Strong Normalization for System F

Notes for the TAPL reading group

Primarily based on *Proofs and Types* by J.Y. Girard, but with different notation and details filled in.

See also the thesis *On Girard's "Candidats de Reductibilité"* by Jean H. Gallier for history of and variations on this proof and proof technique.

Introduction

Some typed λ-calculi have the property that *every well-typed program in them halts*. Here, "program" is to be interpreted loosely, we really want to talk about well-typed (potentially *open*) terms *t*. Given the single-step reduction relation ⇒, we say that a term is:

- **In normal form** if there is no t' such that t ⇒ t'
- **Weakly Normalizing** if there exists a sequence of reductions t ⇒ … ⇒ t' such that t' is a normal form.
- **Strongly Normalizing** if all such sequences terminate in a normal form.

We'll prove Strong Normalization for the Simply-typed Lambda Calculus, and then extend the technique (that of *Reducibility Candidates*) to cover System F.

Simply-Typed Lambda Calculus (STLC)

Letting *x* range over variables and *X* range over the name of a fixed set of *atomic base types*. STLC types and terms are defined by the following grammar:

T ::= *X* | T → T

t ::= *x* | t t | λ *x*:T. t

Typing Rules for STLC

> Γ, x : T
>
> ---------
>
> x : T

> Γ, x : T ⊢ t : S; Γ ⊢ x : T
>
> ------------------------------
>
> λ x:T. t : T → S

> Γ ⊢ t : S → T, t' : S
>
> ----------------------
>
> Γ ⊢ t t' : T

Note on contexts and conventions

Following Girard, all terms are going to be open: they will have free variables and type variables in an ambient context of sorts. We will simply refer to variables having a particular type.

The fact that these terms are open ends up being crucial to many of the steps in the proof. As does the fact that we can (e.g.) reduce inside a lambda.

Reduction Rules for STLC

> (λ *x*:T. s) t ⇒ s[t/*x*]

> t ⇒ t'
>
> -----
>
> t s ⇒ t' s

> s ⇒ s'
>
> -----
>
> t s ⇒ t s'

> s ⇒ s'
>
> -----
>
> λ *x*:T. s ⇒ λ *x*:T. s'

Normalization: Take 1

Proceed via induction on the terms of the STLC:

- Base case: variables are strongly normalizing.
- Inductive case 1: λ abstractions are strongly normalizing if their bodies are.
- Inductive case 2: take an application t t', assume t, t' SN ...

The argument breaks down here. The fact that t and t' are strongly normalizing does not give us enough information to finish the argument. The resulting term may be larger than what we started with

Indeed, any technique we use needs to take types into account. The untyped λ calculus is *not* strongly normalizing, but its reduction relation is more or less the same as the STLC's.

Intuition:

- The types give us a lot of structure. We need to formulate that structure in terms of a stronger inductive hypothesis so our induction can go through.
- The key insight for STLC is that **functions take halting inputs to halting outputs**. Note that this is **not** true of UTLC, where functions like

> ω = λ x. x x

take halting inputs (like ω itself) to non-halting outputs (Ω = ω ω).

This property of the STLC requires proof and is surprisingly tricky to write down, but once properly formulated it forms the key to establishing strong normalization.

Normalization for STLC: Roadmap

Three main tasks:

1. Define a relation R relating types to *reducible terms*, a stronger condition than strong normalization.
2. Prove important lemmas about how R is closed under the reduction relation in various ways.
3. Use these lemmas to show all terms in the STLC are in R and hence SN.

Defining *R*

R is defined recursively on types.

- $R[X] = \{$ strongly normalizing terms of type X $\}$
- $R[S \to T] = \{$ all terms t such that $\forall\, s \in R[S],\ t\ s \in R[T]$ $\}$

Note that *R*[T] is nonempty: it always contains (free) variables of type T.

Properties of *R*

Given a term t : T We can state the following properties of *R*[T]

(CR1) If t ∈ *R*[T], then t is strongly normalizing.

(CR2) If t ∈ *R*[T] and t ⇒ t', then t' ∈ *R*[T]

(CR3) If t is not a λ-abstraction and t' ∈ *R*[T] for all t ⇒ t', then t ∈ *R*[T]

> (NB for the literature: If t is not an abstraction we say it is *neutral*)

We will prove this by induction on the structure of the type T.

