Two *Different* Strong Normalization Proofs?
— computability versus functionals of finite type —

Jaco van de Pol

Department of Philosophy, Utrecht University
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
E-mail: jaco@phil.ruu.nl

Abstract. A proof of $\forall t \exists n \mathrm{SN}(t,n)$ (term t performs at most n reduction steps) is given, based on strong computability predicates. Using modified realizability, a bound on reduction lengths is extracted from it. This upper bound is compared with the one Gandy defines, using strictly monotonic functionals. This reveals a remarkable connection between his proof and Tait's. We show the details for simply typed λ-calculus and Gödel's T. For the latter system, program extraction yields considerably sharper upper bounds.

1 Introduction

The purpose of this paper is to compare two different methods to prove strong normalization (SN). The first method uses the notion of ***strong computability predicates.*** This method is attributed to Tait [9], who used convertibility predicates to prove a normal form theorem for various systems. The other method to prove strong normalization uses functionals of finite type. To each typed term a functional of the same type is associated. This functional is measured by a natural number. In order to achieve that a rewrite step gives rise to a decrease of the associated number, the notion *strictly monotonic functional* is developed. The number is an upper bound for the length of reduction sequences starting from a certain term. This method was invented by Gandy [2].

In the literature, these two methods are often put in contrast (e.g. [2, § 6.3] and [3, § 4.4]). Using functionals seems to be more transparent and economizes on proof theoretical complexity; strong computability should generalize to more complex systems. On the other hand, seeing the two proofs one gets the feeling that "somehow, the same thing is going on". Indeed De Vrijer [11, § 0.1] remarks that a proof using strong computability can be seen as abstracting from concrete information in the functionals that is not strictly needed in a termination proof, but which provides for an estimate of reduction lengths.

In this paper we will substantiate this feeling. First we will ***decorate*** the proof à la Tait with concrete numbers. This is done by introducing binary predicates $\mathrm{SN}(t,n)$, which mean that the term t may perform at most n reduction steps. A formal, constructive proof of $\exists n \mathrm{SN}(t,n)$ is given for any t. From this proof, we extract a program, via the ***modified realizability interpretation.*** Remarkably, this program equals (more or less) the functional assigned to the term t in the proof à la Gandy.

The paper is organized as follows. In Section 3, we decorate Tait's SN-proof for simply typed λ-calculus. Modified realizability is introduced in Section 4. In Section 5 the proofs of Section 3 are formalized; also the program extraction is carried out there. In Section 5.3, the extracted functionals are compared with those used by Gandy. The same project is carried out for Gödel's T in Section 6. Other possible extensions are considered in Section 7.

The idea of using a realizability interpretation to extract functionals from Tait's SN-proof already occurs in [1]. In that paper, the *normal form* of a term is extracted. The contribution of this paper is, that by extracting numerical upper bounds for the length of reduction sequences, a comparison with Gandy's proof can be made. Furthermore, we also deal with Gödel's T, which yields a sharper upper bound than provided by Gandy's proof. The author is grateful to Ulrich Berger for discussions on the subject, and to Marc Bezem and Jan Springintveld for reading and improving preliminary versions of the paper.

2 Simply Typed λ-Calculus

The set of *simple types* contains a certain set of base types and is closed under the binary operator $\rightarrow$. By convention, metavariables $\iota, \iota_1, \cdots$ range over base types; $\rho, \sigma, \tau, \cdots$ over arbitrary types. The adjective "simple" will often be dropped.

The set of *simply typed λ-terms* contains a certain set of typed variables and is closed under typed application and λ-binding. We reserve $r, s, t, \cdots$ for arbitrary simply typed terms, and $x^\sigma, y^\tau, \cdots$ for typed variables. Constants can be seen as variables that will not occur after a λ. With *terms* we will mean simply typed λ-terms. To indicate that r has type σ, we write r^σ. The typing rules are as follows:

1. A variable (or constant) x^σ is of type σ.
2. If s and t are of type $\rho \rightarrow \sigma$ and ρ, respectively, then (st) is of type σ.
3. If x^σ is a variable and s is of type τ, then $(\lambda x^\sigma s)$ is of type $\sigma \rightarrow \tau$.

Type decoration and outer brackets are often omitted.

Standard notions of bound and free variables ($\mathrm{FV}(s)$) will be used. We will identify α-convertible terms (i.e. terms that are equal up to the names of the bound variables). Substituting a term t for the free occurrences of x in a term s is denoted by $s[x := t]$. Renamings to avoid unintended capture of free variables are performed automatically.

The binary rewrite relation $s \rightarrow_\beta t$ is defined as the compatible closure of the β-rule: $(\lambda xs)t \mapsto s[x := t]$. We write $s \rightarrow_\beta^n t$, if there is a reduction sequence from s to t of n steps.

Definition 1. (SN for simply typed λ-calculus)

1. A term t is *strongly normalizing*, denoted by $\mathrm{SN}(t)$, if every reduction sequence $t \equiv s_0 \rightarrow_\beta s_1 \rightarrow_\beta \cdots$ is finite.

2. A term is *strongly normalizing in at most n steps* (SN(t,n)) if every reduction sequence out of t is finite, and has length at most n.

In the next section, we present the proof à la Tait, that every simply typed λ-term has an upper bound n such that it is strongly normalizing in at most n steps. By König's Lemma, this is equivalent to strong normalization, because the β-reduction relation is finitely branching.

We will often abbreviate a sequence of terms $t_1,\ldots,t_n$ by $\bar{t}$. In the same way, sequences of variables $\bar{x}$ or types $\bar{\sigma}$ will occur frequently. The length of such sequences is implicitly known, or unimportant. The empty sequence is denoted by ϵ. The *simultaneous* substitution of the variables $\bar{x}$ by the terms $\bar{t}$ in a term s is denoted by $s[\bar{x} := \bar{t}]$.

By convention, $\bar{\sigma} \to \tau$ means $\sigma_1 \to (\sigma_2 \to \cdots \to (\sigma_n \to \tau))$ and $s\bar{t}$ means $(((st_1)t_2)\cdots t_n)$ and $\lambda\bar{x}.t$ means $(\lambda x_1(\lambda x_2 \cdots (\lambda x_n t)))$. We also use the following (less standard) notation for sequences of variables, terms and types:

- $\bar{x}^{\bar{\sigma}} \equiv x_1^{\sigma_1},\cdots,x_n^{\sigma_n}$
- $\bar{\sigma} \to \bar{\tau} \equiv \bar{\sigma} \to \tau_1,\ldots,\bar{\sigma} \to \tau_n$
- $\bar{s}\bar{t} \equiv s_1\bar{t},\ldots,s_n\bar{t}$
- $\lambda\bar{x}.\bar{t} \equiv \lambda\bar{x}.t_1,\ldots,\lambda\bar{x}.t_n$

Note that by this convention $\bar{\sigma} \to \epsilon \equiv \epsilon$ (in particular, $\epsilon \to \epsilon \equiv \epsilon$) and $\epsilon \to \sigma \equiv \sigma$.

3 Informal SN-Proof à la Tait

Tait's method to prove strong normalization starts with defining a "strong computability" predicate which is stronger than "strong normalizability". The proof consists of two parts: One part stating that strongly computable terms are strongly normalizing, and one part stating that any term is strongly computable. The first is proved with induction on the types (simultaneously with the statement that every variable is strongly computable). The second part is proved with induction on the term structure (in fact a slightly stronger statement is proved). We will present a version of this proof that contains information about reduction lengths.

Definition 2. The set of *strongly computable* terms is defined inductively as follows:

(i) SC$_\iota(t)$ iff there exists an n such that SN(t,n).
(ii) SC$_{\sigma\to\tau}(t)$ iff for all s with SC$_\sigma(s)$, SC$_\tau(ts)$.

Lemma 3 (SC Lemma). *(a) For all terms t, if* SC(t) *then there exists an n with* SN(t,n).
(b) For all terms t of the form $x\bar{t}$, if there exists an n with SN(t,n), *then* SC(t)

In (b), $\bar{t}$ may be the empty sequence.

Proof. (Simultaneous induction on the type of t)

(a) Assume $\mathrm{SC}(t)$.
If t is of base type, then $\mathrm{SC}(t)$ just means that there exists an n with $\mathrm{SN}(t,n)$. If t is of type $\sigma \to \tau$, we take a variable x^σ, which is of the form $x\bar{t}$. Note that x is in normal form, hence $\mathrm{SN}(x,0)$ holds. By IH(b), $\mathrm{SC}(x)$; and by the definition of $\mathrm{SC}(t)$, $\mathrm{SC}(tx)$. By IH(a) we have that there exists an n such that $\mathrm{SN}(tx,n)$. We can take this n, because any reduction sequence from t gives rise to a sequence from tx of the same length. Hence $\mathrm{SN}(t,n)$ holds.

(b) Assume that $t = x\bar{t}$ and $\mathrm{SN}(t,n)$ for some n.
If t is of base type, then the previous assumption forms exactly the definition of $\mathrm{SC}(t)$.
If t has type $\sigma \to \tau$, assume $\mathrm{SC}(s)$ for arbitrary s^σ. By IH(a), $\mathrm{SN}(s,m)$ for some m. Because reductions in $x\bar{t}s$ can only take place inside $\bar{t}$ or s, we have $\mathrm{SN}(x\bar{t}s, m+n)$. IH(b) yields that $\mathrm{SC}(x\bar{t}s)$. This proves $\mathrm{SC}(t)$.

□

Lemma 4 (Abstraction Lemma). *For all terms s,t and $\bar{r}$ and variables x, it holds that if $\mathrm{SC}(s[x:=t]\bar{r})$ and $\mathrm{SC}(t)$, then $\mathrm{SC}((\lambda x.s)t\bar{r})$.*

Proof. (Induction on the type of $s\bar{r}$.) Let s, x, t and $\bar{r}$ be given, with $\mathrm{SC}(s[x:=t]\bar{r})$ and $\mathrm{SC}(t)$. Let σ be the type of $s\bar{r}$.

