here an *evaluation semantics* to distinguish it from the semantics used in earlier SN proofs. Our evaluation semantics is similar to that used in Scedrov's proof that the terms of F_2 have normal forms—a weaker condition than SN [Sce87].

We emphasize that, like in some earlier SN proofs [Mit86, Sce87], we work with the implicit type systems in the style of Curry (i.e. the types do not occur in terms). Additional work is needed to prove the SN property of the explicit versions of these systems. The reader is referred to [Gal90] for a full review of earlier proofs of SN for F_2.

Although the term model semantics presented here is designed to make SN a corollary of soundness, it is also a natural semantics in its own right. The evaluation process which defines the semantics is similar to standard implementations of a variety of programming languages. All of the technical conditions in the evaluation process have simple motivations. Also, by taking the types to be sets of values rather than sets of terms we are able to close the allowable types under arbitrary unions and intersections. This simplifies the treatment of F-bounded quantification and other extensions of F_2 presented below.

The main contribution of this paper lies in the proof technique based on an evaluation semantics. However, the SN result for a system with F-bounded quantification appears to be new. We are hopeful that the evaluation technique presented here will give a more streamlined method of establishing SN for novel systems.

2 Strong Normalization for System F_2

In this section we give a proof of SN for F_2 based on evaluation semantics. The proof consists of defining the semantics of typing sequents and showing that the inference rules of F_2 preserve the validity of typing sequents. Moreover, the evaluation semantics is such that any term assigned a type in a valid sequent must be strongly normalizing.

We consider the implicit version of F_2 in which types do not appear explicitly in terms. This formulation uses the set Λ of untyped lambda terms (or simply terms) which is generated by the grammar

$$\Lambda ::= \texttt{TOP} \mid x \mid (\Lambda\ \Lambda) \mid \lambda x.\Lambda,$$

where x stands for ordinary variables. We will also use y and z to denote ordinary variables. Terms will be denoted by N and M. The constant `TOP` has no *a priori* meaning, but in the context of our evaluation semantics, it can be thought of as the denotation of an error condition (e.g. "unbound variable"). Our semantics will also use `TOP` to insure that no type is empty. The set $\mathcal{T}$ of type expression, also called types, in F_2 is defined by the grammar

$$\mathcal{T} ::= t \mid \mathcal{T} \rightarrow \mathcal{T} \mid \forall t.\mathcal{T},$$

where t stands for type variables. Type variables will be denoted by r, s and t. Types will be denoted by σ and τ. We use familiar notions of bound and free variables and substitution. We will not distinguish between α-convertible λ-terms or types.

A *type judgment formula* is a formula of the form $N{:}\,\tau$ where N is a term and τ is a type. A *type assignment* is a finite set $\{x_1{:}\,\tau_1,\ x_2{:}\,\tau_2,\ \ldots,\ x_n{:}\,\tau_n\}$ of type judgment formulas where the variables x_i are all distinct. Finally, a *typing sequent* is an expression of the form $\Sigma \triangleright N{:}\,\tau$ where Σ is a type assignment and $N{:}\,\tau$ is a type judgment formula. We say that a typing sequent is derivable in F_2 if it can be derived using the following type inference rules.

$$\Sigma \triangleright x{:}\,\sigma, \qquad \text{if } x{:}\,\sigma \in \Sigma \tag{var}$$

$$\frac{\Sigma \triangleright M{:}\,\sigma \rightarrow \tau, \quad \Sigma \triangleright N{:}\,\sigma}{\Sigma \triangleright (M\ \ N){:}\,\tau} \tag{app}$$

$$\frac{\Sigma, x{:}\,\sigma \triangleright M{:}\,\tau}{\Sigma \triangleright (\lambda x.M){:}\,\sigma \rightarrow \tau} \tag{abs}$$

$$\frac{\Sigma \triangleright M{:}\,\forall t.\sigma}{\Sigma \triangleright M{:}\,\sigma[\tau/t]} \tag{inst}$$

$$\frac{\Sigma \rhd M\colon \sigma}{\Sigma \rhd M\colon \forall t.\sigma}, \qquad t \text{ is not free in } \Sigma \tag{gen}$$

If $\Sigma \rhd M\colon \sigma$ is derivable in F_2, we write $F_2 \vdash \Sigma \rhd M\colon \sigma$.

To prove that all terms typable in F_2 are SN we now define an evaluation semantics. The evaluation semantics used in our proofs all involve an evaluation function which is a partial map from terms to terms. The terms in the range of the evaluator are called values. Types are interpreted as sets of values.

Definition 1. An *evaluation function* is a function V from Λ to $\Lambda \cup \{\perp\}$ such that, for N in Λ, if $V[N] \neq \perp$ then $V[V[N]] = V[N]$. Terms which are the value of other terms, i.e., the elements of $\Lambda \cap V[\Lambda]$, are called *values.* If $V[M] = \perp$ then M is called *divergent.* An *evaluation semantics* is a pair $\langle V, \mathcal{D}\rangle$ where V is an evaluation function and $\mathcal{D}$ is a collection of sets of values. The sets in $\mathcal{D}$ are called *type sets.*

In the definitions given below we consider a fixed but arbitrary evaluation semantics.

