Two *Different* Strong Normalization Proofs?

— computability versus functionals of finite type —

Jaco van de Pol [*]

October 5, 1995

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. The first method uses the notion of *strong computability predicates.* This method is attributed to Tait [Tai67], who used convertibility predicates to prove a normal form theorem for various systems. Prawitz [Pra71] and Girard [Gir72] introduced stronger variants, to deal with permutative conversions (arising from natural deduction for first order predicate logic) and the impredicative system F, respectively. For the moment we are interested in simply typed lambda calculus and Gödel's T, a system with higher-order primitive recursion; therefore we can stick to Tait's variant.

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 [Gan80]. De Vrijer [dV87] used a variant to compute the exact length of the longest reduction sequence. Van de Pol [vdP94] adapted the notion of strict monotonicity to the general case of higher-order rewrite systems. Schwichtenberg and

[*]Department of Philosophy, Utrecht University. e-mail: `jaco@phil.ruu.nl`

Van de Pol [vdPS95] applied the adapted notion to simply typed lambda calculus, Gödel's T and natural deduction with permutative conversions for the existential quantifier.

In the literature, these two methods are often put in contrast ([Gan80, § 6.3] and [Gir87, annex 2.C.1]). The proof using functionals seems to be more transparent and economizes on proof theoretical complexity. On the other hand, seeing the two proofs one gets the feeling that "somehow, the same thing is going on". Indeed De Vrijer [dV87, § 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. It turns out that the decoration with numbers is not a great modification of the original proof. After constructing such a proof, we use the paradigm of *program extraction* via the *modified realizability interpretation*, to obtain a functional from this proof in a canonical way. Remarkably, this functional 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, the informal (decorated) proof à la Tait of the strong normalization theorem for simply typed lambda calculus is given. 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. Finally, the same project is carried out for Gödel's T in Section 6, after which a conclusion follows (Section 7).

The idea of using a realizability interpretation to extract functionals from a normalization proof already occurs in [Ber93]. A difference is that he uses the same machinery to compute the *normal form* of a term. The contribution of this paper is that we compute numerical upper bounds for the length of reduction sequences, thus enabling a comparison with Gandy's proof. Furthermore, we also deal with Gödel's T, which is a non-trivial extension. 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 Preliminaries

This paper deals with strong normalization (SN) proofs of β-reduction in the simply typed lambda calculus and of βR-reduction in Gödel's T. In this section we will introduce types and terms and introduce our notational conventions.

The set of *simple types* is based on a certain set of base types. In Gödel's T the base type o of natural numbers is included. There is one type operator, $\rightarrow$. With types we mean simple types (also known as finite types). Metavariables $\iota, \iota_1, \cdots$ range over base types; $\rho, \sigma, \tau, \cdots$ over arbitrary types. The set of *simply typed λ-terms* $(r, s, t, \cdots)$ is based on a certain set of typed variables (with metavariables by $x^\sigma, y^\tau, \cdots$), and has typed application and λ-binding as operators. With terms we mean simply typed λ-terms. We let $p, q, \ldots$ range over terms of type o. We will also admit constants, but these can often be regarded as a subset of the variables. They will not occur after a λ. In Gödel's T, the constants 0, S and R_σ play a special rôle, as they appear in the rules for higher-order primitive recursion.

Definition 2.1. *(Simple types and simply typed λ-terms, constants of Gödel's T)*

1. *Base types are types and whenever σ and τ are types, $\sigma \to \tau$ is also a type. These are the only types.*

2. *Variables (and constants) are terms of the designated type. If s and t are terms of type $\rho \to \sigma$ and ρ, respectively, then (st) is a term of type σ. If x^σ is a variable and s is a term of type τ, then $(\lambda x^\sigma s)$ is a term of type $\sigma \to \tau$. These are the only terms.*

3. *Gödel's T contains a special base type o and constants $0 : o$, $S : o \to o$ and for each type σ, $R_\sigma : \sigma \to (o \to \sigma \to \sigma) \to o \to \sigma$.*

We often omit type decoration and outer brackets. Furthermore, we will use a vector notation for sequences. We will often abbreviate a sequence of terms $t_1, \ldots, t_n$ by $\vec{t}$. In the same way, sequences of variables $\vec{x}$ or types $\vec{\sigma}$ will occur frequently. The length of such sequences is implicitly known, or unimportant. The empty sequence is denoted by ϵ. Finally, the following abbreviations are used:

- $\vec{x}^{\vec{\sigma}} \equiv x_1^{\sigma_1}, \cdots, x_n^{\sigma_n}$ (a sequence of variables with their types)
- $\vec{\sigma} \to \tau \equiv \sigma_1 \to (\sigma_2 \to \cdots \to (\sigma_n \to \tau))$ (a type)
- $\vec{\sigma} \to \vec{\tau} \equiv \vec{\sigma} \to \tau_1, \ldots, \vec{\sigma} \to \tau_n$ (a sequence of types)
- $s\vec{t} \equiv (((st_1)t_2) \cdots t_n)$ (a term)
- $\vec{s}\vec{t} \equiv s_1\vec{t}, \ldots, s_n\vec{t}$ (a sequence of terms)
- $\lambda\vec{x}.t \equiv (\lambda x_1(\lambda x_2 \cdots (\lambda x_n t)))$ (a term)
- $\lambda\vec{x}.\vec{t} \equiv \lambda\vec{x}.t_1, \ldots, \lambda\vec{x}.t_n$ (a sequence of terms)

Example 2.2. *$\lambda x, y, z.x, y, z \equiv \lambda x\lambda y\lambda z.x, \lambda x\lambda y\lambda z.y, \lambda x\lambda y\lambda z.z$, the identity function on sequences of three terms with unspecified type. $\vec{\sigma} \to \epsilon \equiv \epsilon$; in particular, $\epsilon \to \epsilon \equiv \epsilon$). $\epsilon \to \sigma \equiv \sigma$.*

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). Simultaneous substitution is denoted by $s[\vec{x} := \vec{t}]$. Necessary renamings are performed automatically.

Definition 2.3.

1. *The rewrite relation* $s \to_\beta t$ *is defined as the compatible closure of the* β*-rule:*

$$(\lambda x s)t \mapsto s[x := t] \ .$$

2. *The rewrite relation* $s \to_{\beta R} t$ *is the binary relation on terms, defined as the compatible closure of the* β*-rule and the two recursion rules:*

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

3. *Let* $\to$ *be* $\to_\beta$ *or* $\to_{\beta R}$*. A finite reduction sequence* $s_0 \to \cdots \to s_n$ *is* maximal *if* s_n *is normal (i.e. there is no term* t *with* $s_n \to t$*). An infinite reduction sequence is always maximal. We write* $s \to^n t$*, if there is a reduction sequence from* s *to* t *of* n *steps.*

Definition 2.4. *(SN for simply typed lambda calculus)*

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

2. *A term is* strongly normalizing in at most n steps *(*$\mathrm{SN}(t, n)$*) if every reduction sequence is finite, and has length at most* n*.*

In Section 6.1 a similar definition for Gödel's T is presented. We will first prove strong normalization for simply typed lambda calculus. In the next section, we present the proof à la Tait, that every simply typed lambda 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. In Section 6 we prove strong normalization for Gödel's T.

3 Informal Proof à la Tait

The Tait method consists of 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 normalizable, 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).

Definition 3.1. *The set of strongly computable terms is defined inductively as follows:*

(i) $\mathrm{SC}_\iota(t)$ *iff there exists an* n *such that* $\mathrm{SN}(t, n)$*.*

(ii) $\mathrm{SC}_{\sigma\to\tau}(t)$ *iff for all* s *with* $\mathrm{SC}_\sigma(s)$*,* $\mathrm{SC}_\tau(ts)$*.*

Lemma 3.2. (SC Lemma)

(a) For all terms t, if $\mathrm{SC}(t)$ *then there exists an n with* $\mathrm{SN}(t,n)$.

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

In (b), $\vec{t}$ 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 $x\vec{t}$. 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 $t = x\vec{t}$ 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 $x\vec{t}s$ can only take place inside $\vec{t}$ or s, we have $\mathrm{SN}(x\vec{t}s, m+n)$. IH(b) yields that $\mathrm{SC}(x\vec{t}s)$. This proves $\mathrm{SC}(t)$.

$\boxtimes$

Lemma 3.3. (Abstraction Lemma) *For all terms s, t and $\vec{r}$ and variables x, it holds that if* $\mathrm{SC}(s[x:=t]\vec{r})$ *and* $\mathrm{SC}(t)$, *then* $\mathrm{SC}((\lambda x.s)t\vec{r})$.