Comments on CR 1-3

Girard expresses some disappointment in how these rules feel arbitrary:

> The choice of (CR 1-3) is crucial. We need to identify some useful induction hypotheses on a set of terms which is otherwise arbitrary, and they must be preserved by the construction of the “true reducibility” . These conditions were originally found by trial and error. In linear logic, reducibility candidates appear much more naturally, from a notion of orthogonality on terms …

There is some recent work claiming to use a more category-theoretic framing of these questions to provide a settings in which these definitions fall out more naturally. I am still a ways away from making sense of it though.

Proof of CR1-2 for Base Types *X*

> **(CR1)** If t ∈ *R*[T], then t is strongly normalizing.

For base types, *R* coincides with the strongly normalizing terms by definition. We therefore get **(CR1)** without any extra work.

> **(CR2)** If t ∈ *R*[T] and t ⇒ t', then t' ∈ *R*[T]

If t is SN then all chains of reductions originating at t are finite. The same is therefore true of any t' where t ⇒ t', so we conclude that t' ∈ *R*[*X*].

Proof of CR3 for Base Types *X*

> **(CR3)** If t is not a λ and t' ∈ *R*[T] for all t⇒t', then t ∈ *R*[T]

If all t' directly reachable from t are strongly normalizing, then all chains of reduction starting from t go through a strongly normalizing term. All such chains must be finite, so t must also be strongly normalizing.

Proof of CR1-2 for Functions S → T

Let t be a term of type S → T.

> **(CR1)** If t ∈ *R*[T], then t is strongly normalizing.

We want to show that there is no infinite chain of reductions starting at t. Consider some term s ∈ *R*[S]. By the definition of *R*[S → T] we know that t s is strongly normalizing. That means in particular that there is no infinite chain t ⇒ t' ⇒ …, because then there would also be an infinite chain t s ⇒ t' s ⇒ ….

> **(CR2)** If t ∈ *R*[T] and t ⇒ t', then t' ∈ *R*[T]

Let s ∈ *R*[S]. Then t s is reducible. This means t' s is *also* reducible by the IH because t s ⇒ t' s. That's enough to give us t' ∈ *R*[S → T].

Proof of CR3 for Functions S → T

Let t be a term of type S → T.

> **(CR3)** If t is not a λ-abstraction and t' ∈ *R*[T] for all t ⇒ t', then t ∈ *R*[T]

Let s ∈ *R*[S]. We want to show that t is reducible assuming all t' it steps to are, which amounts to showing that t s is reducible. Note that s can only reduce some finite number of times *n*. We prove this by induction on *n* that all terms t s steps to are reducible (note that such an induction implicitly uses CR2):

Proof of CR3 for Functions S → T (cont.)

- t s ⇒ t' s, which is reducible because t' is reducible by assumption.
- t s ⇒ t s', where s' can only reduce *n*-1 times. The inductive hypothesis allows us to conclude that t s' is reducible.
- There are no other cases, because t is not an abstraction.

We can therefore conclude via **(CR3)** for type T that t s is reducible.

Reducibility for λ

Lastly, we need to show that reducibility commutes with substitution.

Lemma 1: If s $\in$ R[S] and v[s/x] $\in$ R[T], then t = λ x:S.v $\in$ R[S → T]

We want to show that t s for s $\in$ R[S] is in R[T]. First, note that t is strongly normalizing: if v had an infinite series of reductions then so would v[s/x]. We can therefore proceed by induction on the maximum number of reductions required for t *and* s.

Reducibility for λ (cont.)

We again consider all the possible ways for t s to take a step:

- If t s ⇒ v[s/*x*] then we are done by assumption.
- If (λx.v)s ⇒ (λx.v')s then we can invoke the IH as v' will reduce in strictly fewer steps than v.
- If (λx.v)s ⇒ (λx.v)s' then we can similarly invoke the IH due to s'.

We conclude via CR3 of type T that t s ∈ *R*[T] and hence t ∈ *R*[S → T].

… One Last Lemma

Lemma 2: Let t be any term. Suppose its free variables are some set $x_0 \ldots x_n$ of types $S_0 \ldots S_n$. If $s_0 \ldots s_n$ are reducible, then t$[s_0/x_0 \ldots s_n/x_n]$ is reducible.