If $\sigma = \iota$, then by definition of SC, we have an n such that $\mathrm{SN}(s[x:=t]\bar{r},n)$. By Lemma 3(a) we obtain the existence of m, such that $\mathrm{SN}(t,m)$. We have to show, that there exists a p with $\mathrm{SN}((\lambda x.s)t\bar{r},p)$. We will show that we can put $p := m+n+1$. Consider an arbitrary reduction sequence of $(\lambda x.s)t\bar{r}$. Without loss of generality, we assume that it consists of first a steps in s (yielding s'), b steps in $\bar{r}$ (yielding $\bar{r}'$) and c steps in t (yielding t'). After this the outermost redex is contracted, yielding $s'[x:=t']\bar{r}'$, and finally d steps occur. Clearly, $c \le m$. Notice that we also have a reduction sequence $s[x:=t]\bar{r} \to^* s'[x:=t']\bar{r}'$ of at least $a+b$ steps (we cannot count reductions in t, because we do not know whether x occurs free in s). So surely, $a+b+d \le n$. Summing this up, we have that any reduction sequence from $(\lambda x.s)t\bar{r}$ has length at most $m+n+1$.

Let $\sigma = \rho \to \tau$. Assume $\mathrm{SC}(r)$, for arbitrary r^ρ. Then by definition of $\mathrm{SC}(s[x:=t]\bar{r})$, we have $\mathrm{SC}_\tau(s[x:=t]\bar{r}r)$, and by IH $\mathrm{SC}((\lambda x.s)t\bar{r}r)$. This proves $\mathrm{SC}((\lambda x.s)t\bar{r})$. □

In the following lemma, θ is a substitution, i.e. a finite mapping from variables into terms.

Lemma 5 (Main Lemma). *For all terms t and substitutions θ, if $\mathrm{SC}(x^\theta)$ for all free variables x of t, then $\mathrm{SC}(t^\theta)$.*

Proof. (Induction on the structure of t.) Let t and θ be given, such that $\mathrm{SC}(x^\theta)$ for all $x \in \mathrm{FV}(t)$.

If $t = x$, then the last assumption yields $\mathrm{SC}(t^\theta)$.

If $t = rs$, we have $\mathrm{SC}(r^\theta)$ and $\mathrm{SC}(s^\theta)$ by IH for r and s. Then by definition of $\mathrm{SC}(r^\theta)$, we have $\mathrm{SC}(r^\theta s^\theta)$, hence by equality of $r^\theta s^\theta$ and $(rs)^\theta$, $\mathrm{SC}(t^\theta)$ follows.

If $t = \lambda x.s$, assume that $\mathrm{SC}(r)$ for an arbitrary r. By IH for s, applied on the substitution $\theta[x := r]$, we see that $\mathrm{SC}(s^{\theta[x:=r]})$, hence by equality $\mathrm{SC}((s^\theta)[x := r])$. Now we can apply Lemma 4, which yields that $\mathrm{SC}((\lambda x.s^\theta)r)$. Again by using equality, we see that $\mathrm{SC}((\lambda x.s)^\theta r)$ holds. This proves $\mathrm{SC}((\lambda x.s)^\theta)$. (Note that implicitly renaming of bound variables is required.) □

Theorem 6. *For any term t there exists an n, such that* $\mathrm{SN}(t, n)$.

Proof. Let θ be the identity substitution, with as domain the free variables of t. By Lemma 3(b), $\mathrm{SC}(x)$ is guaranteed. Now we can apply Lemma 5, yielding $\mathrm{SC}(t^\theta)$. Because $t^\theta = t$, we obtain $\mathrm{SC}(t)$. Lemma 3(a) yields the existence of an n with $\mathrm{SN}(t, n)$. □

4 A Variant of Modified Realizability

As mentioned before, we want to extract the computational content from the SN-proof of Section 3. To this end we use *modified realizability*, introduced by Kreisel [4]. In [10, § 3.4] modified realizability is presented as a translation of HA^ω into itself. This interpretation eliminates existential quantifiers, at the cost of introducing functions of finite type (functionals), represented by λ-terms.

Following Berger [1], we present modified realizability as an interpretation of a first order fragment (MF) into a higher-order, negative (i.e. $\exists$-free) fragment (NH). We will also take over a refinement by Berger, which treats specific parts of a proof as *computationally irrelevant.*

4.1 The Modified Realizability Interpretation

A formula can be seen as the specification of a program. E.g. $\forall x \exists y.P(x, y)$ specifies a program f of type $o \to o$, such that $\forall x.P(x, f(x))$ holds. In general a sequence of programs is specified.

A refinement by Berger enables to express that existentially quantified variables are independent of certain universal variables, by underlining the universal ones. In $\underline{\forall x} \exists y.P(x, y)$ the underlining means that y is not allowed to depend on x. So a number m is specified, with $\forall x.P(x, m)$. This could of course also be specified by the formula $\exists y \forall x.P(x, y)$, but in specifications of the form $\underline{\forall x}.P(x) \to \exists y.Q(x, y)$ the underlining cannot be eliminated that easily. This formula specifies a number m, such that $\forall x.P(x) \to Q(x, m)$ holds. The $\underline{\forall x}$ cannot be pushed to the right, nor can the $\exists y$ be pulled to the left, without changing the intuitionistic meaning.

Specifications are expressed in minimal many-sorted first-order logic (MF). This logic is based upon a many-sorted first-order signature. Terms over such a signature are defined as usual ($a, b, c, \ldots$ denote arbitrary terms). The formulae of MF are either atomic ($P\vec{a}$), or of the form $\varphi \to \psi$, $\forall x^\iota \varphi$, $\underline{\forall x^\iota} \varphi$ or $\exists x^\iota \varphi$. Here $\varphi, \psi, \ldots$ denote arbitrary MF formulae. This logic is Minimal, because negation is not included, and it deals with First-order objects only.

As programming language, we use the simply typed λ-calculus. Because programs are higher-order objects, MF cannot talk about them. To express correctness of programs, we introduce **N**egative **H**igher-order logic (NH). The terms of NH are simply typed λ-terms *considered modulo β*, with the MF sorts as base types, MF function symbols as constants and with the MF predicate symbols. The formulae are atomic ($P\overline{s}$), or composed with $\varphi \to \psi$ or $\forall x^\rho \varphi$. Here $\varphi, \psi, \ldots$ denote arbitrary NH formulae. Negative means that there are no existential quantifiers, and Higher-order refers to the objects.

Below we define $\tau(\varphi)$, the sequence of types of the programs specified by the MF formula φ. This operation is known as "forgetting dependencies" (of types on terms). Furthermore, if $\overline{s}$ is a sequence of programs of type $\tau(\varphi)$, we define an NH formula $\overline{s}\ \mathbf{mr}\ \varphi$ (**m**odified **r**ealizes). This NH formula expresses correctness of $\overline{s}$ with respect to the specification φ.

Definition 7. (modified realizability interpretation)

$$
\begin{array}{rlrl}
\tau(P\overline{a}) & := \epsilon & \epsilon\ \mathbf{mr}\ P\overline{a} & := P\overline{a} \\
\tau(\varphi \to \psi) & := \tau(\varphi) \to \tau(\psi) & \overline{s}\ \mathbf{mr}\ \varphi \to \psi & := \forall \overline{x}^{\tau(\varphi)}(\overline{x}\ \mathbf{mr}\ \varphi) \to (\overline{s}\overline{x}\ \mathbf{mr}\ \psi) \\
\tau(\forall x^\iota \varphi) & := \iota \to \tau(\varphi) & \overline{s}\ \mathbf{mr}\ \forall x^\iota \varphi & := \forall x^\iota(\overline{s}x\ \mathbf{mr}\ \varphi) \\
\tau(\underline{\forall x^\iota}\varphi) & := \tau(\varphi) & \overline{s}\ \mathbf{mr}\ \underline{\forall x^\iota}\varphi & := \forall x^\iota(\overline{s}\ \mathbf{mr}\ \varphi) \\
\tau(\exists x^\iota \varphi) & := \iota, \tau(\varphi) & r, \overline{s}\ \mathbf{mr}\ \exists x^\iota \varphi(x) & := \overline{s}\ \mathbf{mr}\ \varphi(r)
\end{array}
$$

In the **mr**-clauses, x^ι should not occur in $\overline{s}$ and $\overline{x}$ should be fresh. Note that only existential quantifiers give rise to a longer sequence of types. In particular, if φ has no existential quantifiers, then $\tau(\varphi) \equiv \epsilon$. (We use that $\overline{\sigma} \to \epsilon \equiv \epsilon$). Nested implications give rise to arbitrarily high types. In $\underline{\forall x^\iota}\varphi$, the program specified by φ may not depend on x, so the "$\iota \to$" is discarded in the τ-clause. In the **mr**-clause, the programs $\overline{s}$ do not get x as input, as intended. But to avoid that x becomes free in φ, we changed Berger's definition by adding $\forall x^\iota$.

By induction on the MF formula φ one sees that if $\overline{s}$ is of type $\tau(\varphi)$, then $\overline{s}\ \mathbf{mr}\ \varphi$ is a correct formula of NH, so in particular, it will not contain $\exists$- and $\underline{\forall}$-quantifiers (nor of course the symbol **mr**).

4.2 Derivations and Program Extraction

In the previous section we introduced the formulae of MF, the formulae of NH and a translation of the former into the latter. In this section we will introduce proofs for MF and for NH. The whole point will be, that from an MF proof of φ a program can be extracted, together with an NH proof that this program meets its specification φ.