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

If $\sigma = \iota$, then by definition of SC, we have an n such that $\mathrm{SN}(s[x:=t]\vec{r},n)$. By Lemma 3.2(a) we obtain the existence of m, such that $\mathrm{SN}(t,m)$. We have to show, that there exists a p with $\mathrm{SN}((\lambda x.s)t\vec{r},p)$. We will show that we can put $p := m+n+1$. Consider an arbitrary reduction sequence of $(\lambda x.s)t\vec{r}$. Without loss of generality, we assume that it consists of first a steps in s (yielding s'), b steps in $\vec{r}$ (yielding $\vec{r}\,'$) and c steps in t (yielding t'). After this the outermost redex is contracted, yielding $s'[x:=t']\vec{r}\,'$, and finally d steps occur. Clearly, $c \le m$. Notice that we also have a reduction sequence $s[x:=t]\vec{r} \to^* s'[x:=t']\vec{r}\,'$ of at least $a+b$ steps (we cannot count reductions in t, because we don't know whether x occurs free in s). So surely, $a+b+d \le n$. Summing this up, we have that any reduction sequence from $(\lambda x.s)t\vec{r}$ has length at most $m+n+1$.

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

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

Lemma 3.4. **(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 3.3, 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.) $\boxtimes$

Theorem 3.5. *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 3.2(b), $\mathrm{SC}(x)$ is guaranteed. Now we can apply Lemma 3.4, yielding $\mathrm{SC}(t^\theta)$. Because $t^\theta = t$, we obtain $\mathrm{SC}(t)$. Lemma 3.2(a) yields the existence of an n with $\mathrm{SN}(t, n)$. $\boxtimes$

4 A Variant of Modified Realizability

As mentioned before, we want to extract the computational content from the SN proof of Section 3. We prefer to use a general method for extracting a program from it. This guarantees the objectivity of the claimed connection between the SN proofs using strong computability and those using strictly monotonic functionals. To this end we use *modified realizability*, introduced by Kreisel [Kre59]. In [Tro73, § 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 lambda terms.

Following Berger [Ber93], 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 work in the setting of minimal predicate logic (i.e. negation plays no special rôle), formalized by means of natural deduction. Negation, equality and induction are added as axioms. In the extended theory (minimal arithmetic) the extracted programs will contain primitive recursion operators.

We will also take over a refinement by Berger, which treats specific parts of a proof as *computationally irrelevant*. As a consequence of this refinement the functionals extracted from the SN proof à la Tait are almost the same as those used by Gandy.

4.1 The Modified Realizability Interpretation

A formula can be seen as the specification of a program. The formula $\forall x \exists y.P(x,y)$ specifies a program f of type $o {\rightarrow} o$, such that $\forall x.P(x, f(x))$ holds. In general a sequence of programs is specified. The types of the specified programs can be read off from the formula. An important observation is that a formula without existential quantifiers specifies the empty sequence of programs.

A refinement by Berger enables to express that existentially quantified variables are independent of certain universal variables, by underlining some universal quantifiers. In $\underline{\forall x} \exists y.P(x,y)$ the underlining means that y is not allowed to depend on x. It specifies a number m, such that $\forall x.P(x,m)$ holds. This could of course also be specified by the formula $\exists y \forall x.P(x,y)$, but in specifications of the form $\underline{\forall x}.P(x) \rightarrow \exists y.Q(x,y)$ the underlining cannot be eliminated that easily. This formula specifies a number m, such that $\forall x.P(x) \rightarrow Q(x,m)$ 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 will be expressed in many-sorted first-order logic. We will call this logic **MF**. It is **M**inimal, because negation is not included, and it deals with **F**irst-order objects only. Formally, we have to define a first-order language and the formulae belonging to such a language.

Definition 4.1. *A* many-sorted minimal first-order language *is determined by its* sorts *$(\iota, \iota_1, \cdots)$,* function symbols *with arities $(f^{\iota_1 \times \cdots \times \iota_n \rightarrow \iota}, g, \ldots)$,* variables *of some sort $(x^\iota, y, z \ldots)$ and* predicate symbols *with arities $(P^{\iota_1, \ldots, \iota_n}, Q, \ldots)$.*

The Terms *of* **MF** *$(a, b, c, \ldots)$ are built from the function symbols and the variables in the usual way. Terms have a unique sort. (If for $1 \leq i \leq n$, a_i is a term of sort ι_i, and f is a function symbol of arity $\iota_1 \times \cdots \times \iota_n \rightarrow \iota$, we write $f\vec{a}$ for the term $f(a_1, \ldots, a_n)$ of sort ι).*

The Formulae *of* **MF** *$(\varphi, \psi, \ldots)$ are atomic $(P\vec{a})$, or of the form $\varphi \rightarrow \psi$, $\forall x^\iota \varphi$, $\underline{\forall x^\iota} \varphi$ or $\exists x^\iota \varphi$.*

The language is minimal in the sense that it doesn't contain negation. The intuitive difference between $\forall x \varphi$ and $\underline{\forall x} \varphi$ has been explained above. This difference is more clearly explained by giving proof rules for $\underline{\forall}$ (this is postponed until Section 4.2) and by providing the realizability interpretation for this quantifier.