We proceed by induction on the term t. We will notate t$[s_0/x_0 \ldots s_n/x_n]$ as t[**S/X**].

- If $t = x_i$ then t[**S/X**] = s_i which is reducible by assumption.
- If t = w v then invoking the IH on w and v gives us that w[**S/X**] and v[**S/X**] are reducible. By the definition of reducibility of functions (w[**S/X**] v[**S/X**]) is reducible as well, but (w[**S/X**] v[**S/X**]) = (w v)[**S/X**] = t[**S/X**].
- If t = λy:Y.w then invoking the IH on w gives us that for all v ∈ *R*[Y], [**S/X**, v/y]w is reducible. By Lemma 1 that means λy.(w[**S/X**]) is reducible. But λy.(w[**S/X**]) = (λy.w)[**S/X**] = t[**S/X**].

Strong Normalization for STLC

> **Lemma 2**: Let t be any term. Suppose its free variables are some set $x_0 \ldots x_n$ of types $S_0 \ldots S_n$. If $s_0 \ldots s_n$ are reducible, then t$[s_0/x_0 \ldots s_n/x_n]$ is reducible.

Applying Lemma 2, setting $s_0 \ldots s_n$ to the variables $x_0 \ldots x_n$ themselves hands us back the terms we started with, giving us the statement that **all STLC terms are reducible**.

By (CR1) we have that **all STLC terms are SN**.

System F

System F adds parametric polymorphism to the STLC:

$T ::= \ldots \mid \forall X.\ T$

$t ::= \ldots \mid t[T] \mid \Lambda X.t$

Operational Semantics (new rules)

> t ⇒ t'
>
> -----------
>
> t[T] ⇒ t'[T]

> t ⇒ t'
>
> -----------
>
> ΛX.t ⇒ ΛX.t'

> (ΛX.t)[T] ⇒ t[T/X]

New Typing Rules

> Γ, X ⊢ t' : T
>
> ---------------
>
> Γ ⊢ Λx.t' : ∀X.T

> Γ ⊢ t : ∀X.T
>
> -------------
>
> Γ ⊢ t[T'] : T[T'/X]

A Note on Curry-Howard

- System F corresponds to a logical system of equal power to "second-order peano arithmetic"
- Strong Normalization for System F amounts to proving *consistency* for this system.
- Hence, by Godel's Second incompleteness theorem, we cannot prove strong normalization for System F using only statements that we could state in 2nd-order PA.

2nd-order PA lets us quantify over both numbers and sets of numbers. We will end up quantifying over sets of sets of terms, and our doing so will be essential.

Intuition

The main case to handle are type applications. We want to say that t=(ΛX.t) is reducible if t[T'] is reducible for all t'. We run into two issues this way:

1. What does "reducible" mean here? The type T that we are substituting can be anything, including itself, making this definition potentially circular.
2. Consider a system where recursive types are added to the universe of types. Then any proof that concluded all t[T']s were reducible without "inspecting" T' would prove too much.

Just as our definition of *R* relied on the fact that STLC lambdas took halting inputs to halting outputs, we'll define a new notion of "parametric reducibility candidate" which maps reducibility candidates to reducibility candidates. As a consequence, we will show that polymorphic types take "types where all terms halt to types where all terms halt."

Roadmap

- Instead of having a single set R[T] of reducible terms for a type T, we'll axiomatize the sets that are *reducibility candidates* and show how they can be used to recursively build up a notion of reducibility for types with ∀s in them.
- We will then show that our new definition for redicibility of t[T] and ΛX.t satisfy **(CR1-3)**.
- We will use this to conclude that all terms in System F are reducible and hence strongly normalizable

Neutrality and CR3

These are as before, except **CR3** applies to any *neutral* terms.

A term is neutral if it is neither a λ nor a Λ abstraction, i.e. it looks like:

- x (a variable)
- t t'
- t[T]

And CR3 reads:

> **(CR3)** If t is neutral and t' is reducible for all t ⇒ t', then t is reducible

Reducibility Candidates

We will speak abstractly of reducibility *candidates* for a type T. A reducibility candidate for type T is any set of terms satisfying **CR1-3**. We can compose reducibility candidates together as before: given two candidates C_1, C_2 for types T_1, T_2 we can construct a new candidate $C_1 \to C_2$ for type $T_1 \to T_2$ using the expected definition:

> t $\in C_1 \to C_2$ iff $\forall$u(u $\in C_1 \Rightarrow$ t u $\in C_2$)

Which we showed in the STLC proof still satisfies **CR1-3**.