Proofs are formalized by derivation terms, a linear notation for natural deduction. Derivation terms are defined as the least set containing assumption variables ($u^\varphi, v^\psi, \ldots$) and closed under certain syntactic operations. To express some side conditions, the sets of assumption variables ($\mathrm{FA}(d)$) and of computational relevant variables ($\mathrm{CV}(d)$) are defined simultaneously. By convention, x and y range over object variables. We let d, e range over derivations.

The introduction rule for the $\underline{\forall}$-quantifier has an extra proviso: we may only extend a derivation d of φ to one of $\underline{\forall x}\varphi$, if x is not *computationally relevant* in d. Roughly speaking, all free object variables of d occurring as argument of a $\forall$-elimination or as witness in an $\exists$-introduction are computationally relevant.

Definition 8. (derivations, free assumptions, computational relevant variables)

$$\begin{array}{lll}
\mathit{ass}: u^{\varphi} & \mathrm{FA}(u) = \{u\} & \mathrm{CV}(u) = \emptyset \\
\to^{+}: (\lambda u^{\varphi} d^{\psi})^{\varphi\to\psi} & \mathrm{FA}(\lambda u d) = \mathrm{FA}(d) \setminus \{u\} & \mathrm{CV}(\lambda u d) = \mathrm{CV}(d) \\
\to^{-}: (d^{\varphi\to\psi} e^{\varphi})^{\psi} & \mathrm{FA}(de) = \mathrm{FA}(d) \cup \mathrm{FA}(e) & \mathrm{CV}(de) = \mathrm{CV}(d) \cup \mathrm{CV}(e) \\
\forall^{+}: (\lambda x^{\sigma} d^{\varphi})^{\forall x^{\sigma}\varphi} & \mathrm{FA}(\lambda x d) = \mathrm{FA}(d) & \mathrm{CV}(\lambda x d) = \mathrm{CV}(d) \setminus \{x\} \\
\qquad \text{provided (1)} & & \\
\forall^{-}: (d^{\forall x^{\sigma}\varphi(x)} a^{\sigma})^{\varphi(a)} & \mathrm{FA}(da) = \mathrm{FA}(d) & \mathrm{CV}(da) = \mathrm{CV}(d) \cup \mathrm{FV}(a) \\
\underline{\forall}^{+}: (\underline{\lambda x}^{\sigma} d^{\varphi})^{\underline{\forall x}^{\sigma}\varphi} & \mathrm{FA}(\underline{\lambda x} d) = \mathrm{FA}(d) & \mathrm{CV}(\underline{\lambda x} d) = \mathrm{CV}(d) \\
\qquad \text{provided (2)} & & \\
\underline{\forall}^{-}: (d^{\underline{\forall x}^{\sigma}\varphi(x)} \underline{a}^{\sigma})^{\varphi(a)} & \mathrm{FA}(d\underline{a}) = \mathrm{FA}(d) & \mathrm{CV}(d\underline{a}) = \mathrm{CV}(d) \\
\exists^{+}: (\exists^{+}[a^{\sigma}; d^{\varphi(a)}])^{\exists x^{\sigma}\varphi(x)} & \mathrm{FA}(\exists^{+}[a;d]) = \mathrm{FA}(d) & \mathrm{CV}(\exists^{+}[a;d]) \\
 & & \quad = \mathrm{CV}(d) \cup \mathrm{FV}(a) \\
\exists^{-}: (\exists^{-}[d^{\exists x^{\sigma}\varphi(x)}; y; u^{\varphi(y)}; e^{\psi}])^{\psi} & \mathrm{FA}(\exists^{-}[d;y;u;e]) & \mathrm{CV}(\exists^{-}[d;y;u;e]) \\
\qquad \text{provided (3)} & \quad = \mathrm{FA}(d) \cup (\mathrm{FA}(e) \setminus \{u\}) & \quad = \mathrm{CV}(d) \cup (\mathrm{CV}(e) \setminus \{y\})
\end{array}$$

where the provisos are:

(1) $x \notin \mathrm{FV}(\psi)$ for any $u^{\psi} \in \mathrm{FA}(d)$.
(2) $x \notin \mathrm{FV}(\psi)$ for any $u^{\psi} \in \mathrm{FA}(d)$ and moreover, $x \notin \mathrm{CV}(d)$.
(3) $y \notin \mathrm{FV}(\psi)$ and $y \notin \mathrm{FV}(\chi)$ for all $v^{\chi} \in \mathrm{FA}(e) \setminus \{u\}$.

An MF-derivation is a derivation with all quantifier rules restricted to base types. An NH-derivation is a derivation without the $\underline{\forall x}$ and the $\exists$-rules. We will write $\Phi \vdash_{\mathrm{MF}} \psi$ if there exists a derivation d^{ψ}, with all free assumptions among Φ. Likewise for $\vdash_{\mathrm{NH}}$.

From MF-derivations, we can read off a program and a correctness proof for this program. This is best illustrated by the $\exists^{+}$ rule: If we use this rule to prove $\exists x \varphi(x)$, then we immediately see the witness a and a proof d of $\varphi(a)$. In genera we can define $\mathbf{ep}(d)$, the sequence of extracted programs from a derivation ϵ To deal with assumption variables in d, we fix for every assumption variabl u^{φ} a sequence of object variables $\overline{x}_u^{\tau(\varphi)}$. The extracted program is defined with respect to this choice.

Definition 9. (extracted program from MF-derivations)

$$\begin{array}{rlrl}
\mathbf{ep}(u^{\varphi}) := & \overline{x}_u^{\tau(\varphi)} & \mathbf{ep}(d^{\forall x^{\iota}\varphi(x)} a^{\iota}) := & \mathbf{ep}(d)a \\
\mathbf{ep}(\lambda u^{\varphi} d^{\psi}) := & \lambda \overline{x}_u^{\tau(\varphi)} \mathbf{ep}(d) & \mathbf{ep}(\underline{\lambda x}^{\iota} d^{\varphi}) := & \mathbf{ep}(d) \\
\mathbf{ep}(d^{\varphi\to\psi} e^{\varphi}) := & \mathbf{ep}(d)\mathbf{ep}(e) & \mathbf{ep}(d^{\underline{\forall x}^{\iota}\varphi(x)} \underline{a}^{\iota}) := & \mathbf{ep}(d) \\
\mathbf{ep}(\lambda x^{\iota} d^{\varphi}) := & \lambda x^{\iota} \mathbf{ep}(d) & \mathbf{ep}(\exists^{+}[a^{\iota}; d^{\varphi(a)}]) := & a, \mathbf{ep}(d)
\end{array}$$

$$\mathbf{ep}(\exists^{-}[d; y; u^{\varphi(y)}; e^{\psi}]) := \mathbf{ep}(e)[y := s][\overline{x}_u := \overline{t}], \text{where } s, \overline{t} = \mathbf{ep}(d^{\exists x^{\iota}\varphi(x)})$$

The whole enterprise is justified by the following

Theorem 10 (Correctness [1]). *If d is an* MF *derivation of φ, then there exists an* NH *derivation $\mu(d)$ of* $\mathbf{ep}(d)$ **mr** φ. *Moreover, the only free assumptions in $\mu(d)$ are of the form $\overline{x}_u$* **mr** ψ, *for some assumption u^ψ occurring in d already.*

Proof. First the following facts are verified by induction on d:

1. $\mathrm{FV}(\mathbf{ep}(d)) \subseteq \bigcup\{\overline{x}_u | u \in \mathrm{FA}(d)\} \cup \mathrm{CV}(d)$.
2. $\mathbf{ep}(d^\varphi)$ is a sequence of terms of type $\tau(\varphi)$.

Then the existence of $\mu(d)$ can be proved by induction on d. We only deal with one typical case. See e.g. [1] for the other cases.

$$\mu(\underline{\lambda x^\iota} d) := \lambda x^\iota(\mu(d))$$

By induction hypothesis, we have $\mu(d)$ proves $\mathbf{ep}(d)$ **mr** φ. By the proviso of $\underline{\forall}^+$, $x \notin \mathrm{CV}(d)$, hence (by fact 1) $x \notin \mathrm{FV}(\mathbf{ep}(d))$. Furthermore, x does not occur in free assumptions of d, hence not in assumptions of $\mu(d)$, so $\lambda x \mu(d)$ is a correct derivation of $\forall x(\mathbf{ep}(d)$ **mr** $\varphi)$, which is equivalent (because $x \notin \mathbf{ep}(d)$, and using β-equality) to $\mathbf{ep}(\underline{\lambda x} d)$ **mr** $\underline{\forall x}\varphi$. □

4.3 On Using Axioms

If we use an axiom $\mathbf{ax}^\varphi$ (as open assumption) in a proof d of MF, then the extracted program $\mathbf{ep}(d)$ will probably contain the free variables $\overline{x}_{\mathbf{ax}}^{\tau(\varphi)}$ (as holes) and the correctness proof $\mu(d)$ may contain a free assumption $\overline{x}_{\mathbf{ax}}$ **mr** φ (according to Theorem 10).

The goal is to complete the program in a correct way. More specifically, we look for realizers $\overline{t}_{\mathbf{ax}}$, such that the NH formula $\overline{t}_{\mathbf{ax}}$ **mr** φ holds in our intended interpretation. If this succeeds, then the extracted program can be completed by taking $\mathbf{ep}(d)[\overline{x}_{\mathbf{ax}} := \overline{t}_{\mathbf{ax}}]$. This is correct, because the assumptions in the correctness proof $\mu(d)$ hold. We conclude that the justification of postulated principles should be given in terms of NH, because in this logic the correctness proofs live. We will now justify some typical principals. (See also [10] and [1].)[1]

$\exists$-free axioms. Let φ be such a formula. Then $\tau(\varphi) \equiv \epsilon$. The only candidate realizer is the empty sequence of programs. Note that the formula ϵ **mr** φ is obtained from φ by removing all underlinings. So a formula without existential quantifiers may be used as axiom, whenever it is true after removing all underlinings.

