Strong Normalisation for System F

Thomas Waring

September 18, 2020

Contents

1 Introduction

The reference for these notes is [GLT93], chapter 11 (for definitions) and chapter 14 (for normalisation). First we briefly lay out the structure of System F.

We skip for now proper definitions of free variables and of substitution. In short, free variables are those not "bound" by λ- or universal abstraction. The substitution $a[b/c]$ denotes replacing all free occurences of the variable c in a with b. Also, note that terms are considered up to α-equivalence: changing the names of bound variables. For more detail see [Sel08] chapter 8, noting slight differences of notation.

Types are defined inductively, starting from an infinite set $X, Y, Z, \dots$ of *type variables*, and with three rules.

1. Type variables are types.
2. If U and V are types, then $U \to V$ is a type.
3. If V is a type, and X is a type variable, then $\Pi X.V$ is a type.

From this, there are five ways to form terms.

1. Variables: $x^T, y^T, z^T, \dots$ of type T.
2. Application: if t and u are terms of type $U \to V$ and U, then tu is a term of type V.

3. λ-abstraction: if x^U is a variable of type U, and v is a term of type V then $\lambda x^U.v$ is a term of type $U \to V$.

4. Universal application (or extraction): if t is a term of type $\Pi X.V$ and U is type, then tU is a term of type $V[U/X]$.

5. Universal abstraction: if v is a term of type V, then $\Lambda X.v$ is a term of type $\Pi X.V$, so long as X is not free in the type of any free variable of v.

There are two "reduction" operations on terms. The first familiar is familiar (as β-reduction) from simply typed lambda calculus:

$$(\lambda x^U.v)u \rightsquigarrow v[u/x].$$

The second is its equivalent for universal abstraction / application:

$$(\Lambda X.v)U \rightsquigarrow v[U/X].$$

The restriction on universal abstraction arises to avoid terms of the form $\Lambda X.x^X$, where the free variable x has no type. On the other hand, the term $\Lambda X.\lambda x^X.x^X$ has the type $\Pi X.X \to X$. Note that $fu_1u_2 := (fu_1)u_2$. Free variables must have a defined type, to ensure that λ-abstraction and β-reduction are well-defined.

A term is called *neutral* if it is of the form x, tu or tU. That is, if it does not start with an abstraction of either type.

For a given term u, define $\nu(u)$ to be the longest sequence of reductions starting with u. For example:

$$(\Lambda X.\lambda x^X.x)Vv^V \rightsquigarrow (\lambda x^V.x)v^V \rightsquigarrow v^V$$

so $\nu((\Lambda X.\lambda x^X.x)Vv^V) = 2$ (if v is a variable), as in each case there was a single possible reduction (a single *redex*). The primary goal of these notes is to show that $\nu(u)$ is finite for every term u.

2 Church-Rosser Theorem

See [Gal90], §10.

3 Strong Normalisation

3.1 Preamble

The following idea comes from [Tai67], applied to Gödel's System T: which amounts to simply typed λ-calculus with additional "constant types", for the integers and booleans[1]. The focus there, however, is on precisely *what* system of arithmetic (logic) is required to represent this system. In particular, the logic must be able to prove (define?) equality of terms under the equivalent of β-reduction.

[1]Note that §1 of Tait's paper is (to me) practically incomprehensible, but (bravely pushing on) §2 is better: the language is similar to that of λ-calculus. The idea of "convertability" appears at the bottom of the first page.

3.2 Reducibility candidates

The key idea of this proof is that of *reducibility candidates*. In the simply typed case, we may define a set RED_T of *reducible terms of type* T, inductively on the type T. In particular, when $T = U \to V$, we have:

$$t \in \mathrm{RED}_T \iff \forall u(u \in \mathrm{RED}_U \implies tu \in \mathrm{RED}_V)$$

However, applying this to universal abstraction, the definition is circular. Let's examine the universal identity $i = \Lambda X.\lambda x^X.x$ to see this. Naively, we might say i is reducible if for any type U, $iU = \lambda x^U.x$, of type $U \to U$ is reducible. Then, iU is reducible if for any term of type u, iUu is reducible. But then we may set $U = \Pi X.X \to X$, and $u = i$, and

$$iUu = (\Lambda X.\lambda x^X.x)(\Pi X.X \to X)(\Lambda X.\lambda x^X.x) \rightsquigarrow \Lambda X.\lambda x^X.x$$

So i is reducible, so long as i is reducible — no good.