Definition 2. A *type environment* is a mapping from type variables to elements of $\mathcal{D}$. The meaning of a type expression τ under type environment ρ, denoted as $[\![\tau]\!]\rho$, is defined inductively as follows:

$$
\begin{aligned}
[\![t]\!]\rho &= \rho(t) && (1)\\
[\![\sigma \to \tau]\!]\rho &= \{M \mid M \text{ is a value such that } \forall N \,.\, V[N] \in [\![\sigma]\!]\rho \supset V[(M\ N)] \in [\![\tau]\!]\rho\} && (2)\\
[\![\forall t.\sigma]\!]\rho &= \{M \mid \forall S \in \mathcal{D} \,.\, M \in [\![\sigma]\!]\rho[S/t]\} && (3)
\end{aligned}
$$

The following lemma gives us a useful property of the meaning function $[\![\cdot]\!]$. This property is used in the proof of the soundness of rule (inst).

Lemma 1. For any types σ and τ and any type environment ρ, $[\![\sigma]\!]\rho[([\![\tau]\!]\rho)/t] = [\![\sigma[\tau/t]]\!]\rho$.

Proof: By the induction on length of σ. ■

Note that the set $\mathcal{D}$ plays an important role in defining the semantics of types of the form $\forall t.\sigma$. The smaller (or "tighter") one makes the collection $\mathcal{D}$, the larger (or "looser") the meaning of quantified types. In general $\mathcal{D}$ may not be closed under $[\![\cdot]\!]$. In other words, for some τ, $[\![\tau]\!]\rho$ may not be an element of $\mathcal{D}$. However, we will only be concerned with evaluation semantics for which the meaning of types are elements of $\mathcal{D}$.

Definition 3. An *interpretation* is a pair $\langle s, \rho\rangle$ where s is a term substitution, i.e. a mapping from variables to terms, and ρ is a type environment. We say that an interpretation $\langle s, \rho\rangle$ *satisfies* a type judgment formula $N\colon \tau$, written $\langle s, \rho\rangle \models N\colon \tau$, if $V[s(N)]$ is a member of $[\![\tau]\!]\rho$. We say that an interpretation satisfies a typing assignment Σ if it satisfies every member of Σ, i.e. if $\langle s, \rho\rangle \models x\colon \sigma$ for all $x\colon \sigma \in \Sigma$. We say that a sequent $\Sigma \rhd M\colon \tau$ is *valid*, written $\Sigma \models M\colon \tau$, if every interpretation satisfying Σ also satisfies $M\colon \tau$.

A typing rule is *sound* if it preserves validity. More explicitly, the rule

$$\frac{\Sigma \rhd M\colon \tau}{\Sigma' \rhd M'\colon \tau'} \tag{r}$$

is sound iff $\Sigma \models M\colon\tau$ implies $\Sigma' \models M'\colon\tau'$. If, under a particular semantics, each inference rule of F_2 is sound, then all sequents derivable in F_2 must be valid relative to that semantics.

There is a variety of semantics under which the inference rules are sound. One interesting example is the semantics where $V[M]$ is defined to be the normal form of M, if one exists and $\perp$ otherwise, and elements of $\mathcal{D}$ are sets of normal forms containing all applications of variables. This semantics is used in Scedrov's proof of normalization for F_2 [Sce87]. However, this semantics is not sufficiently strong to establish that all the typable terms are SN. To prove SN for F_2 we need a "tighter" semantics.

Now we give conditions on the evaluation semantics under which the inference rules are sound. These conditions are satisfied by the particular semantics used in the SN proof for F_2 and by other evaluation semantics used in our proofs of SN for extensions of F_2. From this point forward we will only consider evaluation semantics satisfying the following conditions.

$$\texttt{TOP}\text{ is a value, and }\mathcal{D}\text{ is the collection of all sets }S\text{ of values such that }S\text{ contains }\texttt{TOP}. \tag{V0}$$

$$\text{If } V[M] \neq\perp \text{ and } V[N] \neq\perp \text{ then } V[(V[M]\ N)] = V[(M\ N)]. \tag{V1}$$

$$\text{If } V[N] \neq\perp \text{ then } V[((\lambda x.M)\ N)] = V[M[N/x]]. \tag{V2}$$

$$\text{If } V[\lambda x.M] =\perp \text{ then } V[M[\texttt{TOP}/x]] =\perp. \tag{V3}$$

$$\text{If } V[M] \neq\perp \text{ then } V[(\texttt{TOP} M)] = \texttt{TOP}. \tag{V4}$$

Recall that $\perp$ is not a value. Condition (V0) specifies that $\perp$ is not a member of any type. The motivation for this condition is that $\perp$ will be the meaning of a divergent term. It also says that an error (`TOP`) can occur at any type. Condition (V1) says that if both operator and operand are non-divergent, it is fine to evaluate the operator first. Condition (V2) tells us that if the operand is non-divergent then β-reduction is sound. Condition (V3) says that a λ-abstraction can diverge only in those cases where the body of that λ-abstraction diverges in a strong sense—the body must diverge in the case where the bound variable is interpreted as the "totally convergent" value `TOP`. Condition (V4) says that `TOP` is totally convergent. It can also be viewed as saying that if an error occurs in the evaluation of an operator, and the argument does not diverge, the application results in an error.

It is important that type expressions are interpreted as elements of $\mathcal{D}$. In other words $[\![\tau]\!]\rho$ should be an element of $\mathcal{D}$. For the collection of type sets defined by (V0) this is guaranteed by the following lemma.