Equality axioms. Assume that $=$ is a binary predicate symbol. The usual axioms for symmetry, transitivity and reflexivity of $=$ are existential-free and are justified as above. The replacement scheme requires another justification. We introduce the axiom scheme

$$\mathbf{repl} : s = t \rightarrow \varphi(s) \rightarrow \varphi(t) \ .$$

[1] In the full version we regard some principles that depend on the underlining.

Note that $\tau(s = t \to \varphi(s) \to \varphi(t)) \equiv \tau(\varphi) \to \tau(\varphi)$. The identity can be taken as realizer, as the following calculation shows:

$$\begin{aligned}&\lambda \overline{x}^{\tau(\varphi)}.\overline{x}\ \mathbf{mr}\ (s = t \to \varphi(s) \to \varphi(t))\\ &\equiv s = t \to \forall \overline{x}.\overline{x}\ \mathbf{mr}\ \varphi(s) \to \overline{x}\ \mathbf{mr}\ \varphi(t),\end{aligned}$$

The latter NH formula is true, if we interpret $=$ by the identity relation. This means that we can use the replacement scheme in MF proofs. Its realizer is the identity on sequences.

5 Formalized Proofs and Extracted Programs

In this section the proof of Section 3 will be formalized in first-order predicate logic, as introduced in Section 4. This is not unproblematic as the informal proof contains induction on types and terms, which is not a part of the framework. This is solved by defining a series of proofs, by recursion over types or terms. In this way the induction is shifted to the metalevel. There is a price to be paid: instead of a uniform function U, such that $U(t)$ computes the desired upper bound for a term t, we only extract for any t an expression Upper$[t]$, which computes an upper bound for term t only. So here we lose a kind of uniformity. It is well known that the absence of a uniform first-order proof is essential, because the computability predicate is not arithmetizable [10, § 2.3.11].

Another incompleteness arises, because some combinatorial results will be plugged in as axioms. This second incompleteness is harmless for our purpose, because all these axioms are formulated without using existential quantifiers. Hence they are realized by the empty sequence (and finding formal proofs for these facts would be waste of time).

5.1 Fixing Signature and Axioms

As to the language, we surely have to represent λ-terms. To this end, we adopt for each type ρ new sorts $\mathcal{V}_\rho$ and $\mathcal{T}_\rho$, that interpret variables and terms *modulo α-conversion* of type ρ, respectively. Constants of sort $\mathcal{V}_\rho$ are added to represent variables (written $\ulcorner x \urcorner$). Function symbols for typed application and abstraction are included as well. With $\ulcorner s \urcorner$, we denote the representation of a λ-term s in this first-order language, using the following function symbols:

$\mathsf{V}_\rho : \mathcal{V}_\rho \to \mathcal{T}_\rho$, to inject variables into terms;
$\_ \bullet_{\rho,\sigma} \_ : \mathcal{T}_{\rho\to\sigma} \times \mathcal{T}_\rho \to \mathcal{T}_\sigma$, denoting typed application;
$\boldsymbol{\lambda}_{\rho,\sigma} : \mathcal{V}_\rho \times \mathcal{T}_\sigma \to \mathcal{T}_{\rho\to\sigma}$, denoting typed abstraction.

Note that e.g. $\ulcorner \lambda x.x \urcorner \equiv \boldsymbol{\lambda}(\ulcorner y \urcorner, \mathsf{V}(\ulcorner y \urcorner))$, for some arbitrary but fixed choice of y. Although the terms in the intended model are taken modulo α-conversion, the first-order terms cannot have this feature. We will also need function symbols to represent simultaneous substitution: for any sequence of types $\sigma, \tau_1, \ldots, \tau_n$, a symbol $\_(\_,\_,\ldots := \_,\_,\ldots)$ of arity $\mathcal{T}_\sigma \times \mathcal{V}_{\tau_1} \times \cdots \times \mathcal{V}_{\tau_n} \times \mathcal{T}_{\tau_1} \times \cdots \times \mathcal{T}_{\tau_n} \to \mathcal{T}_\sigma$. The intended meaning of $s(\overline{x} := \overline{t}\,)$ is the simultaneous substitution in s of x_i by

Due to the underlined quantifier, $\tau(\mathrm{SC}_\sigma(s)) \equiv \sigma'$, where σ' is obtained from σ by renaming base types ι to nat. The underlined quantifier takes care that numerical upper bounds only use numerical information about subterms: the existential quantifier hidden in $\mathrm{SC}(t \bullet s)$ can only use the existential quantifier in $\mathrm{SC}(s)$; not s itself. In fact, this is the reason for introducing the underlined quantifier.

Formalizing the SC Lemma. We proceed by formalizing Lemma 3. We will define proofs

$$\begin{aligned}&\Phi_\rho : \underline{\forall t}.\, \mathrm{SC}_\rho(t) \to \exists n \mathrm{SN}_\rho(t,n) \quad \text{and}\\ &\Psi_\rho : \underline{\forall x\bar{t}}.\, (\exists m \mathrm{SN}_\rho(\mathsf{V}(x) \bullet \bar{t}, m)) \to \mathrm{SC}_\rho(\mathsf{V}(x) \bullet \bar{t}\,)\end{aligned}$$

with simultaneous induction on ρ:

$$\begin{aligned}&\Phi_\iota := \underline{\lambda t}\lambda u^{\mathrm{SC}(t)} u\\ &\Phi_{\rho\to\sigma} := \underline{\lambda t}\lambda u^{\mathrm{SC}(t)}\\ &\qquad\qquad \exists^-[\Phi_\sigma \underline{(t \bullet \mathsf{V}(x))}\Big(u\underline{\mathsf{V}(x)}(\Psi_\rho \underline{x} \exists^+[0; (ax_1 \underline{x})])\Big);\\ &\qquad\qquad\quad m; v^{\mathrm{SN}(t\bullet\mathsf{V}(x),m)};\\ &\qquad\qquad\quad \exists^+[m; (ax_3 \underline{txm}v)]]\end{aligned}$$

$$\begin{aligned}&\Psi_\iota := \underline{\lambda x\bar{t}}\lambda u^{\exists m \mathrm{SN}(\mathsf{V}(x)\bullet\bar{t},m)} u\\ &\Psi_{\rho\to\sigma} := \underline{\lambda x\bar{t}}\ \lambda u^{\exists m \mathrm{SN}(\mathsf{V}(x)\bullet\bar{t},m)} \underline{\lambda s}\lambda v^{\mathrm{SC}(s)}\\ &\qquad\qquad \exists^-[u; m; u_0^{\mathrm{SN}(\mathsf{V}(x)\bullet\bar{t},m)};\\ &\qquad\qquad\quad \exists^-[(\Phi_\rho \underline{s} v); n; v_0^{\mathrm{SN}(s,n)};\\ &\qquad\qquad\qquad \Psi_\sigma \underline{x\bar{t}s}\ \exists^+[(m+n); (ax_2 \underline{x\bar{t}smn}u_0 v_0)]]]\end{aligned}$$

Having the concrete derivations, we can extract the computational content, using the definition of **ep**. Note that the underlined parts are discarded, and that an $\exists$-elimination gives rise to a substitution. The resulting functionals are $\mathbf{ep}(\Phi_\rho) : \rho \to \mathsf{nat}$ and $\mathbf{ep}(\Psi_\rho) : \mathsf{nat} \to \rho$,

$$\begin{aligned}\mathbf{ep}(\Phi_\iota) &= \lambda x_u x_u\\ \mathbf{ep}(\Phi_{\rho\to\sigma}) &= \lambda x_u m[m := \mathbf{ep}(\Phi_\sigma)(x_u(\mathbf{ep}(\Psi_\rho)0))]\\ &= \lambda x_u \mathbf{ep}(\Phi_\sigma)(x_u(\mathbf{ep}(\Psi_\rho)0))\\ \mathbf{ep}(\Psi_\iota) &= \lambda x_u x_u\\ \mathbf{ep}(\Psi_{\rho\to\sigma}) &= \lambda x_u \lambda x_v \mathbf{ep}(\Psi_\sigma)(m+n)[n := \mathbf{ep}(\Phi_\rho)x_v][m := x_u]\\ &= \lambda x_u \lambda x_v \mathbf{ep}(\Psi_\sigma)(x_u + (\mathbf{ep}(\Phi_\rho)x_v))\end{aligned}$$

Formalizing the Abstraction Lemma. We proceed by formalizing Lemma 4, which deals with abstractions. Let r have sort $\mathcal{T}_{\bar{\rho}\to\rho}$, and each r_i sort $\mathcal{T}_{\rho_i}$ (so $r \bullet \bar{r}$ has sort $\mathcal{T}_\rho$). Let s have sort $\mathcal{T}_\sigma$, y sort $\mathcal{V}_\sigma$, each t_i sort $\mathcal{T}_{\tau_i}$ and each x_i sort $\mathcal{V}_{\tau_i}$. We construct proofs

$$\begin{aligned}\Lambda_{\rho,\sigma,\bar{\rho},\bar{\tau}} : \underline{\forall r, y, \bar{x}, s, \bar{t}, \bar{r}}.\, \mathrm{SC}_\rho(r(y,\bar{x} := s,\bar{t}\,) \bullet \bar{r}\,) \to &\mathrm{SC}_\sigma(s) \to\\ &\mathrm{SC}_\rho(\boldsymbol{\lambda}(y,r)(\bar{x} := \bar{t}\,) \bullet \langle s,\bar{r}\,\rangle)\end{aligned}$$

t_i. If for some i and j, x_i and x_j happen to be the same, the first occurrence from left to right takes precedence (so the other substitution is simply discarded).

In order to represent upper bounds for reduction sequences, we introduce a sort nat, denoting the natural numbers, with constants 0^{nat}, 1^{nat} and $\_ + \_$ of arity $\mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$, with their usual meaning.