Let us 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).

Definition 4.2. *(types of a specification)*

$$\begin{array}{rcl}
\tau(P\vec{a}) & := & \epsilon \\
\tau(\varphi \rightarrow \psi) & := & \tau(\varphi) \rightarrow \tau(\psi) \\
\tau(\forall x^\iota \varphi) & := & \iota \rightarrow \tau(\varphi) \\
\tau(\underline{\forall x^\iota} \varphi) & := & \tau(\varphi) \\
\tau(\exists x^\iota \varphi) & := & \iota, \tau(\varphi)
\end{array}$$

Here we use the sequence notation introduced in Section 2. Note that only existential quantifiers give rise to a longer sequence of types. In particular, if φ has no existential quantifiers, then $\tau(\varphi) \equiv \epsilon$. (use that $\vec{\sigma} \to \epsilon \equiv \epsilon$). In $\underline{\forall x^\iota}\varphi$, the program specified by φ may not depend on x, so the "$\iota \to$" is discarded. Furthermore, nested implications give rise to arbitrarily high types.

A sequence of terms $\vec{t}$ is called a *potential realizer* of φ, if it has type $\tau(\varphi)$. Next we want to express that a potential realizer meets its specification. Because the programs have higher types, we need another language to express correctness. We introduce a second logic, called **NH**. It is **N**egative in the sense that it doesn't contain existential quantifiers, and it has **H**igher-order objects. In **NH**, the terms are considered modulo β-conversion. The definition of the language of **NH** refers to the language of **MF**.

Definition 4.3. *A language of negative minimal logic with higher-order objects contains* simple types *$(\rho, \sigma, \tau, \ldots)$ (with the sorts of* **MF** *as base types);* simply typed lambda terms *$(r, s, t, \ldots)$ (with the function symbols of* **MF** *included as constants); and the* predicate symbols *of* **MF**.

The formulae *of* **NH** *are atomic $(P\vec{r})$ or of the form $\varphi \to \psi$ or $\forall x^\rho \varphi$.*

Now we can define what it means that a potential realizer $\vec{t}$ (i.e. a sequence of the right type) meets its specification φ. Traditionally, we say that $\vec{t}$ realizes φ in the modified sense, or $\vec{t}$ **mr** φ for short.

Definition 4.4. *(modified realizability interpretation)*

$$\begin{array}{rcll}
\epsilon\ \mathbf{mr}\ P\vec{a} & := & P\vec{a} & \\
\vec{s}\ \mathbf{mr}\ \varphi \to \psi & := & \forall \vec{x}^{\tau(\varphi)}(\vec{x}\ \mathbf{mr}\ \varphi) \to (\vec{s}\vec{x}\ \mathbf{mr}\ \psi) & \text{(with } \vec{x} \text{ fresh)} \\
\vec{s}\ \mathbf{mr}\ \forall x^\iota \varphi & := & \forall x^\iota(\vec{s}x\ \mathbf{mr}\ \varphi) & (x \notin \mathrm{FV}(\vec{s})) \\
\vec{s}\ \mathbf{mr}\ \underline{\forall x^\iota} \varphi & := & \forall x^\iota(\vec{s}\ \mathbf{mr}\ \varphi) & (x \notin \mathrm{FV}(\vec{s})) \\
r, \vec{s}\ \mathbf{mr}\ \exists x^\iota \varphi(x) & := & \vec{s}\ \mathbf{mr}\ \varphi(r) &
\end{array}$$

Again we use the vector notation of Section 2. In the $\underline{\forall x}$ case, the programs $\vec{s}$ don't get x as input, as intended. But to avoid that x becomes free in φ, we changed Berger's definition by adding $\forall x$. The soundness proof is corrected accordingly. By induction on the formula φ of **MF** one sees immediately that if $\vec{s}$ is of type $\tau(\varphi)$, then $\vec{s}$ **mr** φ 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 is, that from a proof of φ in **MF**, we can extract a program, and a proof in **NH** that this program meets its specification φ.