Lemma 2. In any semantics satisfying (V0) and (V4), for any type environment ρ and type expression τ, we have $\texttt{TOP} \in [\![\tau]\!]\rho$ (and hence $[\![\tau]\!]\rho \in \mathcal{D}$).

Proof: Consider Definition 2 given earlier. Expressions (1) and (3) obviously define type sets. To see that expression (2) defines a type set, let $N \in [\![\sigma]\!]\rho$. Then, by condition (V4), $V[(\texttt{TOP}\ N)] = \texttt{TOP}$ which is in $[\![\tau]\!]\rho$ since $[\![\tau]\!]\rho$ is a type set. Therefore, $\texttt{TOP} \in [\![\sigma\rightarrow\tau]\!]\rho$ and $[\![\sigma\rightarrow\tau]\!]\rho$ is a type set. ■

We now have the following soundness lemmas.

Lemma 3. The typing rule (var) is sound under any semantics.

Lemma 4. The typing rule (app) is sound under any semantics satisfying (V1).

Proof: Let $\langle s, \rho\rangle \models \Sigma$. Then we have $V[s(M)] \in [\![\sigma \rightarrow \tau]\!]\rho$ and $V[s(N)] \in [\![\sigma]\!]\rho$. So both $V[s(N)] \neq \perp$ and $V[s(M)] \neq \perp$. By the definition of arrow types we have that $V[(V[M]\ N)] \in [\![\tau]\!]\rho$. Property (V1) implies $V[(M\ N)] \in [\![\tau]\!]\rho$ as required. ∎

Lemma 5. In any semantics satisfying (V0) through (V4) the typing rule (abs) is sound.

Proof: Suppose $\langle s, \rho\rangle \models \Sigma, x{:}\,\sigma$. Since by Lemma 2, $\texttt{TOP} \in [\![\sigma]\!]\rho$, $\langle s[\texttt{TOP}/x], \rho\rangle \models \Sigma, x{:}\,\sigma$, and $V[s[\texttt{TOP}/x](M)] \in [\![\tau]\!]\rho$. Therefore, by property (V3), $V[s(\lambda x.M)]$ is a value. Let N be such that $V[N] \in [\![\sigma]\!]\rho$. Then $\langle s[N/x], \rho\rangle \models \Sigma, x{:}\,\sigma$ and $V[s[N/x](M)] \in [\![\tau]\!]\rho$. Using properties (V1) and (V2), $V[(V[s(\lambda x.M)]\ N)] = V[(s(\lambda x.M)\ N)] = V[(s[x/x](M))[N/x]] = V[s[N/x](M)]$. Therefore $V[s(\lambda x.M)] \in [\![\sigma \rightarrow \tau]\!]\rho$. ∎

Lemma 6. In any semantics satisfying (V0) and (V4) the typing rule (inst) is sound.

Proof: To prove the soundness of the instantiation rule it suffices to show that if $M \in [\![\forall t.\sigma]\!]\rho$ then $M \in [\![\sigma[\tau/t]]\!]\rho$. By Lemma 2, in any semantics satisfying (V0) and (V4), we have that $[\![\tau]\!]\rho$ is a type set. By the definition of $[\![\forall t.\sigma]\!]\rho$ we then have that if $M \in [\![\forall t.\sigma]\!]\rho$ then $M \in [\![\sigma]\!]\rho[[\![\tau]\!]\rho/t]$. But by Lemma 1 $[\![\sigma]\!]\rho[[\![\tau]\!]\rho/t]$ equals $[\![\sigma[\tau/t]]\!]\rho$. ∎

Lemma 7. The typing rule (gen) is sound under any semantics.

We can now define a particular semantics satisfying conditions (V0) through (V4) such that SN for F_2 is a corollary of the above soundness lemmas. Other evaluation semantics satisfying (V0) through (V4) will be used for various extensions of F_2.

Condition (V0) specifies that $\mathcal{D}$ is the collection of all sets of values containing `TOP`. To define the evaluation function V we first define an evaluation relation on λ-terms using the following evaluation rules.

1. $(f\ N_1\ \ldots\ N_k) \rightarrow \texttt{TOP}$, provided that f is either a variable or the constant `TOP`.
2. $((\lambda\, x.M)\ N_1\ N_2\ \ldots\ N_k) \rightarrow (M[N_1/x]\ N_2\ \ldots\ N_k)$

The reductions rules only apply at the top level. They define a deterministic evaluation process—only a single reduction can be applied to any given term. We then define $V[M]$ to be the normal form of M under the above rules if M is SN and $\perp$ otherwise. We see that a value with respect to this evaluation function is either `TOP` or an SN λ-abstraction. To prove that this evaluation function satisfies conditions (V0) through (V4) we need the following property of SN terms.

Lemma 8. If N_1 is SN and $(M[N_1/x]\ N_2\ \ldots\ N_k)$ is SN, then $((\lambda x.M)\ N_1\ N_2\ \ldots\ N_k)$ is SN.

To our knowledge, this lemma is needed in all proofs of SN for F_2. We omit the proof here.

Lemma 9. The semantics defined above satisfies conditions (V0) through (V4) and hence the inference rules are sound under this semantics.