We let r, s and t range over terms of sorts $\mathcal{T}_\rho$; x and y are variables of sorts $\mathcal{V}_\rho$; m and n range over sort nat. We abbreviate $((s \bullet t_1) \bullet \cdots \bullet t_n)$ by $s \bullet \bar{t}$. Type decoration is often suppressed.

Finally, we add binary predicate symbols $\_ =_\rho \_$ for equality on sort $\mathcal{T}$ and SN_σ of arity $\mathcal{T}_\sigma \times \mathsf{nat}$, representing the relation of Definition 1(2).

We can now express the axioms that will be used in the formal proof. We will use the axiom schema **repl** : $s = t \to \varphi(s) \to \varphi(t)$ to replace equals by equals. Furthermore, we use all well typed instances of the following axiom schemata.

1. $\underline{\forall x}.\ \mathrm{SN}_\rho(\mathsf{V}(x), 0)$
2. $\underline{\forall x, \bar{t}, s, m, n}.\ \mathrm{SN}_{\rho\to\sigma}(\mathsf{V}(x) \bullet \bar{t}, m) \to \mathrm{SN}_\rho(s, n) \to \mathrm{SN}_\sigma(\mathsf{V}(x) \bullet \langle \bar{t}, s\rangle, m+n)$
3. $\underline{\forall s, x, m}.\ \mathrm{SN}_\sigma(s \bullet \mathsf{V}(x), m) \to \mathrm{SN}_{\rho\to\sigma}(s, m)$
4. $\underline{\forall r, y, \bar{x}, s, \bar{t}, \bar{r}, m, n}.\ \mathrm{SN}_\iota(r(y, \bar{x} := s, \bar{t}) \bullet \bar{r}, m) \to \mathrm{SN}_\rho(s, n) \to$
   $\mathrm{SN}_\iota(\boldsymbol{\lambda}(y, r)(\bar{x} := \bar{t}) \bullet \langle s, \bar{r}\rangle, m+n+1)$
5. $\underline{\forall \bar{t}}.\ t_i = \mathsf{V}(\ulcorner x_i \urcorner)(\overline{\ulcorner x \urcorner} := \bar{t})$, provided i is the first occurrence of $\ulcorner x_i \urcorner$ in $\overline{\ulcorner x \urcorner}$.
6. $\underline{\forall r, s, \bar{x}, \bar{t}}.\ r(\bar{x} := \bar{t}) \bullet s(\bar{x} := \bar{t}) = (r \bullet s)(\bar{x} := \bar{t})$
7. $\underline{\forall s, \bar{x}}.\ s(\bar{x} := \mathsf{V}(\bar{x})) = s$, where $\mathsf{V}(\bar{x})$ stands for $\mathsf{V}(x_1), \ldots, \mathsf{V}(x_m)$

In the formal proofs, we will refer to these axioms by number (e.g. ax_5). Axioms 1–3 express simple combinatorial facts about SN. The equations 5–7 axiomatize substitution. Axiom 4 is a mix, integrating a basic fact about reduction and an equation for substitution. The reason for this mixture is that we thus avoid variable name clashes. This is the only axiom that needs some elaboration.

In the intended model, $(\lambda x r)[\bar{x} := \bar{t}]$ equals $\lambda x(r[\bar{x} := \bar{t}])$, because we can perform an α-conversion, renaming x. However, we cannot postulate the similar equation

$$\forall x, \bar{x}, \bar{t}, r.\ \boldsymbol{\lambda}(x, r)(\bar{x} := \bar{t}) = \boldsymbol{\lambda}(x, r(\bar{x} := \bar{t}))$$

as an axiom, because we cannot avoid that e.g. t_1 gets instantiated by a term containing the free variable x, such that the same x would occur both bound and free[2]. Now in the proof of Lemma 4 it is shown how the reduction length of $(\lambda y.t)s\bar{r}$ can be estimated from the reduction lengths of s and $t[y := s]\bar{r}$. After substituting $r[\bar{x} := \bar{t}]$ for t, and using the abovementioned equation (thus avoiding that variables in $\bar{t}$ become bound), we get Axiom 4.

5.2 Proof Terms and Extracted Programs

As in the informal proof, we define formulae $\mathrm{SC}_\rho(t)$ by induction on the type ρ. These will occur as abbreviations in the formal derivations.

$$\begin{cases} \mathrm{SC}_\iota(t) := \exists n^{\mathsf{nat}} \mathrm{SN}_\iota(t, n) \\ \mathrm{SC}_{\rho\to\sigma}(t) := \underline{\forall s^{\mathcal{T}_\rho}} \mathrm{SC}_\rho(s) \to \mathrm{SC}_\sigma(t \bullet s) \end{cases}$$

[2] Strictly speaking, [1] erroneously ignores this subtlety.

by induction on ρ. This corresponds to the induction on ρ in the informal proof. The base case uses Axiom 4. Only the first two subscripts will be written in the sequel.

$$\begin{aligned}\Lambda_{\iota,\sigma} = {} & \underline{\lambda r, y, \overline{x}, s, \overline{t}, \overline{r}}\, \lambda u^{\mathrm{SC}_\iota(r(y,\overline{x}:=s,\overline{t}\,)\bullet\overline{r}\,)} \lambda v^{\mathrm{SC}(s)} \\ & \quad \exists^-[u; m; u_0^{\mathrm{SN}(r(y,\overline{x}:=s,\overline{t}\,)\bullet\overline{r},m)}; \\ & \qquad \exists^-[(\Phi_\sigma \underline{s} v); n; v_0^{\mathrm{SN}(s,n)}; \\ & \qquad\quad \exists^+[m+n+1; (ax_4 \underline{ry\overline{x}s\overline{t}\overline{r}mn}u_0 v_0)]]]\end{aligned}$$

$$\begin{aligned}\Lambda_{\rho\to\tau,\sigma} = {} & \underline{\lambda r, y, \overline{x}, s, \overline{t}, \overline{r}}\, \lambda u^{\mathrm{SC}(r(y,\overline{x}:=s,\overline{t}\,)\bullet\overline{r}\,)} \lambda v^{\mathrm{SC}(s)} \\ & \quad \underline{\lambda r'} \lambda w^{\mathrm{SC}_\rho(r')} (\Lambda_{\tau,\sigma} \underline{ry\overline{x}s\overline{t}\overline{r}r'} (u\underline{r'}w)v)\end{aligned}$$

Having these proofs, we can extract their programs, using the definition of **ep**. In this way we get $\mathbf{ep}(\Lambda_{\rho,\sigma}) : \rho \to \sigma \to \rho$,

$$\begin{aligned}\mathbf{ep}(\Lambda_{\iota,\sigma}) &= \lambda x_u \lambda x_v (m+n+1)[n := \mathbf{ep}(\Phi_\sigma)x_v][m := x_u] \\ &= \lambda x_u \lambda x_v (x_u + (\mathbf{ep}(\Phi_\sigma)x_v) + 1) \\ \mathbf{ep}(\Lambda_{\rho\to\tau,\sigma}) &= \lambda x_u \lambda x_v \lambda x_w (\mathbf{ep}(\Lambda_{\tau,\sigma})(x_u x_w)x_v)\end{aligned}$$

Formalizing the Main Lemma. The main lemma (5) states that every term s is strongly computable, even after substituting strongly computable terms for variables. The informal proof of Lemma 5 is with induction on s. Therefore, we can only give a formal proof for each s separately. Given a term s with all free variables among $\overline{x}$, we construct by induction on the term structure a proof

$$\Pi_{s,\overline{x}} : \underline{\forall t_1, \ldots, t_n}.\, \mathrm{SC}(t_1) \to \cdots \to \mathrm{SC}(t_n) \to \mathrm{SC}(\ulcorner s \urcorner(\overline{\ulcorner x \urcorner} := \overline{t}\,)).$$

$$\begin{aligned}&\Pi_{x_i,\overline{x}} := \underline{\lambda \overline{t}} \lambda \overline{u} (\mathbf{repl}\ (ax_5\, \underline{\overline{t}}\,)\ u_i) \\ &\Pi_{rs,\overline{x}} := \underline{\lambda \overline{t}} \lambda \overline{u} (\mathbf{repl}\ (ax_6\, \underline{\ulcorner r \urcorner \ulcorner s \urcorner \overline{\ulcorner x \urcorner} \overline{t}}\,)\ (\Pi_{r,\overline{x}} \underline{\overline{t}} \overline{u}\ \overline{\ulcorner s \urcorner (\overline{\ulcorner x \urcorner} := \overline{t}\,)}\ (\Pi_{s,\overline{x}} \underline{\overline{t}} \overline{u}))) \\ &\Pi_{\lambda xr,\overline{x}} := \underline{\lambda \overline{t}} \lambda \overline{u} \underline{\lambda s} \lambda v^{\mathrm{SC}(s)} (\Lambda_{\rho,\sigma} \underline{r' \ulcorner y \urcorner \overline{\ulcorner x \urcorner} s \overline{t}}\ (\Pi_{r',y,\overline{x}} \underline{s\overline{t}} v \overline{u}) v),\end{aligned}$$

where in the last equation, we assume that $\ulcorner \lambda x r \urcorner = \boldsymbol{\lambda}(\ulcorner y \urcorner, r')$, with $x : \sigma$ and $r : \rho$.