Reducibility for Polymorphic Types

We notate a type T with free variables X_i as T[X_i...], we can define substitution of types S_i in the usual way, notated T[S_i/X_i...].

> Note: there are exactly as many Ss as there are Xs, and so one.

Now let R_i be reducibility candidates for each S_i. We can define reducibility candidates for T "pointwise" by specifying how each candidate behaves under a suitable substitution of reducibility candidates. We'll notate the set RED⟨T⟩[R_i/X_i...] to be the "parametric reducibility candidate" for type T.

We can define it recursively on the structure of types.

Definition of RED⟨T⟩$[R_i/X_i \ldots]$

- If T = X_i, then RED⟨T⟩$[R_i/X_i \ldots] = R_i$
- If T = U → W, then RED⟨T⟩$[R_i/X_i \ldots]$ = RED⟨U⟩$[R_i/X_i \ldots]$ → RED⟨W⟩$[R_i/X_i \ldots]$

For ∀ types, we do the same move as we did with functions, but at the level of types and reducibility candidates:

- If T = $\forall Z$.W, then RED⟨T⟩$[R_i/X_i \ldots]$ is the set of terms t in the substituted type T$[U_i/X_i \ldots]$ where for every type V and reducibility candidate R_V for V, t[V] $\in$ RED⟨W⟩$[R_i/X_i \ldots, R_V/Z]$

We must now show that these candidates satisfy **CR1-3**. The first case is tautological, the second case follows by the same argument as for the STLC. We will focus on the third case where T is some ∀Z.W.

RED⟨∀Z.W⟩[R_i/X_i...] is a reducibility candidate: CR1

The argument is very similar to what we've seen before:

> **(CR1)** If t ∈ RED⟨∀Z.W⟩[R_i/X_i...], then t is strongly normalizing.

For an arbitrary type V and candidate R_V we have that t[V] ∈ RED⟨W⟩[R_i/X_i...,R_V/Z]. By the IH, that means t[V] is strongly normalizing. But that also means there is no infinite chaing t⇒t'⇒... because then we would also have t[V]⇒t'[V]⇒.... We conclude that t is strongly normalizing.

RED⟨∀Z.W⟩[R_i/X_i...] is a reducibility candidate: CR2

> **(CR2)** If t ∈ RED⟨∀Z.W⟩[R_i/X_i...] and t ⇒ t', then t' ∈ RED⟨∀Z.W⟩[R_i/X_i...]

For an arbitrary type V and candidate R_v we have that t[V] ∈ RED⟨W⟩[R_i/X_i...,R_v/Z]. By the IH, t'[V] is also reducible.

But then we've shown that for all V, t'[V] is reducible under the appropriate assumptions, which is exactly the definition of reducibility for t'.

RED⟨∀Z.W⟩$[R_i/X_i ...]$ is a reducibility candidate: CR3

> **(CR3)** If t is neutral and t' ∈ RED⟨∀Z.W⟩$[R_i/X_i ...]$ for all t ⇒ t', then t ∈ RED⟨∀Z.W⟩$[R_i/X_i ...]$

Once again we consider and arbirarty type V and candidate R_v. For any t' where t ⇒ t', we also have t[V] ⇒ t'[V].

> These are the *only* such transitions for t[V] because t is neutral. In particular it is not a type abstraction.

We know that t'[V] ∈ RED⟨W⟩$[R_i/X_i ..., R_v/Z]$ because t' is reducible. We therefore have via the IH that t[V] ∈ ∈ RED⟨W⟩$[R_i/X_i ..., R_v/Z]$, and can therefore conclude that t ∈ RED⟨∀Z.W⟩$[R_i/X_i ...]$.

Two lemmas

Next we will show that redicibility is preserved across two different instances of type substitution

> **Lemma 3**: If for every type V and candidate R_v we have w[V/R_v] ∈ RED⟨W⟩[R_i/X_i...,V/R_v], then ΛZ.w ∈ RED⟨∀Z.W⟩[R_i/X_i...]