Proofs are formalized by derivation terms, a linear notation for natural deduction, exploiting the Curry-Howard isomorphism. Advantages are economy of space (in Section 5 large deductions occur) and a more precise treatment of discarded assumptions (resulting in a smooth definition of extracted programs). A drawback is that the proofs are less readable.

The definition of derivation terms should be read as a simultaneous inductive definition. The set of derivation terms is the least set that contains assumption variables and is closed under various syntactic operations (e.g. $\lambda x.\_$, $\exists^-[\_; x; u; \_], \cdots$). By convention, u^φ and v range over assumption variables; x and y range over object variables. We let d, e range over derivations.

The different clauses in the definition of derivation terms mimic the standard introduction and elimination rules of natural deduction. 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. The sets of assumption variables ($\mathrm{FA}(d)$) and of computational relevant variables ($\mathrm{CV}(d)$) are defined simultaneously, because they are needed to express well-formedness of derivations.

Definition 4.5. *(derivation terms, $d, e, \cdots$)*

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

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 $\Phi \vdash_{\mathbf{MF}} \psi$ *if there exists a derivation* d^ψ*, with all assumptions among* Φ*. Likewise for* $\vdash_{\mathbf{NH}}$.

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 general, we can define $\mathbf{ep}(d)$, the sequence of extracted programs from a derivation d. To deal with assumption variables in d, we fix for every assumption variable u^φ a sequence of object variables $\vec{x}_u^{\tau(\varphi)}$. The extracted program is defined with respect to this choice.

Definition 4.6. *(extracted program from* **MF***-derivations)*

$$\begin{array}{rcl}
\mathbf{ep}(u^\varphi) & := & \vec{x}_u^{\tau(\varphi)} \\
\mathbf{ep}(\lambda u^\varphi d^\psi) & := & \lambda\vec{x}_u^{\tau(\varphi)}\mathbf{ep}(d) \\
\mathbf{ep}(d^{\varphi\to\psi}e^\varphi) & := & \mathbf{ep}(d)\mathbf{ep}(e) \\
\mathbf{ep}(\lambda x^\iota d^\varphi) & := & \lambda x^\iota\mathbf{ep}(d) \\
\mathbf{ep}(d^{\forall x^\iota\varphi(x)}a^\iota) & := & \mathbf{ep}(d)a \\
\mathbf{ep}(\underline{\lambda x^\iota} d^\varphi) & := & \mathbf{ep}(d) \\
\mathbf{ep}(d^{\underline{\forall x^\iota}\varphi(x)}\underline{a^\iota}) & := & \mathbf{ep}(d) \\
\mathbf{ep}(\exists^+[a^\iota; d^{\varphi(a)}]) & := & a, \mathbf{ep}(d) \\
\mathbf{ep}(\exists^-[d; y; u^{\varphi(y)}; e^\psi]) & := & \mathbf{ep}(e)[y := s][\vec{x}_u := \vec{t}], \\
 & & \textit{where } s, \vec{t} = \mathbf{ep}(d^{\exists x^\iota\varphi(x)})
\end{array}$$

We need the following facts about the extracted programs:

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

Both can be easily verified by induction on d. Because of the second fact, the correctness formula $\mathbf{ep}(d)$ $\mathbf{mr}$ φ is well defined, and will be a formula of **NH**. The whole theory comes together in the following

Theorem 4.7. Soundness theorem *[Ber93].*
If d^φ is an **MF***-derivation, then an* **NH***-derivation $\mu(d)$ can be given of $\mathbf{ep}(d)$ $\mathbf{mr}$ φ. Moreover, $\mathrm{FA}(\mu(d)) \subseteq \{u^{\vec{x}_u \mathbf{mr} \varphi} | u^\varphi \in \mathrm{FA}(d)\}$.*

Proof: Define:

$$\begin{array}{rcl}
\mu(u^\varphi) & := & u^{\vec{x}_u\mathbf{mr}\varphi} \\
\mu(\lambda u^\varphi d^\psi) & := & \lambda\vec{x}_u^{\tau(\varphi)}\lambda u^{\vec{x}_u\mathbf{mr}\varphi}\mu(d) \\
\mu(de) & := & \mu(d)\mathbf{ep}(d)\mu(e) \\
\mu(\lambda x^\iota d) & := & \lambda x^\iota\mu(d) \\
\mu(da) & := & \mu(d)a \\
\mu(\underline{\lambda x^\iota} d) & := & \lambda x^\iota(\mu(d)) \\
\mu(d\underline{a}) & := & \mu(d)a \\
\mu(\exists^+[a; d^{\varphi(a)}]) & := & \mu(d) \\
\mu(\exists^-[d^{\exists x\varphi(x)}; y; u^{\varphi(y)}; e^\psi]) & := & \mu(e)[y := s][\vec{x}_u := \vec{t}][u := \mu(d)], \\
 & & \text{where } s, \vec{t} = \mathbf{ep}(d)
\end{array}$$