The solution is to build our notion of reducibility up with the type. Then, a term t of type $\Pi X.V$ is reducible if and only if for every type U, and *for every reducibility candidate* $\mathcal{R}$, the term tU is reducible of type $T[U/X]$. We avoid circularity by defining reducibility of the new term with respect to $\mathcal{R}$, decoupling it from the original term t. We then need to show that the property "is strongly normalising" is a reducibility candidate, which leads to the final result.

Definition 3.1. A *reducibility candidate* of type U is a set $\mathcal{R}$ of terms of type U, such that:

- (CR1) If $t \in \mathcal{R}$, then t is strongly normalising.
- (CR2) $t \in \mathcal{R}$ and $t \rightsquigarrow t'$, then $t' \in \mathcal{R}$.
- (CR3) t is neutral, and whenever we convert a redex in t we obtain a term $t' \in \mathcal{R}$, then $t \in \mathcal{R}$ also.

By (CR3), any term which is neutal and normal belongs to every reducibility candidate of the appropriate type.

Lemma 3.2. *The set $\mathcal{SN}_U$ of strongly normalising terms of type U is a reducibility candidate.*

Proof. (CR1) is tautological. If $t \rightsquigarrow t'$, then $\nu(t') < \nu(t)$, so t' is also strongly normalising. If there were an infinite path of reductions starting from t, then the t' in the second step would also not be strong normalising, so $t' \notin \mathcal{SN}_U$. □

Lemma 3.3. *Given reducibility candidates $\mathcal{R}$ and $\mathcal{S}$ of types U and V, the set $\mathcal{R} \to \mathcal{S}$ of terms of type $U \to V$ is defined by:*

$$t \in \mathcal{R} \to \mathcal{S} \iff \forall u(u \in \mathcal{R} \implies tu \in \mathcal{S})$$

is a reducibility candidate.

Proof. (CR1) Given $t \in \mathcal{R} \to \mathcal{S}$ and any u of type U, $\nu(t) \leqslant \nu(tu)$, so t is strongly normalising.

(CR2) Let some $t \in \mathcal{R} \to \mathcal{S}$ be given, and t' such that $t \rightsquigarrow t'$. For any u, $tu \rightsquigarrow t'u$, so $t'u \in \mathcal{S}$ (by 2). This implies $t' \in \mathcal{R} \to \mathcal{S}$.

(CR3) Let some $u \in \mathcal{R}$ and neutral t as in (CR3) be given. As t does not begin with an abstraction, the only possible one-step reductions beginning with tu are $tu \rightsquigarrow t'u$ and $tu \rightsquigarrow tu'$, where $t \rightsquigarrow t'$ and $u \rightsquigarrow u'$ are one-step reductions. By assumption, $t' \in \mathcal{R} \to \mathcal{S}$, which means that $t'u \in \mathcal{S}$. For the other case, we induct on $\nu(u)$, which is finite. $u' \in \mathcal{R}$ by (2), and $\nu(u') < \nu(u)$, which implies, by induction, that $tu' \in \mathcal{S}$. Therefore, by (CR3) applied to $\mathcal{S}$, $tu \in \mathcal{S}$, and so $t \in \mathcal{R} \to \mathcal{S}$. □

3.3 Reducibility with parameters

Lemma 3.4. *Let $T[\underline{X}]$ be a type, and suppose the sequence $\underline{X} = X_1, X_2, \ldots$ is assumed to contain all free (type) variables of T. With a sequence $\underline{U}$ of types, we may define a type $T[\underline{U}/\underline{X}]$ by simultaneous substitution. Let $\mathcal{R}$ be a sequence of reducibility candidates, with $\mathcal{R}_i$ of type U_i. Then we can define a set $RED_T[\underline{\mathcal{R}}/\underline{X}]$ of terms of type $T[\underline{U}/\underline{X}]$ inductively by the following.*