Again we extract the programs from these formal proofs. Because the realizer of **repl** is the identity, we can safely drop it from the extracted program. For terms s^σ with free variables among $\overline{x}$, each $x_i : \tau_i$, we get $\mathbf{ep}(\Pi_{s,\overline{x}}) : \overline{\tau} \to \sigma$,

$$\begin{aligned}\mathbf{ep}(\Pi_{x_i,\overline{x}}) &= \lambda \overline{x}_u x_{u,i} \\ \mathbf{ep}(\Pi_{rs,\overline{x}}) &= \lambda \overline{x}_u (\mathbf{ep}(\Pi_{r,\overline{x}})\overline{x}_u (\mathbf{ep}(\Pi_{s,\overline{x}})\overline{x}_u)) \\ \mathbf{ep}(\Pi_{\lambda xr,\overline{x}}) &= \lambda \overline{x}_u \lambda x_v (\mathbf{ep}(\Lambda_{\rho,\sigma})(\mathbf{ep}(\Pi_{r',y,\overline{x}})x_v \overline{x}_u)x_v),\end{aligned}$$

where again it is assumed that $\ulcorner \lambda x r \urcorner = \boldsymbol{\lambda}(\ulcorner y \urcorner, r')$, $x : \sigma$ and $r : \rho$.

Formalization of the Theorem. Now we are able to give a formal proof of $\exists n \mathrm{SN}(\ulcorner s \urcorner, n)$, for any term s. Extracting the computational content of this proof, we get an upper bound for the length of reduction sequences starting from s. We will define formal proofs $\Omega_s : \exists n \mathrm{SN}(\ulcorner s \urcorner, n)$ for each term s ($\ulcorner s \urcorner$ denotes the representation of s). Let $\overline{x}$ be the sequence of free variables in $s : \sigma$, each $x_i : \tau_i$.

$$\Omega_s := (\Phi_\sigma \underline{\ulcorner s \urcorner}(\mathbf{repl}\ (ax_7 \underline{\ulcorner s \urcorner \overline{\ulcorner x \urcorner}})\ (\Pi_{s,\overline{x}} \underline{\mathsf{V}(\overline{\ulcorner x \urcorner})} \Psi_1 \cdots \Psi_n))),$$

where $\Psi_i := (\Psi_{\tau_i} \underline{\mathsf{V}(\ulcorner x_i \urcorner)} \exists^+ [0; (ax_1 \underline{\ulcorner x_i \urcorner})])$ is a proof of $\mathrm{SC}(\mathsf{V}(\ulcorner x_i \urcorner))$ (Ψ is defined in Section 5.2) and $\overline{\mathsf{V}(\ulcorner x \urcorner)}$ stands for $\mathsf{V}(\ulcorner x_1 \urcorner), \cdots, \mathsf{V}(\ulcorner x_n \urcorner)$. As extracted program, we get $\mathbf{ep}(\Omega_s) : \mathsf{nat}$,

$$\mathbf{ep}(\Omega_s) = \mathbf{ep}(\Phi_\sigma)(\mathbf{ep}(\Pi_{s,\overline{x}})(\mathbf{ep}(\Psi_{\tau_1})0) \cdots (\mathbf{ep}(\Psi_{\tau_n})0))$$

5.3 Comparison with Gandy's Proof

In order to compare the extracted programs from the formalized proofs with the strictly monotonic functionals used by Gandy [2], we recapitulate these programs and introduce a readable notation for them.

$$\begin{aligned}
M_\sigma &: \sigma \to \mathsf{nat} && := \mathbf{ep}(\Phi_\sigma) \\
S_\sigma &: \sigma && := \mathbf{ep}(\Psi_\sigma)0 \\
L_{\rho,\sigma} &: \rho \to \sigma \to \rho && := \mathbf{ep}(\Lambda_{\rho,\sigma}) \\
[\![s^\sigma]\!]_{\overline{x} \mapsto \overline{t}} &: \sigma && := \mathbf{ep}(\Pi_{s,\overline{x}})\overline{t} \\
\mathrm{Upper[t]} &: \mathsf{nat} && := \mathbf{ep}(\Omega_t).
\end{aligned}$$

Function application is written more conventionally as $f(x)$ and some recursive definitions are unfolded. Assuming that $\sigma = \sigma_1 \to \cdots \to \sigma_n \to \mathsf{nat}$, these functionals obey the following equations:

$$\begin{aligned}
M_\sigma(f) &= f(S_{\sigma_1}, \ldots, S_{\sigma_n}) \\
S_\sigma(\overline{x}) &= M_{\sigma_1}(x_1) + \cdots + M_{\sigma_n}(x_n) \\
L_{\sigma,\tau}(f, y, \overline{x}) &= f(\overline{x}) + M_\tau(y) + 1 \\
[\![x_i]\!]_{\overline{x} \mapsto \overline{t}} &= t_i \\
[\![rs]\!]_{\overline{x} \mapsto \overline{t}} &= [\![r]\!]_{\overline{x} \mapsto \overline{t}} ([\![s]\!]_{\overline{x} \mapsto \overline{t}}) \\
[\![\lambda x^\sigma r^\rho]\!]_{\overline{x} \mapsto \overline{t}} (y) &= L_{\rho,\sigma}([\![r]\!]_{x,\overline{x} \mapsto y,\overline{t}}, y) \\
\mathrm{Upper}[t^\tau] &= M_\tau([\![t]\!]_{\overline{x} \mapsto \overline{S}}).
\end{aligned}$$

The Correctness Theorem 10 guarantees that $\mathrm{SN}(\ulcorner t \urcorner, \mathrm{Upper}[t])$ is provable in NH, so Upper[t] puts an upper bound on the length of reduction sequences from t. This expression can be compared with the functionals in the proof of Gandy.

First of all, the ingredients are the same. In [2] a functional (say G) is defined playing the rôle of both S and M (and indeed, $S_{\sigma \to \mathsf{nat}} = M_\sigma$). S is a special strictly monotonic functional and M serves as a measure on functionals. Then Gandy gives a non-standard interpretation t^* of a term t, by assigning the special strict functional to the free variables, and interpreting λ-abstraction by a λI term, so that reductions in the argument will not be forgotten. This corresponds to our $[\![t]\!]_{\overline{x} \mapsto \overline{S}}$, where in the λ-case the argument is remembered by $L_{\rho,\sigma}$ and eventually added to the result. Finally, Gandy shows that in each reduction step the measure of the assigned functionals decreases. So the measure of the non-standard interpretation serves as an upper bound.

Looking into the details, there is one slight difference. The bound Upper$[t]$ is sharper than the upper bound given by Gandy. The reason is that Gandy's special functional (resembling S and M by us) is inefficient. It obeys the equation (with $\sigma \equiv \sigma_1 \to \cdots \to \sigma_n \to \mathsf{nat}$)

$$G_\sigma(x_1, \ldots, x_n) := G_{\sigma_1 \to \mathsf{nat}}(x_1) + 2^0 G_{\sigma_2 \to \mathsf{nat}}(x_2) + \cdots + 2^{n-2} G_{\sigma_n \to \mathsf{nat}}(x_n).$$

Gandy defines G_σ with a $+$ functional on all types and a peculiar induction. By program extraction, we found functionals defined by simultaneous induction, using an extra argument as accumulator (see the definition of $\mathbf{ep}(\Phi)$ and $\mathbf{ep}(\Psi)$), thus avoiding the $+$ functional and the implicit powers of 2.

We conclude this section by stating that program extraction provides a tool to compare the two SN-proofs in the case of simply typed λ-calculus.

6 Application to Gödel's T

Gödel's T extends simply typed λ-calculus with higher-order primitive recursion. The set of base types is extended with a type o of natural numbers. We let p and q range over terms of type o. Constants 0^o and $S^{o \to o}$ are added. For each type σ, we add a constant $R_\sigma : \sigma \to (o \to \sigma \to \sigma) \to o \to \sigma$. The following rules express higher-order primitive recursion:

$$R_\sigma st0 \mapsto s \qquad \text{and} \qquad R_\sigma st(Sp) \mapsto tp(R_\sigma stp) \ .$$

With $\to_{\beta R}$ we denote the compatible closure of the β rule and the two recursion rules. It is a well known fact that $\to_{\beta R}$ is a terminating rewrite relation.

The proof à la Tait of this fact (see e.g. [10, 2.2.31]) extends the case of the β-rule, by proving that the new constants are strongly computable. We will present a version with concrete upper bounds. It turns out to be rather cumbersome to give a concrete number. Some effort has been put in identifying and proving the right "axioms" (Lemma 12–15) from which the decorated proof can be constructed (Lemma 16, 17). The extracted upper bounds are compared with the functionals used by Gandy (Section 6.3).

6.1 Changing the Interpretation of SN(t, n)

Consider the following consequence of $\mathrm{SC}_{o \to o}(r)$ for fixed r. This formula is equivalent to $\underline{\forall p} \forall m.\mathrm{SN}(p, m) \to \exists n \mathrm{SN}(rp, n)$. So we can bound the reduction length of rp uniformly in the upper bound for p. More precisely, if $\mathrm{SN}(p, m)$ then $\mathrm{SN}(rp, [\![r]\!](m))$. A stronger uniformity principle appears in [11, § 2.3.4]).

The uniformity principle does not hold if we substitute $R_o st$ for r: Although $\mathrm{SN}(S^k 0, 0)$ holds for each k, $Rst(S^k 0)$ can perform k reduction steps. So $\mathrm{SC}(Rst)$ cannot hold. This shows that it is impossible to prove $\mathrm{SC}(R)$ with SC as in

Definition 2. Somehow, the numerical value (k) has to be taken into account too.

To proceed, we have to change the interpretation of the predicate $\mathrm{SN}(t,n)$. We have to be a bit careful here, because speaking about *the* numerical value of a term s would mean that we assume the existence of a unique normal form. The following definition avoids this assumption:

Definition 11. 1. Second interpretation of SN: $\mathrm{SN}(t,n)$ holds if and only if for all reduction sequences of the form $t \equiv s_0 \rightarrow_{\beta R} s_1 \rightarrow_{\beta R} \cdots \rightarrow_{\beta R} s_m \equiv S^k(r)$, we have $m+k \leq n$. Note that k can only be non-zero for terms of type o.