Proof: The only property which is not straightforward is (V1). To show (V1) it suffices to show that if M, N and $(V[M]\ N)$ are SN then $(M\ N)$ is SN. First suppose that $V[M]$ is a λ-abstraction. In this case the term $(V[M]\ N)$ can be derived from $(M\ N)$ by a series of applications of the

β-reduction evaluation rule (rule 2 above). Furthermore, since M and N are SN, every reduction step in the conversion of $(M\ N)$ to $(V[M]\ N)$ must be of the form

$$((\lambda x.M)\ N_1\ N_2\ \ldots\ N_k) \rightarrow (M[N_1/x]\ N_2\ \ldots\ N_k)$$

where N_1 is SN. The above lemma implies that each such reduction preserves divergence, i.e., if the left hand side of the above rewrite is not SN then the right hand side is also not SN. If each rewrite step preserves divergence, then the whole rewrite chain preserves divergence. So if $(V[M]\ N)$ is SN then $(M\ N)$ is SN. In the case where $V[M]$ is `TOP` we have that $(M\ N)$ rewrites to an SN term of the form $(\mathtt{TOP}\ N_1\ \ldots\ N_k)$ using rewrite steps that preserve divergence. ■

Theorem 1 (Strong Normalization for F_2). If the sequent $\Sigma \triangleright M\colon \sigma$ is derivable in F_2 then M is SN.

Proof: Since all provable sequents are valid under the semantics given here, we have that $V[s(M)] \in [\![\sigma]\!]\rho$ for any term substitution s and type environment ρ satisfying Σ. By taking s to be the substitution $s_{\mathtt{TOP}}$, taking every variable to `TOP`, we have that $V[s_{\mathtt{TOP}}(M)] \neq \perp$ and hence $s_{\mathtt{TOP}}(M)$ is SN. This implies that M is also SN. ■

Remark. The evaluation function used in the proof is very simple, but it belongs to quite general a class of evaluation functions. Notice that reduction strategy that defines V never reduces any redexes in the scope of a λ-abstraction. In fact, every reduction strategy that has this property yields an evaluation function satisfying (V1) through (V4). Moreover, such a reduction strategy is a very natural one—any reasonable interpreter will not evaluate under λ-abstractions.

3 Extensions of F_2

The above results for the second-order polymorphic λ-calculus can easily be generalized to more powerful typing disciplines. Let us assume we are given the semantics $\langle V, \mathcal{D}\rangle$ defined in the previous section. Note that $\mathcal{D}$ is a complete lattice with respect to set inclusion and is closed under arbitrary intersections and unions. This allows us to interpret intersection types as finite intersections, union and existential types as finite and infinite unions. We can also interpret product types in a manner similar to the treatment of functional types. In this way we get immediate proofs of SN for these extensions of F_2. Because of the modular nature of the proof, we get SN for any combination of these features.

These are well known results (for example for intersection types see [vB92], for universal and existential quantification plus products see [Gal93]), so we will not describe the proofs in great length. We will only illustrate how the proof method generalizes to these systems.

Yet another typing discipline for which our proof has an easy generalization is the system F_2 with subtyping and F-bounded quantification, call it $F_{2\leq}$. To our knowledge, the SN theorem for this system has not been published, so we present it here. However, we note that since the first draft of this paper, Ghelli [Ghe93] has informed us of his manuscript on this theorem.

3.1 Intersections, Unions and Existential Quantification

Let us extend the set of types $\mathcal{T}$ with expressions of the form $\mathcal{T} \cap \mathcal{T}$, $\mathcal{T} \cup \mathcal{T}$ and $\exists t.\mathcal{T}$. These new types have a natural interpretation in our semantics. Namely let

$$\begin{array}{rcl} [\![\sigma \cap \tau]\!]\rho & = & [\![\sigma]\!]\rho \cap [\![\tau]\!]\rho \\ [\![\sigma \cup \tau]\!]\rho & = & [\![\sigma]\!]\rho \cup [\![\tau]\!]\rho \\ [\![\exists t.\tau]\!]\rho & = & \{M \mid \exists S \in \mathcal{D}\ .\ M \in [\![\tau]\!]\rho[S/t]\} \end{array}$$

Given the semantics $\langle V, \mathcal{D}\rangle$ satisfying conditions (V0)–(V4), it is immediate that the introduction and elimination rules for these types are sound. Just for illustration, consider the rule

$$\frac{\Sigma \triangleright M\colon \sigma, \quad \Sigma \triangleright M\colon \tau}{\Sigma \triangleright M\colon \sigma \cap \tau} \tag{$\cap$I}$$

Obviously, if $\langle s, \rho\rangle \models \Sigma$, then $V[s(M)]$ is in both $[\![\sigma]\!]\rho$ and $[\![\tau]\!]\rho$, so it must be in the intersection of the two.

3.2 Products and Sums

The addition of product and sum types is slightly more involved, since it requires the addition of not only new type expressions and typing rules, but also new term constructors and new reduction rules.

The set $\mathcal{T}$ of types is extended with the expressions of the form $\mathcal{T} \times \mathcal{T}$. The set Λ of untyped λ-terms is extended with the terms of the form $\langle\Lambda, \Lambda\rangle$ as well as with constants π_1 and π_2 denoting left and right projections. The reduction rules for projections are

$$(\pi_1\ \langle M, N\rangle) \rightarrow_\pi M \qquad (\pi_2\ \langle M, N\rangle) \rightarrow_\pi N \tag{π}$$

To make the pairs surjective, one can also add the reduction rule

$$\langle \pi_1\ M, \pi_2\ M\rangle \rightarrow_{\langle\cdot,\cdot\rangle} M \tag{$\langle\cdot,\cdot\rangle$}$$

The above rules are used in defining the notion of SN for systems with products. The rule $\langle\cdot,\cdot\rangle$ is optional—we will show that the system is SN when the rule is included and it immediately follows that the system is SN without the rule.