- *If $T = X_i$ then $RED_T[\underline{\mathcal{R}}/\underline{X}] = \mathcal{R}_i$.*
- *If $T = V \to W$, then $RED_T[\underline{\mathcal{R}}/\underline{X}] = RED_V[\underline{\mathcal{R}}/\underline{X}] \to RED_W[\underline{\mathcal{R}}/\underline{X}]$.*
- *If $T = \Pi Y.W$, then $RED_T[\underline{\mathcal{R}}/\underline{X}]$ is the set of terms t of type $T[\underline{U}/\underline{X}]$ such that, for every type V and reducibility candidate $\mathcal{S}$ of this type, $tV \in RED_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$.*

Proof. As we will see later, this definition is remarkably circular as $\mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$ is itself a reducibility candidate. As such, we make the definition extra precise (as a lemma). We conceive of this definition as a function, assigning to a type T, and substitution as defined, a set of terms of type $T[\underline{U}/\underline{X}]$. Note that, entirely separate from this definition, we have for each type U a family $\mathcal{C}_U$ of reducibility candidates of this type: this is defined by comprehension on definition 3.1. The complexity $c(T)$ of a type T is defined in the obvious way, counting the number of Λ or $\to$ symbols.

We seek to define a function for each type T, assigning a valid substitution (one including all free variables) to the set $\mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$. In excruiciating detail, let $\mathcal{X}$ be the set of all type variables, and Sub the set of partial functions:

$$\mathcal{X} \to \coprod_{U \in \mathcal{U}} \mathcal{C}_U$$

with finite domain. Then for a given type T the domain $\Delta(T)$ of our function is the subset:

$$\Delta(T) = \{\eta \in \mathrm{Sub} \mid \mathrm{dom}(\eta) \supset \mathrm{FV}_{\mathrm{type}}(T)\}$$

Let Σ be the set of System F terms. To induct, we need to prove that for any $n \in \mathbb{N}$, if we are given the set:

$$\{\mathrm{RED}_T : \Delta(T) \to \mathcal{P}\Sigma \mid c(T) < n\} \tag{1}$$

then there is a unique choice of set:

$$\{\mathrm{RED}_T : \Delta(T) \to \mathcal{P}\Sigma \mid c(T) < n+1\}$$

corresponding to the above definition. From this perspective, the fact that $\mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$ is a set of terms of a particular type has been glossed over, so this must be part of the induction. By the disjoint union, each η determines an assignment $\mathcal{X} \to \mathcal{U}$, for each free varibale. Abusing our notation slightly we denote $T[\eta] = T[\underline{U}/\underline{X}]$, where $\eta(X_i)$ is a reducibility candidate of type U_i.

For $n = 0$, T must be a variable X, so we assign $\mathrm{RED}_T[\eta] = \eta(X)$. If $\eta(X)$ is a reducibility candidate of type U, then $\mathrm{RED}_T[\eta]$ is a set of terms of type $U = T[\eta]$.

For $n > 0$, T is either an arrow or universal abstraction. If $T = V \to W$ then for any given η we may construct the set as usual, noting that the free type variable of V and W are each at most those of T. Also, by the inductive hypothesis, the members of $\mathrm{RED}_T[\eta]$ will terms of type:

$$V[\eta] \to W[\eta] = T[\eta]$$

Finally, suppose $T = \Pi Y.W$. For any $\eta : \Delta(W) \to \coprod_{U \in \mathcal{U}} \mathcal{C}_U$ and reducibility candidate $\mathcal{S}$ of type V, define:

$$(\eta + \mathcal{S}/Y)(X) = \begin{cases} \mathcal{S} & X = Y \\ \eta(X) & \text{else} \end{cases}$$

Then we define $\mathrm{RED}_T[\eta]$ to be the set of terms of type $\Pi Y.W[\eta]$, such that for any type V and reducibility candidate $\mathcal{S}$ of that type, $tV \in \mathrm{RED}_W[\eta + \mathcal{S}/Y]$.

This constructs $\mathrm{RED}_T[\eta]$ for any T of complexity n, and $\eta \in \Delta(T)$, so by induction our construction uniquely determines the sets as claimed. □

Remark 3.5. Observe that the notation $\mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$ does not explicitly include the substitutions U_i/X_i, which are nonetheless necessary to choose the right $\mathcal{R}_i$ (see [Gal90] p.38).