With induction on d one verifies that $\mu(d)$ is a valid proof of the correctness formula, and that its free assumption variables are of the form $u^{\vec{x}_u \mathbf{mr} \varphi}$ for $u^\varphi \in \mathrm{FA}(d)$. We only deal with three cases:

$\to^+$: By the induction hypothesis, $\mu(d) : \mathbf{ep}(d)\ \mathbf{mr}\ \psi$. Then $\mu(\lambda u d)$ proves

$$\begin{array}{lll} & \forall \vec{x}_u(\vec{x}_u\ \mathbf{mr}\ \varphi \to \mathbf{ep}(d)\ \mathbf{mr}\ \psi) & \\ \equiv & \forall \vec{x}_u(\vec{x}_u\ \mathbf{mr}\ \varphi \to (\lambda \vec{x}_u \mathbf{ep}(d))\vec{x}_u\ \mathbf{mr}\ \psi) & \text{(identify } \beta\text{-equal terms)} \\ \equiv & \forall \vec{x}_u(\vec{x}_u\ \mathbf{mr}\ \varphi \to (\mathbf{ep}(\lambda u d))\vec{x}_u\ \mathbf{mr}\ \psi) & \text{(definition } \mathbf{ep}\text{)} \\ \equiv & \mathbf{ep}(\lambda u d)\ \mathbf{mr}\ (\varphi \to \psi) & \text{(definition } \mathbf{mr}\text{).} \end{array}$$

$\underline{\forall}^+$: By induction hypothesis, we have $\mu(d)$ proves $\mathbf{ep}(d)\ \mathbf{mr}\ \varphi$. By the proviso of $\underline{\forall}^+$, $x \notin \mathrm{CV}(d)$, hence (by the first fact about $\mathbf{ep}(d)$) $x \notin \mathrm{FV}(\mathbf{ep}(d))$. Furthermore, x doesn't occur in free assumptions of d, hence not in assumptions of $\mu(d)$, so $\lambda x \mu(d)$ is a correct derivation of $\forall x(\mathbf{ep}(d)\ \mathbf{mr}\ \varphi)$, which is equivalent (because $x \notin \mathbf{ep}(d)$) to $\mathbf{ep}(\underline{\lambda x} d)\ \mathbf{mr}\ \underline{\forall x}\varphi$.

$\exists^-$: Let $s, \vec{t} := \mathbf{ep}(d)$. By induction hypothesis we have proofs $\mu(d)$ of $\mathbf{ep}(d)\ \mathbf{mr}\ \exists x \varphi(x) \equiv \vec{t}\ \mathbf{mr}\ \varphi(s)$ and $\mu(e)$ of $\mathbf{ep}(e)\ \mathbf{mr}\ \psi$, possibly with $u^{\vec{x}_u \mathbf{mr} \varphi(y)}$ among its free assumption variables. As neither y nor $\vec{x}_u$ occur in ψ, $\mu(e)[y := s][\vec{x}_u := \vec{t}]$ (possibly with $u^{\vec{t} \mathbf{mr} \varphi(s)}$ among its free assumption variables) is a proof of $(\mathbf{ep}(e)[y := s][\vec{x}_u := \vec{t}])\ \mathbf{mr}\ \psi$. Hence $\mu(\exists^-[d; y; u; e])$ is a proof of $\mathbf{ep}(\exists^-[d; y; u; e])\ \mathbf{mr}\ \psi$, with the intended free assumptions.

$\boxtimes$

4.3 Realization of Axioms for Equality, Negation, Induction

In this section we will explore the use of axioms. If we use an axiom $\mathbf{ax}^\varphi$ (as open assumption) in a proof d of $\mathbf{MF}$, then the extracted program $\mathbf{ep}(d)$ contains free variables $\vec{x}_{\mathbf{ax}}^{\tau(\varphi)}$ (as holes), and the correctness proof $\mu(d)$ contains free assumption variables $\mathbf{ax}^{\vec{x}_{\mathbf{ax}} \mathbf{mr} \varphi}$ (according to Theorem 4.7).