The usual typing rules for products are

$$\frac{\Sigma \triangleright M\colon \sigma, \quad \Sigma \triangleright N\colon \tau}{\Sigma \triangleright \langle M, N\rangle\colon \sigma \times \tau} \tag{$\times$I}$$

$$\frac{\Sigma \triangleright M\colon \sigma \times \tau}{\Sigma \triangleright (\pi_1\ M)\colon \sigma} \quad \frac{\Sigma \triangleright M\colon \sigma \times \tau}{\Sigma \triangleright (\pi_2\ M)\colon \tau} \tag{$\times$E}$$

The meaning of a product type is defined similarly to the meaning of functional types. In particular we define

$$[\![\sigma \times \tau]\!]\rho = \{M \mid M \text{ is a value, and } V[\pi_1\ M] \in [\![\sigma]\!]\rho,\ V[\pi_2\ M] \in [\![\tau]\!]\rho\}$$

If we think of `TOP` being an error condition, it is natural to identify $(\pi_i\ \texttt{TOP})$ with `TOP`—both components of an error are still errors. Recall that the evaluation rules only apply at the top level of expressions. This allows the rules control where evaluation can be performed. We add the following rules to the evaluation rules 1 and 2 defined in Section 2.

3. if $M \rightarrow M'$ then $((\pi_i\ M)\ N_1\ \ldots\ N_k) \rightarrow ((\pi_i\ M')\ N_1\ \ldots\ N_k)$, $i \in \{1,2\}$.
4. $(\pi_i\ \texttt{TOP}\ N_1\ \ldots\ N_k) \rightarrow \texttt{TOP}$, $i \in \{1,2\}$
5. $(\pi_i\ \langle M_1, M_2\rangle\ N_1\ \ldots\ N_k) \rightarrow (M_i\ N_1\ \ldots\ N_k)$, $i \in \{1,2\}$.

As in section 2, we define $V[N]$ to be $\perp$ if N is not SN and the normal form of N under the above evaluation rules otherwise. It is not hard to see that $\texttt{TOP} \in [\![\sigma \times \tau]\!]\rho$. Therefore $[\![\sigma \times \tau]\!]\rho$ is a type set. Furthermore, the properties (V0)–(V4) still hold for V with respect to this more general evaluation reduction, and by lemmas 3 though 7 the F_2 typing rules (var)–(gen) are still sound. In order to prove the SN property for the extension with product types it therefore suffices to show soundness of the typing rules ($\times$I) and ($\times$E) which is straightforward. The sum types can be treated in the similar manner as the product types.

3.3 Subtyping and F-Bounded Quantification

Subtyping is a partial order $\leq$ on types. The intended meaning of $\sigma \leq \tau$ is that σ denotes a subset of τ. In addition to considering subtype relationships we consider bounded quantification in types, i.e., types of the form $\forall t{\leq}\mathcal{T}.\mathcal{T}$. These bounded types can be use to express the unbounded quantification provided that we add a special type Top so that for any type σ, $\sigma \leq \mathit{Top}$. Then the type $\forall t.\sigma$ becomes equivalent to $\forall t{\leq}\mathit{Top}.\sigma$. The typing rules now combine two different relations: ":"—the typing relation and "$\leq$"—the subtyping relation. Therefore, we need to extend the set of type assignments Σ with pairs of the form $t \leq \sigma$, where all t's occurring in Σ are distinct type variables and σ's are types. In addition to rules (var), (app) and (abs) we have the following set of rules where rules (inst′) and (gen′) replace rules (inst) and (gen).

$$\Sigma \rhd t \leq \sigma, \qquad \text{if } t \leq \sigma \in \Sigma \qquad (\mathrm{var}_{\leq})$$

$$\Sigma \rhd \sigma \leq \mathit{Top} \qquad (\mathit{Top}_{\leq})$$

$$\frac{\Sigma \rhd \sigma \leq \sigma', \quad \Sigma \rhd \tau \leq \tau'}{\Sigma \rhd \sigma' {\to} \tau \leq \sigma {\to} \tau'} \qquad (\to_{\leq})$$

$$\frac{\Sigma, t \leq \tau' \rhd \tau' \leq \tau, \quad \Sigma, t \leq \tau' \rhd \sigma \leq \sigma'}{\Sigma \rhd \forall t{\leq}\tau.\sigma \leq \forall t{\leq}\tau'.\sigma'} \qquad (\forall_{\leq})$$

$$\frac{\Sigma \rhd M{:}\,\sigma, \quad \Sigma \rhd \sigma \leq \tau}{\Sigma \rhd M{:}\,\tau} \qquad (\mathrm{subsumption})$$

$$\frac{\Sigma \rhd M{:}\,\forall t{\leq}\tau.\sigma, \quad \Sigma \rhd \tau' \leq \tau}{\Sigma \rhd M{:}\,\sigma[\tau'/t]} \qquad (\mathrm{inst}')$$

$$\frac{\Sigma, t \leq \tau \rhd M{:}\,\sigma}{\Sigma \rhd M{:}\,\forall t{\leq}\tau.\sigma}, \qquad t \text{ is not free in } \Sigma \qquad (\mathrm{gen}')$$

In addition to the above rules we have that $\leq$ is reflexive and transitive. If our typing system contains intersections, unions or products, we add subtyping rules that make these types monotone in both arguments with respect to $\leq$.