Example 3.6. If $T = \Pi X.X \to X$, then (with $\underline{X}$ empty), $\mathrm{RED}_T[-]$ is the set of terms t with type T, such that for every type V and reducibility candidate $\mathcal{S}$:

$$tV \in \mathrm{RED}_{X \to X}[\mathcal{S}/X] = \mathcal{S} \to \mathcal{S}.$$

Lemma 3.7. *$RED_T[\underline{\mathcal{R}}/\underline{X}]$ is a reducibility candidate of type $T[\underline{U}/\underline{X}]$.*

Proof. By induction on T. The only case we need verify is $T = \Pi Y.W$. CR1 and CR2 are practically indentical to Lemma 3.3, and CR3 is easier.

(CR1) Let some $t \in \mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$ be given. With an arbitrary type V, and arbitrary reducibility candidate $\mathcal{S}$, tV is strongly normalising, by inductively applying (CR1) to $\mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$. As $\nu(t) \leqslant \nu(tV)$, t is also strongly normalising.

(CR2) If $t \rightsquigarrow t'$, then for any type V, $tV \rightsquigarrow t'V$. Given a reducibility candidate $\mathcal{S}$ of this type, by induction:

$$t'V \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$$

so $t' \in \mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$.

(CR3) Suppose that t is neutral, and every term t' one step from t belongs to $\mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$. Then for any type V, the only one-step reductions of tV are of the form $t'V$, as t is neutral. Since $t' \in \mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$, $t'V \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$ for every candidate $\mathcal{S}$. By (CR3), this means $t \in \mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$. □

Lemma 3.8. *$RED_{T[V/Y]}[\underline{\mathcal{R}}/\underline{X}] = RED_T[\underline{\mathcal{R}}/\underline{X}, RED_V[\underline{\mathcal{R}}/\underline{X}]/Y]$*

Proof. Again, induction on T. First, if T is a variable, then $T = X_i$ or $T = Y$. In the first case, $T[V/Y] = T$, and both sides are $\mathcal{R}_i$ by definition. In the latter case, both sides are $\mathrm{RED}_V[\underline{\mathcal{R}}/\underline{X}]$.

If $T = W_1 \to W_2$, then the left-hand side is:

$$\mathrm{RED}_{W_1[V/Y] \to W_2[V/Y]}[\underline{\mathcal{R}}/\underline{X}] = \mathrm{RED}_{W_1[V/Y]}[\underline{\mathcal{R}}/\underline{X}] \to \mathrm{RED}_{W_2[V/Y]}[\underline{\mathcal{R}}/\underline{X}]$$

The right-hand side is

$$\mathrm{RED}_{W_1}[\underline{\mathcal{R}}/\underline{X}, \mathrm{RED}_V[\underline{\mathcal{R}}/\underline{X}]/Y] \to \mathrm{RED}_{W_2}[\underline{\mathcal{R}}/\underline{X}, \mathrm{RED}_V[\underline{\mathcal{R}}/\underline{X}]/Y]$$

By induction $\mathrm{RED}_{W_i[V/Y]}[\underline{\mathcal{R}}/\underline{X}] = \mathrm{RED}_{W_i}[\underline{\mathcal{R}}/\underline{X}, \mathrm{RED}_V[\underline{\mathcal{R}}/\underline{X}]/Y]$, for $i = 1, 2$, so the two expressions agree.

Finally, let $T = \Pi Z.W$. Then $T[V/Y] = \Pi Z.(W[V/Y])$, so the equality is clear by applying the inductive hypothesis to W. □

3.4 Reducibility

We eventually want to induct on the construction of a *term*, to ensure its reducibility (with reference to some fixed sequence of candidates). The variable and application cases are simple, but the others require the following lemmas.

Lemma 3.9 (λ-abstraction)**.** *If for every $v \in RED_V[\underline{\mathcal{R}}/\underline{X}]$ the term $w[v/y] \in RED_W[\underline{\mathcal{R}}/\underline{X}]$, then $\lambda y^V.w \in RED_{V\to W}[\underline{\mathcal{R}}/\underline{X}]$.*

Proof. We need to show that $(\lambda y^V.w)v \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}]$ for every $v \in \mathrm{RED}_V[\underline{\mathcal{R}}/\underline{X}]$. Let such a v be given. Noting that by assumption (with $v = y$), w is strongly normalising, we induct on $\nu(v) + \nu(w)$. Considering one-step reductions from $(\lambda y^V.w)v$, there are three cases. In each, they belong to $\mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}]$.