Note that w is SN via a similar argument to why **CR1** holds for terms with types like ∀Z.W. We can then argue by induction on the maximum number of reduction steps required for w to reach a normal form.

We want to show that (ΛY.w)[V] ∈ RED⟨W⟩[R_i/X_i...,V/R_v] for any V, R_v. There are two cases for how this term looks after a single reduction step:

- (ΛY.w')[V], which is reducible by the IH.
- w[V/Y] which is reducible by assumption.

Because **CR3** holds of the set RED⟨W⟩[R_i/X_i...,V/R_v] and all possible terms that (ΛY.w)[V] steps to land in that set, we conclude that (ΛY.w)[V] ∈ RED⟨W⟩[R_i/X_i...,V/R_v] as well.

> **Lemma 4**: RED⟨T[V/Y]⟩$[R_i/X_i\ldots]$ = RED⟨T⟩$[R_i/X_i\ldots$,RED⟨V⟩$[R_i/X_i\ldots]$/Y]

Induction on T

- If T = X_i, then Y does not appear and RED⟨T[V/Y]⟩$[R_i/X_i\ldots]$ = X_i = RED⟨T⟩$[R_i/X_i\ldots$,RED⟨V⟩$[R_i/X_i\ldots]$/Y]
- If T = Y, then T[V/Y] = V,
  - RED⟨T[V/Y]⟩$[R_i/X_i\ldots]$
  - = RED⟨V⟩$[R_i/X_i\ldots]$
  - = RED⟨Y⟩[RED⟨V⟩$[R_i/X_i\ldots]$/Y]
  - = RED⟨Y⟩$[R_i/X_i\ldots$,RED⟨V⟩$[R_i/X_i\ldots]$/Y].
- If T = S → T, then we immediately get the equality by invoking the IH on RED⟨S⟩ and RED⟨T⟩

> **Lemma 4**: RED⟨T[V/Y]⟩[R_i/X_i…] = RED⟨T⟩[R_i/X_i…,RED⟨V⟩[R_i/X_i…]/Y]

If T = ∀Z.W, then RED⟨T[V/Y]⟩[R_i/X_i…] = RED⟨∀Z.W[V/Y]⟩[R_i/X_i…] = the terms w ∈ W[U_i/X_i,V/Y] for which w[S] ∈ RED⟨W[V/Y]⟩[R_i/X_i…,S/R_s] for all type S and reducibility candidates R_s for S.

By the IH, RED⟨W[V/Y]⟩[R_i/X_i…,S/R_s] = RED⟨W⟩[R_i/X_i…,S/R_s, RED⟨V⟩[R_i/X_i…,S/R_s]/Y]

But this is the same result that we get if we unwrap the definition of RED⟨∀Z.W⟩[R_i/X_i…,RED⟨V⟩[R_i/X_i…]/Y]. So we are done.

Corrolary to Lemma 4

> **Lemma 4**: RED⟨T[V/Y]⟩$[R_i/X_i\ldots]$ = RED⟨T⟩$[R_i/X_i\ldots,$RED⟨V⟩$[R_i/X_i\ldots]$/Y]

This does all the heavy lifting needed for us to prove:

> **Lemma 5**: If t ∈ RED⟨∀Y.W⟩$[R_i/X_i\ldots]$, then t[V] ∈ RED⟨W[V/Y]⟩$[R_i/X_i\ldots]$ for all V.

By definition of RED⟨∀Y.W⟩, we have that t[V] ∈ RED⟨W⟩$[R_i/X_i\ldots,R_v/$Y] for any candidate R_v. By setting R_v = RED⟨V⟩$[R_i/X_i\ldots]$, we have t[V] ∈ RED⟨W⟩$[R_i/X_i\ldots,$RED⟨V⟩$[R_i/X_i\ldots]$/Y], and by **Lemma 4** we therefore have t[V] ∈ RED⟨W[V/Y]⟩$[R_i/X_i\ldots]$.

Putting it all together

Just like with the STLC, each of these steps:

- Showing RED has the properties **CR1-3**
- Showing reducibility commutes with Λ-abstraction
- Showing reducibility commutes with type application

Lets us handle different cases of a big induction that will allow us to conclude SN for System F. As before we will state a slightly stronger theorem in order to make the induction go through.