Following our intuition about semantics of $\leq$, it seems natural to interpret $\leq$ as the subset relation on $\mathcal{D}$. Therefore the interpretation of bounded quantifiers becomes

$$[\![\forall t{\leq}\tau.\sigma]\!]\rho = \{M \mid \forall S \in \mathcal{D}\,.\, S \subseteq [\![\tau]\!]\rho[S/t] \supset M \in [\![\sigma]\!]\rho[S/t]\}.$$

The obvious choice for $[\![\mathit{Top}]\!]\rho$ is $\bigcup \mathcal{D}$. Similarly, for a subtyping formula $\sigma \leq \tau$ an interpretation $\langle s, \rho \rangle$ satisfies $\sigma \leq \tau$, written $\langle s, \rho \rangle \models \sigma \leq \tau$ iff $[\![\tau]\!]\rho \subseteq [\![\sigma]\!]\rho$. The soundness of each of the inference rules under the evaluation semantics given in section 2 is straightforward.

Theorem 2. If the sequent $\Sigma \rhd M{:}\,\sigma$ is derivable in system $F_{2\leq}$, the term M is strongly normalizing.

4 SN for F_ω

System F_ω is a natural extension of system F_2 in which the type expressions are enriched to include applications of λ-terms. F_ω includes λ-terms which take types as arguments and return types as values. More precisely, the system F_ω involves four groups of expressions—terms, types, kinds, and type constructors. The terms and types of F_ω are directly analogous to the terms and types of F_2. The types are expressions of kind $\star$. Higher kinds are constructed from $\star$ and $\Rightarrow$. For example $\star{\Rightarrow}\star$ is the kind of expressions which map types to types. Technically the types are a subset of the type constructors—the type constructors are those expressions which can be assigned a kind and the types are the subset of the type constructors which are of kind $\star$.

Formally, the set of type constructors will be defined as the *simply-kinded* λ-calculus with special constants $\rightarrow$ and $\forall_\kappa$ for each kind κ. There is a special (base) kind $\star$ of types. The set $\mathcal{K}$ of kinds is defined by the grammar

$$\mathcal{K} ::= \star \mid \mathcal{K}{\Rightarrow}\mathcal{K}.$$

We define type constructors $\mathcal{T}_\kappa$ of kind κ by the following grammar

$$\begin{array}{rcl} \mathcal{T}_\star & ::= & t^\star \mid (\mathcal{T}_{\kappa\Rightarrow\star}\ \mathcal{T}_\kappa) \mid \mathcal{T}_\star{\rightarrow}\mathcal{T}_\star \mid \forall_\kappa \mathcal{T}_{\kappa\Rightarrow\star} \\ \mathcal{T}_{\kappa_1\Rightarrow\kappa_2} & ::= & t^{\kappa_1\Rightarrow\kappa_2} \mid (\mathcal{T}_{\kappa\Rightarrow(\kappa_1\Rightarrow\kappa_2)}\ \mathcal{T}_\kappa) \mid \lambda t^{\kappa_1}.\mathcal{T}_{\kappa_2}, \end{array}$$

where t^κ stands for type constructor variables of kind κ. We will write $\sigma{:}\,\kappa$ to abbreviate $\sigma \in \mathcal{T}_\kappa$, i.e., that σ is of kind κ. For clarity, we use $\rightarrow$ and $\forall_\kappa$ as constructors in type expressions, rather than as special constants.

We also define equality among type constructors of the same kind by the usual β-η-equational rules for simply-typed lambda calculus. We write $\sigma =_\lambda \sigma'$ if σ and σ' can be proven equal using these equational rules.

As with F_2, the typing rules involve sequents whose antecedents are assignments of types to ordinary variables. We let Σ range over sets of the form $\{x_i{:}\,\sigma_i\}$ where each x_i is an ordinary variable, all x_i are distinct, and $\sigma_i{:}\,\star$. We use typing rules (var), (app) and (abs) of F_2 and we replace rules (inst) and (gen) with (inst$''$) and (gen$''$) and add rule (type-conv).

$$\frac{\Sigma \triangleright M{:}\,\forall_\kappa\sigma}{\Sigma \triangleright M{:}\,(\sigma\ \tau)}, \qquad \text{provided } \tau{:}\,\kappa \qquad (\text{inst}'')$$

$$\frac{\Sigma \triangleright M{:}\,\sigma}{\Sigma \triangleright M{:}\,\forall_\kappa(\lambda t^\kappa.\sigma)}, \qquad t \text{ is not free in } \Sigma \qquad (\text{gen}'')$$

$$\frac{\Sigma \triangleright M{:}\,\sigma}{\Sigma \triangleright M{:}\,\sigma'}, \qquad \text{if } \sigma =_\lambda \sigma' \qquad (\text{type-conv})$$