- $(\lambda y^V.w)v'$ with $v \rightsquigarrow v'$ in one step. Then $\nu(v') < \nu(v)$.
- $(\lambda y^V.w')v$ with $w \rightsquigarrow w'$ in one step. Then $\nu(w') < \nu(w)$.
- $w[v/y] \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}]$ by assumption.

As we are dealing with an application, (CR3) implies that $(\lambda y^V.w)v \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}]$, which implies the result. □

Lemma 3.10 (Universal application)**.** *If $t \in RED_{\Pi Y.W}[\underline{\mathcal{R}}/\underline{X}]$, then $tV \in RED_{W[V/Y]}[\underline{\mathcal{R}}/\underline{X}]$.*

Proof. By assumption, for any reducibility candidate $\mathcal{S}$ of type V, $tV \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$. Taking $\mathcal{S} = \mathrm{RED}_V[\underline{\mathcal{R}}/\underline{X}]$ and using Lemma 3.8 the result is immediate. □

Lemma 3.11 (Universal abstraction)**.** *If for every type V and candidate $\mathcal{S}$ of that type, $w[V/Y] \in RED_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$, then $\Lambda Y.w \in RED_{\Pi Y.W}[\underline{\mathcal{R}}/\underline{X}]$.*

Proof. Given a type V and candidate $\mathcal{S}$, we must show that $(\Lambda Y.w)V \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$. This is entirely analogous to the λ-abstraction case, now we induct on $\nu(w)$. Converting a redex in $(\Lambda Y.w)V$ gives two cases:

- $(\Lambda Y.w)V \rightsquigarrow (\Lambda Y.w')V$, where $\nu(w') < \nu(w)$.
- $(\Lambda Y.w)V \rightsquigarrow w[V/Y] \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$ by assumption.

Applying (CR3) and the definition of $\mathrm{RED}_{\Pi Y.W}[\underline{\mathcal{R}}/\underline{X}]$ the result follows. □

Now we are ready to prove the final result. As for the simply-typed case, we induct on a stronger predicate to allow for a well-formed argument.

Theorem 3.12. *Let t be any term of type T, with free variables among $x_1, \ldots, x_n$, of types $U_1, \ldots, U_n$. Suppose also that the free type variables of $T, U_1, \ldots, U_n$ are among $X_1, \ldots, X_m$. Let $\mathcal{R}_1, \ldots, \mathcal{R}_m$ be reducibility candidates of types $V_1, \ldots, V_n$, and $u_1, \ldots, u_n$ terms of types $U_1[\underline{V}/\underline{X}], \ldots, U_n[\underline{V}/\underline{X}]$, each in $RED_{U_1}[\underline{\mathcal{R}}/\underline{X}], \ldots, RED_{U_n}[\underline{\mathcal{R}}/\underline{X}]$. Then:*

$$t[\underline{V}/\underline{X}][\underline{u}/\underline{x}] \in RED_T[\underline{\mathcal{R}}/\underline{X}]$$

Proof. We induct on the construction of t. If t is a variable, say x_i, then $T[\underline{V}/\underline{X}] = U_i[\underline{V}/\underline{X}]$, and $t[\underline{V}/\underline{X}][\underline{u}/\underline{x}] = u_i \in \mathrm{RED}_{U_i}[\underline{\mathcal{R}}/\underline{X}] = \mathrm{RED}_T[\underline{\mathcal{R}}/\underline{X}]$.

If $t = vw$, then both $v[\underline{V}/\underline{X}][\underline{u}/\underline{x}]$ and $w[\underline{V}/\underline{X}][\underline{u}/\underline{x}]$ belong to the appropriate set by induction. By definition, this implies that:

$$v[\underline{V}/\underline{X}][\underline{u}/\underline{x}](w[\underline{V}/\underline{X}][\underline{u}/\underline{x}]) = t[\underline{V}/\underline{X}][\underline{u}/\underline{x}]$$

belongs to $\mathrm{RED}_T[\mathcal{R}/\underline{X}]$.

Let $t = \lambda y^V.w$ of type $V \to W$. By the inductive hypothesis

$$w[\underline{V}/\underline{X}][\underline{u}/\underline{x}, v/y] \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}]$$

for every v of type $V[\underline{V}/\underline{X}]$. Then by Lemma 3.9 we have that:

$$\lambda y^{V[\underline{V}/\underline{X}]}.w[\underline{V}/\underline{X}][\underline{u}/\underline{x}] = t[\underline{V}/\underline{X}][\underline{u}/\underline{x}]$$

belongs to our reducible set.