2. A finite reduction sequence $s_0 \rightarrow_{\beta R} \cdots \rightarrow_{\beta R} s_n$ is *maximal* if s_n is normal (i.e. there is no term t with $s_n \rightarrow_{\beta R} t$). An infinite reduction sequence is always maximal.

So $\mathrm{SN}(t,n)$ means that for any reduction sequence from t to some s, n is at least the length of this sequence plus the number of leading S-symbols in s. Note that $\mathrm{SN}(t,n)$ already holds, if n bounds the length plus value of all *maximal* reduction sequences from t.

We settle the important question to what extent the proofs of Section 5 remain valid. Because these are formal proofs, with SN just as a predicate symbol, the derivation terms remain correct. These derivation terms contain axioms, the validity of which was shown in the intended model. But we have changed the interpretation of the predicate symbol SN. So what we have to do, is to verify that the axioms of Section 5.1 remain correct in the new interpretation.

The axiom schema **repl**, ax_5, ax_6 and ax_7 are independent of the interpretation of SN. Axioms 1, 2 and 3 remain true, because the terms in their conclusion have no leading S-symbols (note that 1 and 2 have a leading variable; 3 is of arrow type). Axiom 4 is proved by a slight modification of the proof of Lemma 4. The following observation is used: If $(\lambda x.s)t\overline{r} \rightarrow^*_{\beta R} S^\ell(q)$, then at some point we contract the outermost β-redex, say $(\lambda x.s')t'\overline{r'} \rightarrow_\beta s'[x:=t']\overline{r'}$. The latter is also a reduct of $s[x:=t]\overline{r}$, so ℓ is bounded by the upper bound for the numerical value of this term.

6.2 Informal Decorated Proof

To prove SC(0), SC(S) and SC(R), we need some axioms, expressing basic truths about SN. In this section, $\rightarrow$ is written for $\rightarrow_{\beta R}$. For 0 and S we have:

Lemma 12. *1.* $\mathrm{SN}(0^o, 0^{\mathsf{nat}})$

2. For all terms p and numbers m, $\mathrm{SN}_o(p,m)$ *implies* $\mathrm{SN}_o((Sp), m+1)$.

Proof. 0 is normal and has no leading S-symbols. If $(Sp) \rightarrow^n S^k(r)$ for some n, k and r, then $p \rightarrow^n S^{k-1}(r)$. From $\mathrm{SN}(p,m)$ we obtain $k+n \leq m+1$. This holds for every reduction sequence, so $\mathrm{SN}((Sp), m+1)$ holds. □

It is less clear which facts we need for the recursion operator. To prove $\mathrm{SC}(R_\sigma)$ (see Lemma 17), we need to prove $\mathrm{SC}_\sigma(Rstp)$ for strongly computable

s, t and p. If p is strongly computable, then $\mathrm{SN}(p, m)$ holds for some m. With induction on m, we will prove $\forall p(\mathrm{SN}(p,m) \to \mathrm{SC}_\sigma(Rstp))$. We need two axioms to establish the base case and the step case of this induction. For the base case, we need (schematic in the type σ):

Lemma 13.

$$\forall s,t,\overline{r},p,\ell,n.\ \mathrm{SN}_\iota(s\overline{r},\ell) \to \mathrm{SN}_{o\to\sigma\to\sigma}(t,n) \to \mathrm{SN}_o(p,0) \to \mathrm{SN}_\iota(R_\sigma stp\overline{r},\ell{+}n{+}1)$$

Proof. Assume $\mathrm{SN}(s\overline{r},\ell)$, $\mathrm{SN}(t,n)$ and $\mathrm{SN}(p,0)$. The latter assumption tells that p is normal and cannot be a successor. If $p \not\equiv 0$, then reductions in $Rstp\overline{r}$ can only occur inside s, t and $\overline{r}$, and these are bounded by $\ell + n$. If $p \equiv 0$, then a maximal reduction of $Rstp\overline{r}$ will consist of first some steps within s, t and $\overline{r}$ (of respectively a, b and c steps, say) followed by an application of the first recursion rule, and finally d more steps. This gives a reduction of the form:

$$Rst0\overline{r} \to^{a+b+c} Rs't'0\overline{r'} \to s'\overline{r'} \to^{d} S^i(r)$$

We can construct a reduction sequence from $s\overline{r}$ via $s'\overline{r'}$ to $S^i(r)$ of length $a{+}c{+}d$. By the first assumption, $a + c + d + i \leq \ell$, by the second assumption $b \leq n$, so $a + b + c + 1 + d + i \leq \ell + n + 1$. As this upper bound holds for an arbitrary maximal reduction sequence, it holds for all reduction sequences, so we get $\mathrm{SN}(Rstp\overline{r}, \ell + n + 1)$. □

The next lemma is needed for the step case. Note that if $\mathrm{SN}(p, m + 1)$ holds, then p may reduce to either 0 (in at most $m + 1$ steps) or to (Sp') (in at most m steps). This explains the first two hypotheses of the following lemma.

Lemma 14.

$$\begin{aligned}\forall s,t,\overline{r},\ell,m,n.\ &\mathrm{SN}_\iota(s\overline{r},\ell) \to \\ &\Big(\forall q.\mathrm{SN}_o(q,m) \to \mathrm{SN}_\iota(tq(Rstq)\overline{r},n)\Big) \to \\ &\Big(\forall p.\mathrm{SN}_o(p,m+1) \to \mathrm{SN}_\iota(Rstp\overline{r},\ell+m+n+2)\Big)\end{aligned}$$

Proof. Assume $\mathrm{SN}_\iota(s\overline{r},\ell)$, $\forall q.\mathrm{SN}(q,m) \to \mathrm{SN}_\iota(tq(Rstq)\overline{r},n)$ and $\mathrm{SN}(p,m+1)$, for arbitrary $s,t,\overline{r},\ell,m,n$ and p. Consider an arbitrary maximal reduction sequence from $Rstp\overline{r}$. It consists of reduction steps inside s, t, p and $\overline{r}$ (of a, b, c and d steps to the terms s', t', p' and $\overline{r'}$, respectively), possibly followed by an application of a recursion rule, concluded by some more steps. We make a case distinction to the shape of the reduct p' after these steps:

Case A: $p' \equiv 0$ Then the maximal reduction has the following shape:

$$Rstp\overline{r} \to^{a+b+c+d} Rs't'0\overline{r'} \to s'\overline{r'} \to^{e} S^i(r)$$

We can construct a reduction from $s\overline{r}$ to $S^i(r)$ of $a + d + e$ steps, hence, by the first assumption, $a{+}d{+}e{+}i \leq \ell$. From the third assumption, we get $c \leq m{+}1$. To bound b, we can only use the second hypothesis. Note that $\mathrm{SN}(0,0)$ and hence $\mathrm{SN}(0,m)$ holds. The second assumption applied to 0 yields $\mathrm{SN}(t0(Rst0)\overline{r},n)$, so necessarily $b \leq n$. Now the reduction sequence can be bounded, viz. $a+b+c+d+1+e+i \leq \ell+m+n+2$.

Case B: $p' \equiv (Sq)$ Then the maximal reduction has the following shape:

$$Rstp\overline{r} \rightarrow^{a+b+c+d} Rs't'(Sq)\overline{r'} \rightarrow t'q(Rs't'q)\overline{r'} \rightarrow^{e} S^{i}(r)$$

First, $\mathrm{SN}(q,m)$ holds, because if $q \rightarrow^{j} S^{k}(q')$, then $p \rightarrow^{c+j} S^{k+1}(q')$, so $c+j+k+1 \leq m+1$, hence $j+k \leq m$. Next note, that there is a reduction from $tq(Rstq)\overline{r}$ to $S^{i}(r)$ of $a+2b+d+e$ steps. Now the second assumption can be applied, which yields that $a+2b+d+e+i \leq n$. Finally, $c \leq m$. Adding up all information, we get $a+b+c+d+1+e+i \leq m+n+1$.

Case C: If cases A and B do not apply, then p' is normal (because a maximal reduction sequence is considered), and no recursion rule applies. The reduction sequence has length $a+b+c+d$ and the result has no leading S-symbols. Now $c \leq m+1$, $a+d \leq \ell$ and $b \leq n$ can be obtained as in Case A. Clearly $a+b+c+d \leq \ell+m+n+1$.

In all cases, the length of the maximal reduction plus the number of leading S-symbols is bounded by $\ell+m+n+2$, so indeed $\mathrm{SN}(Rstp\overline{r}, \ell+m+n+2)$ holds. □

The nice point is that this lemma is $\exists$-free, so it hides no computational content. Unfortunately, it is not strong enough to enable the induction step. We have $\forall q.\mathrm{SN}(q,m) \rightarrow \mathrm{SC}(Rstq)$ as induction hypothesis, and we may assume $\mathrm{SN}(p, m+1)$. In order to apply Lemma 14, we are obliged to give an n, such that $\forall q.\mathrm{SN}(q,m) \rightarrow \mathrm{SN}(tq(Rstq)\overline{r}, n)$ holds, but using the induction hypothesis we can only find an n for each q separately.

The solution of this problem relies on the fact that the upper bound n above does not really depend on q. In the formalism of Section 4, this is expressed by the $\underline{\forall q}$-quantifier. We change the lemma accordingly:

Lemma 15.

$$\begin{aligned}\underline{\forall s,t,\overline{r},\ell,m}.\ & \mathrm{SN}_{\iota}(s\overline{r},\ell) \rightarrow \\ & \Big(\underline{\forall q}.\mathrm{SN}_{o}(q,m) \rightarrow \exists n \mathrm{SN}_{\iota}(tq(Rstq)\overline{r},n)\Big) \rightarrow \\ & \Big(\underline{\forall p}.\mathrm{SN}_{o}(p,m+1) \rightarrow \exists n \mathrm{SN}_{\iota}(Rstp\overline{r}, \ell+m+n+2)\Big)\end{aligned}$$

The justification of this lemma has to be given in terms of NH, as pointed out in Section 4.3. Lemma 15 contains existential quantifiers, so we have to insert a realizer. Of course we take as realizer $\lambda n.n$. Now it can be verified that

$$\lambda n.n\ \mathbf{mr}\ (\text{Lemma } 15) \equiv (\text{Lemma } 14)\ .$$

Eventually, we can prove that the new constants are strongly computable. The Numeral Lemma is a direct consequence of Lemma 12. The Recursor Lemma uses Lemmas 13, 15 and 3. The SC-formula is an abbreviation introduced in Section 5.2. The proofs below are in MF, so the underlining is important.