We will now prove that if $\Sigma \triangleright M{:}\,\sigma$ is derivable in F_ω then the term M is SN. As with F_2 we do this by giving a natural semantics to the terms and types such that SN is a corollary of the soundness of the inference rules. We assume we are given a semantics $\langle V, \mathcal{D}_\star\rangle$ satisfying conditions (V0) through (V4) of Section 2. We write $\mathcal{D}_\star$, instead of $\mathcal{D}$ to indicate that types will be interpreted as elements of $\mathcal{D}_\star$. More generally, for each kind κ we will define a set $\mathcal{D}_\kappa$ such that expressions of kind κ are interpreted as elements of the set $\mathcal{D}_\kappa$. Namely, we take the set $\mathcal{D}_{\kappa_1\Rightarrow\kappa_2}$ to be the simple set-theoretic space of functions from $\mathcal{D}_{\kappa_1}$ to $\mathcal{D}_{\kappa_2}$.

Definition 4. A *type environment* is a mapping ρ of type variables into $\bigcup_\kappa \mathcal{D}_\kappa$ such that $\rho(t^\kappa) \in \mathcal{D}_\kappa$. For any type constructor σ and type environment ρ we define $[\![\sigma]\!]\rho$ inductively as follows:

$$\begin{aligned}
[\![t^\kappa]\!]\rho &= \rho(t) && (4)\\
[\![(\sigma\ \tau)]\!]\rho &= ([\![\sigma]\!]\rho)([\![\tau]\!]\rho) && (5)\\
[\![(\lambda t^\kappa.\sigma)]\!]\rho &= f \in \mathcal{D}_{\kappa \Rightarrow \kappa'} \text{ such that } \forall v \in \mathcal{D}_\kappa \ .\ f(v) = [\![\sigma]\!]\rho[v/t^\kappa] && (6)\\
[\![\sigma \rightarrow \tau]\!]\rho &= \{M \mid M \text{ is a value, and } \forall N \ .\ V[N] \in [\![\sigma]\!]\rho \supset V[(M\ N)] \in [\![\tau]\!]\rho\} && (7)\\
[\![\forall_\kappa \sigma]\!]\rho &= \bigcap_{v \in \mathcal{D}_\kappa} ([\![\sigma]\!]\rho)(v) && (8)
\end{aligned}$$

Lemma 10. For any evaluation semantics $\langle V, \mathcal{D}_\star \rangle$ satisfying (V0) through (V4) we have that if $\sigma{:}\,\kappa$ then $[\![\sigma]\!]\rho$ is a member of $\mathcal{D}_\kappa$.

Proof: The proof is by induction on the structure of σ. The case of $[\![t^\kappa]\!]\rho$ follows from the definition of ρ. Now consider $[\![(\sigma\ \tau)]\!]\rho$. In this case we must have $(\sigma\ \tau){:}\,\kappa$ for some κ. This implies that there is some unique kind κ' such that σ is of kind $\kappa' \Rightarrow \kappa$ and τ is of kind κ'. So by the induction hypothesis $[\![\sigma]\!]\rho$ is a function with $[\![\tau]\!]\rho$ is in its domain and $[\![(\sigma\ \tau)]\!]\rho$ is in $\mathcal{D}_\kappa$. The case of $[\![(\lambda t^\kappa.\sigma)]\!]\rho$ is also straightforward. For the final two cases we note that $\mathcal{D}_\star$ is the set of value sets continuing `TOP`. As noted in Section 2 the set $\mathcal{D}_\star$ is closed under the formation of arrow types and under arbitrary intersections. This guarantees that $[\![\sigma \rightarrow \tau]\!]\rho$ and $[\![\forall_\kappa \sigma]\!]\rho$ are both members of $\mathcal{D}_\star$.

Lemma 11. If $\sigma =_\lambda \sigma'$ then $[\![\sigma]\!]\rho = [\![\sigma']\!]\rho$.

Proof: The inference rules of β and η conversion are sound under the set-theoretic semantics of the simply kinded type constructor expressions.

We now have a soundness lemma analogous to that given in Section 2.

Lemma 12 (Soundness for F_ω). For any evaluation semantics $\langle V, \mathcal{D}_\star \rangle$ satisfying (V0) through (V4) the typing rules (var), (app), (abs), (inst'), (gen') and (type-conv) are sound.

Proof: First we observe that conditions (V0) and (V4) guarantee that for each type constructor σ, of kind κ, $[\![\sigma]\!]\rho$ is well defined, and an element of $\mathcal{D}_\kappa$. Axiom (var) is trivially sound. The soundness of rules (app) and (abs) is proved as in lemmas 4 and 5. We now show soundness of the remaining rules.

Rule (inst$''$): Assume $\Sigma \models M{:}\,\forall_\kappa \sigma$, and that τ is a type constructor of kind κ. Let $\langle s, \rho \rangle \models \Sigma$. Then $V[s(M)] \in [\![\forall_\kappa \sigma]\!]\rho = \bigcap_{v \in \mathcal{D}_\kappa} ([\![\sigma]\!]\rho)(v)$. In particular, for $v = [\![\tau]\!]\rho$, $V[s(M)]$ is in $([\![\sigma]\!]\rho)([\![\tau]\!]\rho) = [\![(\sigma\ \tau)]\!]\rho$.