If $t = t'V$, with t' of type $\Pi Y.T'$, making $T = T'[V/Y]$. By the inductive hypothesis,

$$t'[\underline{V}/\underline{X}][\underline{u}/\underline{x}] \in \mathrm{RED}_{\Pi Y.T'}[\underline{\mathcal{R}}/\underline{X}]$$

Applying Lemma 3.10 implies the result.

The final case is $t = \Lambda Y.w$. Again, using the inductive hypothesis, for any type V and reducibility candidate $\mathcal{S}$ of this type:

$$w[\underline{V}/\underline{X}, V/Y][\underline{u}/\underline{x}] \in \mathrm{RED}_W[\underline{\mathcal{R}}/\underline{X}, \mathcal{S}/Y]$$

We apply Lemma 3.11 which implies the result. □

Corollary 3.13. *Every term of System F is strongly normalising.*

Proof. Apply the above, with $V_i = X_i$ and $u_j = x_j$, making each substitution the identity. Any sequence $\mathcal{R}_i$ of reducibility candidates works, for example the sets $\mathcal{SN}_i$ of strongly normalising terms of type X_i. Then (CR1) implies that every term is strongly normalising. □

4 Second-order Arithmetic

This section examines Girard's proof to elucidate exactly where the ideas go beyond second-order arithmetic. The key observations are in Section 4.2: we cannot express the membership $\mathrm{RED}_T[\mathcal{R}/\underline{X}]$ as a predicate in the "codes" of T and the substitution. This links fairly explicitly to Gödel's incompleteness theorem, which demonstrates that *any* proof of strong normalisation must go beyond second-order.

4.1 Assumptions

Second-order arithmetic is a logical system dealing with natural numbers and subsets thereof. Without going into detail, it consists of the usual first order Peano axoims, where in the induction we are allowed to quantify over set variables, and a comprehension scheme:

$$\exists Z \forall n (n \in Z \iff \phi(n)) \tag{2}$$

where Z does not appear free in the predicate ϕ. Capital letters indicate set variables, and lower case for integers. Predicates are denoted with Greek letters.

Most of the above proof can be performed inside second-order arithmetic by coding terms and types as integers. Rather than doing this entirely formally, we assume given such a coding, and use the common-sense principles of it. It is instructive to lay out what these principles are. Throughout, we ignore possible issue of clashes in coding: for example, if we have systems to code two distinct concepts as integers, we may assign one to the multiples of 2, and the other to the multiples of 3.

- As $\mathbb{N} \times \mathbb{N}$ is countable, we can code pairs of integers, and as such finite sequences, as integers. We may encode functions (partial or total) $\mathbb{N} \to \mathbb{N}$ as sets of integers, using a graph.

- We assume given a coding of terms and of types into integers (as variable sets are taken to be countable). The predicates, "x codes a type", "y codes a term" and "x codes a term of type y" should be expressible. Similarly, statements about the construction of terms, for example "x, y and z code terms u, v, t, and $t = uv$", should be expressible, as should the sets of free term and type variables.

- By induction, we can define the set corresponding to the complexity function.

- Statements about one-step β-reduction should be encodable. For example, "x and y code for terms, and $x \to_\beta y$". We should also be able to express that a term is normal (or neutral).

- Using this, we can reason about strong normalisation. First, a sequence (set coding a function $\mathbb{N} \to \mathbb{N}$) of terms is a reduction path for x if the following holds (loosely):

$$(0, x) \in X \wedge ((n, y), (n+1, y') \in X \implies y \to_\beta y')$$

- Using this, we can code for the predicate "x codes a term which is strongly normalising", by quantification over such sets. The statement "the sequence X is infinite" is expressible. In a similar way, we can express the function $\nu(x)$, as the length of a sequence is definable.

Now we closely examine the proofs of Section 3 to see what can and can't be done in second-order arithmetic. The short answer is that there is no second-order formula in T and η defining a set $\mathrm{RED}_T[\eta]$ which satisfies the definition of Lemma 3.4. (Specifically, T is the problem.)