The goal is to complete the program in a correct way. More specifically, we look for potential realizers $\vec{t}_{\mathbf{ax}}$, such that we can find an $\mathbf{NH}$-derivation $d_{\mathbf{ax}}$ of the correctness statement $\vec{t}_{\mathbf{ax}}\ \mathbf{mr}\ \varphi$. The derivation $d_{\mathbf{ax}}$ may contain acceptable assumptions. If such a proof exists, we call $\vec{t}_{\mathbf{ax}}$ the *realizer* of the axiom. This is a flexible notion, because we have not specified which assumptions are acceptable. The extracted program can be completed by taking $\mathbf{ep}(d)[\vec{x}_{\mathbf{ax}} := \vec{t}_{\mathbf{ax}}]$. The correctness proof can be mended by substituting the subproof $d_{\mathbf{ax}}$ for the free assumption variable $\mathbf{ax}$. It is clear that the justification of postulated principles should be given in terms of $\mathbf{NH}$, because in this logic the correctness proofs live.

We will summarize several situations that can arise by adding realizable axioms to $\mathbf{MF}$. The various possibilities are characterized by the realizers and the assumptions needed in the correctness proofs. Moreover, we will briefly mention their typical use. In the subsequent sections the correctness of these axioms is described in more detail. We distinguish:

1. True $\exists$-free axioms: they have a trivial realizer ϵ and the correctness proof contains the same axioms (up to underlinings). These will typically be non-logical $\exists$-free axioms that are true in the intended model, e.g. symmetry of $=$. The computation is not affected by these and the correctness proof relies on true assumptions. This enables us to reduce the amount of proof to be formalized. We will benefit a lot of it in Section 5.

2. Purely logical axioms with *absolute* realizers, i.e. realizers that have a correctness proof without any assumptions. These will be purely logical axioms, exploiting the realizability interpretation of the $\underline{\forall}$-quantifier. They give some insight in the meaning of the $\underline{\forall}$-quantifier. Some of them will be used in Section 6.4.

3. Axiom schemata with realizers for which the correctness proof contains new instances of the same schema. Typical examples are "ex falso quod libet" and "replacement of equals by equals".

4. Induction axioms. The realizers are operators for simultaneous primitive recursion, the correctness proof is given in an extended framework. Induction is needed to deal with Gödel's T in Section 6.

This is well known theory, apart from the axioms under (2), which explore the special nature of the $\underline{\forall}$-quantifier. Axioms as under (1) are exploited in [Ber93]. Case (3) and (4) can be found in [Tro73].

4.3.1 $\exists$-free Axioms and Harrop Formulae

Consider a $\exists$-free **MF** formula φ. We have $\tau(\varphi) = \epsilon$, so the only potential realizer is the empty sequence. Let φ' be the formula obtained from φ by deleting all underlinings. We have ϵ **mr** $\varphi \equiv \varphi'$. So the program obtained from a proof using φ as an axiom will be correct, whenever φ' is true. In this sense we are allowed to axiomatize new predicates and functions by $\exists$-free axioms.

More generally, we can consider the class of Harrop formulae, i.e. φ with $\tau(\varphi) = \epsilon$. Roughly speaking, these formulae don't have existential quantifiers in their conclusion. They have the empty sequence as a potential realizer. However, we lose the property that ϵ **mr** $\varphi \equiv \varphi'$, as the following example shows:

$$\epsilon \ \mathbf{mr}\ (\forall x^o \exists y^o Q(x,y)) \to P(\vec{t}) \equiv \ \ \forall f^{o\to o}.(\forall x^o Q(x, fx)) \to P(\vec{t})$$

We are tempted to write this last formula as $(\exists f^{o\to o} \forall x^o Q(x, fx)) \to P(\vec{t})$, but this is neither a formula of **NH** nor of **MF**. HA$^\omega$ + AC is needed, to prove (ϵ **mr** φ) $\leftrightarrow \varphi$ for all Harrop formulae φ.

4.3.2 Realizable Axioms

Inspection of the derivation rules for **MF** reveals an asymmetry. Although the introduction rule for $\underline{\forall}$ has a stronger proviso than that of $\forall$, the elimination rules are the same. The result is that there are some principles that are intuitively true, but not provable in **MF**. One of them is $(\underline{\forall x}\exists y\varphi) \to \exists y\underline{\forall x}\varphi$: If for all x there exists a y independent of x, then one such y suffices for all x. So the witness for y on the left hand side should also suffice on the right hand side. This suggests to postulate this formula as an axiom, with the identity as realizer.