Main Theorem

Let t be any term of type T with

- free variables x_i... with types S_i...
- free type variables X_j....

Let V_j ... be types with reducibility candidates R_j ..., let s_i... be terms of types $S_i[V_j / X_j ...]$... each in RED$\langle S_i \rangle [R_j / X_j ...]$....

Then, t$[V_j / X_j ..., s_i/x_i ...] \in$ RED$\langle$ T $\rangle [R_j / X_j ...]$

We will prove this by induction on t

Proof of Main Theorem (variables)

The first few cases are much the same as **Lemma 2**

> t is a variable x

We need only show that t$[V_j/X_j \ldots, s/x] \in$ RED⟨T⟩$[R_j/X_j \ldots]$, but by assumption s is a member of a suitable reducibility candidate of type S = T, so we are done.

Proof of Main Theorem (applications)

> t is an application `w v`

So w : S → T and v : S. By the IH, for any suitable $V_j \ldots, R_j \ldots$ we have

- w$[V_j / X_j \ldots, s_i / x_i \ldots] \in$ RED⟨S→T⟩$[R_j / X_j \ldots]$
- v$[V_j / X_j \ldots, s_i / x_i \ldots] \in$ RED⟨S⟩$[R_j / X_j \ldots]$
  By the definition of RED for function types, we therefore have that:

(w$[V_j / X_j \ldots, s_i / x_i \ldots]$ v$[V_j / X_j \ldots, s_i / x_i \ldots]$) $\in$ RED⟨T⟩$[R_j / X_j \ldots]$

= (w v)$[V_j / X_j \ldots, s_i / x_i \ldots]$

= t$[V_j / X_j \ldots, s_i / x_i \ldots]$

Proof of Main Theorem (λs)

> t is an abstraction λy:Y.w

Recall **Lemma 1**

> **Lemma 1**: If s ∈ *R*[S] and [s/*x*]v ∈ *R*[T], then t = λ *x*:S.v ∈ *R*[S → T]

The proof here relied only on the definition of *R* for functions (which is the same as that for RED), and the properties **CR1-3** which still hold for component types, even with type parameters.

We can then proceed as before: applying the IH to w, and then using this lemma to conclude that t is reducible as well.

Proof of Main Theorem (Λs)

> t is a type abstraction ΛY.w, and T = ∀Y.Z

w's free type variables are Y,X_j.... Invoking the IH on w gives us that

w[V/Y,V_j/X_j...] ∈ RED⟨Z⟩[R/Y,R_j/X_j...],

For any type V and reducibility candidate R for V. Applying Lemma 3:

> **Lemma 3**: If for every type V and candidate R_v we have w[V/R_v] ∈ RED⟨W⟩[R_i/X_i...,V/R_v], then ΛZ.w ∈ RED⟨∀Z.W⟩[R_i/X_i...]

Immediately gives us that (t = ΛY.w)[V_j/X_j...,s_i/x_i...] ∈ RED⟨T = ∀Y.Z⟩[R_j/X_j...].

Proof of Main Theorem (type applications)

> t is a type application t'[A], with t' : ∀Y.W

Invoke the IH on t', giving us that t'$[V_j/X_j \ldots, s_i/x_i \ldots]$ ∈ RED⟨∀Y.Z⟩$[R_j/X_j \ldots]$. Recall Lemma 5:

> **Lemma 5**: If t ∈ RED⟨∀Y.W⟩$[R_i/X_i \ldots]$, then t[V] ∈ RED⟨W[V/Y]⟩$[R_i/X_i \ldots]$ for all V.

Which immediately gives us the desired result. This completes the proof.

In Conclusion

As before, we can substitute free variables in for themselves (i.e. $s_i = x_i$) to yield that every term in System F is a member of a suitable reducibility candidate. By **CR1** this gives us that every term in System F is strongly normalizing.

> Girard goes further and defines "reducibility" not as membership in any candidate, but membership in RED⟨T⟩$[SN_i/X_i\ldots]$, where SN_i are the strongly normalizing terms of the corresponding type type V_i. It is not clear to me if this is a necessary step for strong normalization, or if it simply "ties the knot". It is certainly the case that strongly normalizing terms form a reducibility candidate.