Rule (gen$''$): Assume $\Sigma \models M{:}\,\sigma$, and let $\langle s, \rho \rangle \models \Sigma$. Then $V[s(M)] \in [\![\sigma]\!]\rho$. For any $v \in \mathcal{D}_\kappa$, $\rho[v/t^\kappa]$ also respects kinds, so we have $V[s(M)] \in [\![\sigma]\!]\rho[v/t^\kappa] = ([\![\lambda t^\kappa.\sigma]\!]\rho)(v)$, and therefore $V[s(M)]$ is in $\bigcap_{v \in \mathcal{D}_\kappa} ([\![\lambda t^\kappa.\sigma]\!]\rho)(v)$, which by definition is $[\![\forall_\kappa (\lambda t^\kappa.\sigma)]\!]\rho$.

Rule (type-conv) is sound by Lemma 11. ■

Finally we prove SN for F_ω by using the evaluation function V defined in Section 2. By definition this evaluation function has the property that if $V[M] \neq \bot$ then M is SN. The soundness of the rules for F_ω, together with the fact that all ordinary variables can be interpreted as `TOP` to satisfy any Σ, immediately gives the following SN theorem for F_ω.

Theorem 3 (Strong Normalization for F_ω). If $\Sigma \rhd M{:}\,\sigma$ is derivable in F_ω then M is strongly normalizing.

It seems worth noting that our proof of SN for F_ω relies on the use of classical set-theoretic semantics for type constructors of higher kind. This allows the semantics defined in Section 2 to be integrated with the semantics of higher kind expressions while preserving the validity of β and η conversion of type constructors.

4.1 Extensions of F_ω

Since the domains $\mathcal{D}_{\kappa_1 \Rightarrow \kappa_2}$ are interpreted as full set-theoretic function spaces, we can easily interpret the type constructors described in Section 3. Therefore, using the same arguments as for F_2, we can show that F_ω can be extended in all the ways discussed in Section 3 while still preserving the strong normalization property.

The systems F_2 and F_ω can also be extended with safe recursive types while preserving SN. First consider a system with general recursive types introduced via type constructor μ_κ of kind $(\kappa \Rightarrow \kappa) \Rightarrow \kappa$ and the equational axiom schema

$$\Sigma \rhd (\mu_\kappa\ \sigma) = (\sigma_\kappa\ (\mu_\kappa\ \sigma)){:}\,\kappa \qquad \text{(Fix)}$$

The terms typable in this system are not necessarily strongly normalizing (Y is typable). Our proof of SN fails because there is no way of assigning a meaning (as a set of values) to non-monotone recursive types such as $\mu_{\star \Rightarrow \star}\ (\lambda\ t^\star.t \rightarrow t)$. Also, it not possible to give meanings to non-monotone recursive expressions at higher kinds. However, if we only allow syntactically monotone recursion then we can assign a set-theoretic meaning to all recursive expressions and our proof of SN goes through. More specifically we only consider expressions of the form $\mu_{\kappa \Rightarrow \kappa}\ (\lambda t^\kappa.\tau)$ and we require that all occurrences of t^κ in τ appear inside an even number of left hand sides of the type constructor $\rightarrow$.

5 Conclusions

Our proof technique for establishing strong normalization for type systems is based on the use of an evaluation semantics in defining the meaning of terms and types. The proof technique is actually very similar to earlier methods. The primary difference is that we use evaluation in defining the meaning of terms and this allows our type sets to be arbitrary sets of values containing the special value `TOP`. In earlier approaches the type sets were taken to be set of terms closed under an "inverse β reduction" principle. The evaluation of terms eliminates the need for inverse β reduction in the type sets allowing a greatly simplified semantic notion of type. To achieve this simplification we impose conditions (V0)–(V4) on the evaluation function. However, we find these conditions more intuitive and better motivated than the closure conditions found in earlier proofs.

The simplified notion of type allows a simplified treatment of the semantics of types. More specifically, it becomes easier to show that the meaning of a type expression is a semantic type set — one does not need to show that the set is closed under inverse reduction. This reduces the length of SN proofs in systems with many type constructs. We hope that the evaluation approach will reduce the effort required in constructing reliable SN proofs in the future.

References

[Gal90] Jean Gallier. On Girard's "Candidats de Reductibilité". In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 123–203. Academic Press, 1990.

[Gal93] Jean Gallier. On the correspondence between proofs and λ-terms. Available by ftp or on WWW as `ftp://ftp.cis.upenn.edu/pub/papers/gallier/cahiers2.dvi.Z.`, January 1993.

[Ghe93] Giorgio Ghelli. Termination of system f-bounded. Available by anonymous ftp or on WWW as `ftp://ftp.di.unipi.it/pub/Papers/ghelli/F-bounded.Term.ps.Z.`, November 1993.

[Gir72] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieure.* PhD thesis, Université Paris VII, 1972.

[Mit86] John C. Mitchell. A type inference approach to reduction properties and semantics of polymorphic expressions. In *Proceedings 1986 ACM Symposium on Lisp and Functional Programming*, pages 308–319, 1986.

[Mit88] John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76:211–249, 1988.

[Rey74] John C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium: Proceedings, Colloque sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, 1974.

[Sce87] Andre Scedrov. Normalization revisited. In *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 357–369. American Mathematical Society, 1987.

[Tai75] William W. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloqium, '73*, volume 453, pages 22–37. Springer-Verlag, 1975.

[vB92] Steffen van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102:153–136, 1992.