4.2 What we can't do

As mentioned, it is in defining $\mathrm{RED}_T[\eta]$ that we run into trouble: this discussion is an adaption of Remark 4.B.7 in [Gir87] (p.254-5), there applied to a first order system of natural deduction. For a fixed type, we are okay. We can define the substitution η as a relation (subset of $\mathbb{N} \times \mathbb{N}$), where $(x, y) \in \eta$ if x codes a type variable X_i, and y codes for a term in the chosen set $\mathcal{R}_i$. Then we can write down the comprehension for the set $\mathrm{RED}_T[\eta]$ as a sequence of first- and second-order quantifications. Take Example 3.6: in words, a term $t \in \mathrm{RED}_{\Pi X.X \to X}[-]$ if for every type V, reducibility candidate $\mathcal{S}$ of that type, and term v of type V, the term $tVv \in \mathcal{S}$. In symbols (glossing over predicates like "codes a type"):

$$\mathrm{RED}_{\Pi X.X \to X}[-] = \{t \mid \forall V, \mathcal{S}, v\, (v \in \mathcal{S} \implies tVv \in \mathcal{S})\}$$

In more complicated cases, we can still define extensions $\eta + \mathcal{S}/Y$, so this format works. However, we *cannot* translate this into a general comprehension, which constructs the set $\mathrm{RED}_T[\eta]$ given the codes of the type T and set η. (That is, with the codes of T and η as free variables in the predicate ϕ of eq. (2).) This comes from the fact that the length of the defining predicate ($\forall V, \mathcal{S}, v\, (v \in \mathcal{S} \implies tVv \in \mathcal{S})$ above) increases with the complexity of the type T.

Let us look at this in the context of Lemma 3.4. For a predicate ϕ, with n a free number variable (and other free number and set variables $\underline{m}$ and $\underline{X}$), the general second-order induction scheme is:

$$\forall \underline{m} \forall \underline{X}\, (\phi(0) \implies \forall n(\phi(n) \implies \phi(n+1)) \implies \forall n \phi(n))$$

The set of eq. (1) is uncountable, but defining the set:

$$\mathrm{RED}^n[\eta] = \bigcup_{c(T)<n} \{T\} \times \mathrm{RED}_T[\eta] \tag{3}$$

we then induct on the predicate:

$$\phi(n) = \forall \eta \exists_1 \mathrm{RED}^n[\eta]$$

However, while this argument proves that $\mathrm{RED}_T[\eta]$ exists, it does not express membership of that set as a predicate. For example, if $T = \Pi Y.W$, with $c(W) = n$, then assuming $\phi(n)$, and given η we may define $\mathrm{RED}_T[\eta]$ as:

$$t \in \mathrm{RED}_T[\eta] \iff \forall V, \mathcal{S}\,((W, tV) \in \mathrm{RED}^n[\eta + \mathcal{S}/Y]).$$

This is, effectively, constructing by induction a predicate specific to T. Once we have our predicate for W (inside the paratheses above), we tack on quantifier to make it suitable for T. This, I think, is the problem with formalising Girard's proof in second-order arithmetic.

Remark 4.1. We can make sense of this discussion in terms of Gödel incompleteness. The problem with formalising our proof is that the defining predicate for $\mathrm{RED}_T[\eta]$ varies with the structure of our type T. A predicate is just a finite list of symbols, which can therefore be coded in second-order arithmetic, and it is reasonable to assume that we may represent the function $\mathfrak{p}$ assigning a type T to the code of its predicate (which has one free number variable, for t, and one set, for η).

Under this assumption, our problem reduces to find a predicate $\mathcal{E}_{m,n}(A, \underline{x}, \underline{X})$ which is true whenever A codes for a predicate with free number and set variables $x_1, \ldots, x_m$ and $X_1, \ldots, X_n$, and such a predicate is true evaluated on the given values. In this case, we define

$$\mathrm{RED}_T[\eta] = \{t \mid \mathcal{E}_{1,1}(\mathfrak{p}(T), t, \eta)\}$$