Lemma 16 (Numeral Lemma). SC(0) *and* SC(S).

Lemma 17 (Recursor Lemma). *For all σ,* SC(R_σ) *is strongly computable.*

Proof. Note that R_σ has type $\sigma \to (o \to \sigma \to \sigma) \to o \to \sigma$. We assume SC($s$), SC($t$) and SC($p$) for arbitrary terms s, t and p. We have to show $\mathrm{SC}_\sigma(R_\sigma stp)$. From the definition of $\mathrm{SC}_o(p)$ we obtain $\exists m \mathrm{SN}(p, m)$. Now $\forall m \underline{\forall p}.\mathrm{SN}(p,m) \to \mathrm{SC}(Rstp)$ is proved by induction on m, which finishes the proof.

Case 0**:** Let SN(p, 0). Let arbitrary, strongly computable $\overline{r}$ be given. We have to prove $\exists k \mathrm{SN}(Rstp\overline{r}, k)$. From SC($s$) and SC($\overline{r}$) we get SC($s\overline{r}$), hence SN($s\overline{r}, \ell$) for some ℓ (using the definition of SC repeatedly). Lemma 3 and the assumption SC(t) imply SN(t, n) for some n. Now Lemma 13 applies, yielding $\mathrm{SN}(Rstp\overline{r}, \ell + n + 1)$. So we put $k := \ell + n + 1$.

Case $m + 1$**:** Assume $\underline{\forall q}.\mathrm{SN}(q, m) \to \mathrm{SC}(Rstq)$ (IH) and SN(p, $m + 1$). Let arbitrary, strongly computable $\overline{r}$ be given. We have to prove $\exists k \mathrm{SN}(Rstp\overline{r}, k)$. As in Case 0, we obtain SN($s\overline{r}, \ell$) for some ℓ. In order to apply Lemma 15, we additionally have to prove $\underline{\forall q}.\mathrm{SN}(q, m) \to \exists n \mathrm{SN}(tq(Rstq)\overline{r}, n)$.

So assume SN(q, m) for arbitrary q. This implies SC(q) and, by IH, SC($Rstq$). Now by definition of SC(t), we have SC($tq(Rstq)\overline{r}$), i.e. SN($tq(Rstq)\overline{r}$, n) for some n. Now Lemma 15 applies, yielding $\mathrm{SN}(Rstp\overline{r}, \ell + m + n' + 2)$ for some n'. We put $k := \ell + m + n' + 2$. □

6.3 Extracted Programs compared with Gandy's Functionals

The informal proof of the previous section can be formalized in the extension of MF, obtained by adding induction axioms to it. System NH has to be extended with induction axioms accordingly. We also have to add the (simultaneous) primitive recursors, in order to get realizers of the induction axioms (see [10, § 1.6.16, § 3.4.5]). Objects in the extended NH are regarded modulo βR-equality.

We will omit the formal proofs here, due to lack of space[3]. Instead, we directly give the program that can be extracted from the formalized proof. This program will contain the primitive recursor R_σ, because Lemma 17 contains induction to a formula φ with $\tau(\varphi) = \sigma$. Using the notation of Section 5.3, the extracted functionals read:

$$\begin{aligned}
[\![0]\!] &= 0\\
[\![S]\!](m) &= m + 1\\
[\![R_\sigma]\!](x, f, 0, \overline{z}) &= x(\overline{z}) + M_{o\to\sigma\to\sigma}(f) + 1\\
[\![R_\sigma]\!](x, f, m + 1, \overline{z}) &= x(\overline{z}) + m + f(m, [\![R_\sigma]\!](x, f, m), \overline{z}) + 2
\end{aligned}$$

These clauses can be added to the definition of $[\![\_]\!]$ (Section 5.3), which now assigns a functional to each term of Gödel's T. This also extends Upper[_], which now computes the upper bound for reduction lengths of terms in Gödel's T. But, due to the changed interpretation of the SN-predicate, we know even more. In

[3] The formal proofs are in the full version.

fact, Upper[t] puts an upper bound on the length *plus the numerical value* of each reduction sequence. More precisely, if $t \rightarrow^i S^j(t')$ then $i + j \leq$ Upper[t].

Gandy's SN-proof can be extended by giving a strictly monotonic interpretation R^* of R, such that the recursion rules are decreasing. The functional used by Gandy resembles the one above, but gives larger upper bounds. It obeys the following equations:

$$\begin{aligned} R^*(x, f, 0, \overline{z}) &= x(\overline{z}) + G(f) + 1 \\ R^*(x, f, m+1, \overline{z}) &= f(m, R^*(x, f, m), \overline{z}) + R^*(x, f, m, \overline{z}) + 1. \end{aligned}$$

Here G is Gandy's version of the functional M (see Section 5.3). Clearly, the successor step of R^* uses the previous result twice, whereas $[\![R]\!]$ uses it only once. Both are variants of the usual recursor. In the base case, the step function f is remembered by both. This is necessary, because the first recursor rule drops its second argument, while reductions in this argument may not be discarded. In step $m+1$ the two versions are really different; R^* adds the results of the steps $0, \cdots, m$, while $[\![R]\!]$ only adds the result of step 0 and the numerical argument m. The addition of the result of step 0 is necessary to achieve strict monotonicity of $[\![R]\!]$ in its third argument.

7 Conclusion

With two case studies we showed, that modified realizability is a useful tool to reveal the similarity between SN-proofs using strong computability and SN-proofs using strictly monotonic functionals. The extra effort for Gödel's T has paid off, because we found sharper upper bounds than in [2, 7]. Moreover, the new upper bound puts a bound on the sum of the length and numerical value of reduction sequences. This information helps to improve the proof that uses strictly monotonic functionals.

We think that our method can be applied more often. In a typical computability proof SC-predicates are defined with induction on types. It is then proved by induction on terms, that any term satisfies SC. By induction on the types, SN follows. After decorating such a proof with an administration for reduction lengths, the appropriate modified realizability interpretation maps SC-predicates to functionals of the original type and SN-predicates to numbers. The extracted program follows the induction on terms to obtain a non-standard interpretation of the term. This object is mapped to an upper bound by the proof that SC implies SN.

The realizability interpretation follows the type system closely. To deal with Gödel's T, induction was added. In the same way, conjunction and disjunction can be added to deal with products and coproducts (see also [2]). Recently, Loader [5] extended Gandy's proof to System F. As he points out, Girard's SN proof for System F (using reducibility candidates, see e.g. [3]) can be decorated, after which modified realizability yields the same upper bound expressions. Another extension could deal with recursion over infinitely branching trees.

A problem arises with the permutative conversions for existential quantifiers in first order logic. Prawitz [8] gives an SN-proof using strong validity (SV). In [7] an SN-proof is given based on strict functionals. The SV-predicate is defined using a general inductive definition, hence the computational contents of Prawitz' proof is not clear. Consequently, the two SN-proofs cannot be related with our method.

The latter system, and also Gödel's T, can be seen as instances of higher-order term rewrite systems. In [6, 7] a method is given to use strict functionals in termination proofs for such rewrite systems. The connection with computability should help in finding strict functionals for such proofs. One could for example extract functionals from a computability proof for a core system, and then change them by hand to obtain termination of a richer system.

The connection between computability and functionals gives rise to the following questions: Are the functionals extracted from SN-proofs always strictly monotonic? What are right notions of strict monotonicity for higher type systems? Can some "easy" classes of strictly monotonic functionals be identified?

References

1. U. Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote ed., *Proc. of TLCA '93,* Utrecht, volume 664 of *LNCS*, pages 91–106. Springer Verlag, 1993.
2. R.O. Gandy. Proofs of strong normalization. In J.R. Hindley and J.P. Seldin ed., *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, London, 1980.
3. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge tracts in theoretical computer science.* Cambridge University Press, 1989.
4. G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting ed., *Constructivity in Mathematics*, pages 101–128. North-Holland, 1959.
5. R. Loader. Normalisation by translation. http://sable.ox.ac.uk/ loader/, 1995.
6. J.C. van de Pol. Termination proofs for higher-order rewrite systems. In J. Heering et al ed., *Proc. of HOA '93*, volume 816 of *LNCS*, pages 305–325. Springer Verlag, 1994.
7. J.C. van de Pol and H. Schwichtenberg. Strict functionals for termination proofs. In M. Dezani-Ciancaglini and G. Plotkin ed., *Proc. of TLCA'95*, volume 902 of *LNCS*, pages 350–364. Springer Verlag, 1995.
8. D. Prawitz. Ideas and results in proof theory. In J.E. Fenstad ed., *Proc. of the 2nd Scandinavian Logic Symposium*, pages 235–307, Amsterdam, 1971. North-Holland.
9. W.W. Tait. Intensional interpretation of functionals of finite types I. *JSL*, 32:198–212, 1967.
10. A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis.* Number 344 in LNM. Springer Verlag, Berlin, 1973. A 2nd corrected edition appeared as ILLC X-93-05, University of Amsterdam.
11. R. de Vrijer. Exactly estimating functionals and strong normalization. *Proc. of the Koninklijke Nederlandse Akademie van Wetenschappen*, 90(4):479–493, Dec 1987.