Glossing over the fact we only need $\mathcal{E}$ to work for certain predicates $\mathfrak{p}(T)$, such an $\mathcal{E}$ cannot exist. We have not properly discussed this, but proofs in second-order arithmetic can be represented as finite labelled trees, and thus as an integer. Checking the validity of the tree represented by a code should also be possible, as should inducting on their heights. Let $\mathcal{C}(\tau)$ be the predicate which is the conclusion of the tree coded by τ, and $h(\tau)$ its height. But then, induction on the predicate:

$$\forall \tau (h(\tau) < n \implies \mathcal{E}_{0,0}(\mathcal{C}(\tau)))$$

amounts to a proof that the deduction rules are sound, which can be done. But this proves exactly the consistency of second-order arithmetic: our system proves no false formula. As such, $\mathcal{E}$ directly contradicts the Second Incompleteness Theorem. In this way, the analogy between the proofs of strong normalisation and consistency becomes clearer. In order to prove these statements, we must induct on a statement which is too strong to be expressed in the system. For strong normalisation, it is $\mathrm{RED}_T[\eta]$, as in Theorem 3.12, and for consistency it is some "truth predicate".

4.3 What we can do

The above establishes that, inside second-order arithmetic, we cannot prove strong normalisation for all of System F. (More properly, this fact follows from Gödel's second incompleteness theorem, but the above makes sense of this claim for this specific proof.) What we can do, however, is demonstrate that a particular term is strongly normalising.

The key here is that we only need to think about (apply the defining properties of) the sets $\mathrm{RED}_T[-]$ for a finite number of types T (note here that it is okay for the substitution to vary, because the predicate changes with the type). This is fine in second-order arithmetic, as we can just write down those reducibilities explicitly. With this in mind, first we examine the proofs of Section 3, under the assumption that the defining properties of $\mathrm{RED}_T[-]$ hold; then, we will tally up how many of these sets we actually deal with, for a given term.

The basic properties of reducibility candidates (in Section 3.2) go through okay: we can reason about strong normalisation and the function ν as remarked above. Assuming given the sets $\mathrm{RED}_T[\eta]$, we can prove that they are reducibility candidates (Lemma 3.7). Moreover, we only apply the inductive hypothesis to immediate subtypes (this also includes Lemma 3.3). In a similar way, Lemma 3.8 goes through. Again, we only deal with the subtypes (substituted and un-substituted) of the type T in question. (NB: this would become a problem, if applied to infinitely many of these substitutions.)

Likewise, the proofs of Lemmas 3.9 to 3.11 go through on the back of the properites of $\mathrm{RED}_T[\eta]$. Now we fix a term t, and trace the induction of Theorem 3.12 to check that we only deal with finitely many reducibility predicates. Mostly this amounts to Girard's remark that we only need the reducibility predicates of our term and its subterms. Proceed by induction, letting (t) be the collection of types whose reducibility we interrogate.

- If t is a variable, then we only see $\mathrm{RED}_{U_i}[\mathcal{R}/\underline{X}]$, so $(t) = \{U_i\}$.
- If $t = uv$ is an abstraction, then we prove $t \in \mathrm{RED}_T[\mathcal{R}/\underline{X}]$ by appealing to the fact that u and v satisfy their respective reducibility predicates, meaning $(t) = (u) \cup (v) \cup \{T\}$.
- When $t = \lambda y^V.w$, we quantify over different substitutions v/y, but we only appeal to membership of $\mathrm{RED}_W[\mathcal{R}/\underline{X}]$, so $(t) = (w) \cup \{T\}$.
- For $t = t'V$, with t' of type $\Pi Y.T'$, we apply Lemma 3.8 (via Lemma 3.10), but only to the specific substitution $T = T'[V/Y]$. It doesn't go through slickly in our notation, but the set (t) extends (t') by $\{T\}$, as well as the types required for Lemma 3.8. These are simply the subtypes of T', with and without our specific substitution.
- Examining Lemma 3.11, we see that when $t = \Pi Y.w$, $(t) = (w) \cup \{T\}$.

At each stage, we take the union of finite sets, so (t) is finite for all terms t. This, I think, justifies the claim that we can prove strong normalisation for a given term.

References

[Gal90] Jean Gallier. On girards candidats de reductibilite. 01 1990.

[Gir87] J.Y. Girard. *Proof Theory and Logical Complexity.* Number v. 1 in Proof Theory and Logical Complexity. Elsevier, 1987.

[Gir11] J. Girard. The blind spot: Lectures on logic. 2011.

[GLT93] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and types.* Cambridge Univ. Press, 1993.

[Sel08] Peter Selinger. Lecture notes on the lambda calculus. *CoRR*, abs/0804.3434, 2008.

[Tai67] W. W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2):198–212, 1967.