Admitting axioms like this one, goes a step further than admitting $\exists$-free formulae as axioms. In the case of $\exists$-free formulae, we can remove all underlinings from the proof, and we obtain a correct proof in a well known logic (i.e. usual minimal first-order predicate logic). If we use the axiom above, this is no longer possible, as it becomes false (even classically) after removing the underlining. On the other hand, for the axioms in this section we can postulate realizers that have a closed correctness proof. In this respect they have a firm base. We will propose the following axiom schemata, where H ranges over Harrop Formulae, i.e. $\tau(H) = \epsilon$. IP stands for *independence of premise* and IU stands for *independence of underlined quantifier*.

$$\begin{array}{lrcll}
\text{IU:} & (\underline{\forall x}\exists y^\iota\varphi) & \to & \exists y^\iota\underline{\forall x}\varphi & \\
\text{IP:} & (H \to \exists x^\iota\varphi) & \to & \exists x^\iota(H \to \varphi) & (x \notin \mathrm{FV}(H)) \\
\text{intro:} & (\forall x H) & \to & \underline{\forall x}H &
\end{array}$$

The first two have associated type $\iota, \tau(\varphi) \to \iota, \tau(\varphi)$ and they are realized by the identity on sequences of this type. The third is realized by the empty sequence. We compute the correctness formulae, which have trivial proofs in **NH**:

$$\begin{array}{lrl}
\text{IU:} & & (\lambda y^\iota, \vec{z}^{\tau(\varphi)}.y, \vec{z})\ \mathbf{mr}\ ((\underline{\forall x}\exists y^\iota\varphi) \to \exists y^\iota\underline{\forall x}\varphi) \\
& \equiv & \forall y, \vec{z}.(y, \vec{z}\ \mathbf{mr}\ \underline{\forall x}\exists y^\iota\varphi) \to y, \vec{z}\ \mathbf{mr}\ \exists y^\iota\underline{\forall x}\varphi \\
& \equiv & \forall y, \vec{z}.(\forall x(\vec{z}\ \mathbf{mr}\ \varphi)) \to \forall x(\vec{z}\ \mathbf{mr}\ \varphi)
\end{array}$$

$$\begin{array}{lrl}
\text{IP:} & & (\lambda x, \vec{z}.x, \vec{z})\ \mathbf{mr}\ ((H \to \exists x^\iota\varphi) \to \exists x^\iota(H \to \varphi)) \\
& \equiv & \forall x, \vec{z}.(\epsilon\ \mathbf{mr}\ H \to \vec{z}\ \mathbf{mr}\ \varphi) \to (\epsilon\ \mathbf{mr}\ H \to \vec{z}\ \mathbf{mr}\ \varphi)
\end{array}$$

$$\begin{array}{lrl}
\text{intro:} & & \epsilon\ \mathbf{mr}\ (\forall x H \to \underline{\forall x}H) \\
& \equiv & (\forall x(\epsilon\ \mathbf{mr}\ H)) \to \forall x(\epsilon\ \mathbf{mr}\ H)
\end{array}$$

This leads to the following

Theorem 4.8. Soundness Theorem 2
If IP+IU+intro $\vdash_{\mathbf{MF}} \varphi$ *then there exists a sequence* $\vec{t}$ *such that* $\vdash_{\mathbf{NH}} \vec{t}\ \mathbf{mr}\ \varphi$*. If* d^φ *is the* **MF***-derivation,* $\vec{t}$ *can be obtained from* $\mathbf{ep}(d)$ *by deleting all free variables introduced by the axioms* **IP**, **IU** *and* **intro**.

We will not address the question whether the inverse of this soundness result also holds. In [Tro73, § 3.4.8] it is proved that HA$^\omega$+IP+AC, axiomatizes modified realizability. However, AC is neither a formula of **MF** nor of **NH**, as it contains both higher-order variables and existential quantifiers, so we cannot use that result here directly.

4.3.3 Axioms for Negation and Equality

We have seen how to realize $\exists$-free axioms. It is not always possible to axiomatize predicates $\exists$-free. We will for example need equality, with the axiom schema $s = t \to \varphi(s) \to \varphi(t)$. Another example is negation, with axioms $\bot \to \varphi$, which can be dealt with in a similar way as equality.

Let $=$ be a binary predicate symbol. The usual axioms of reflexivity, symmetry and transitivity are $\exists$-free and hence harmless. Instances of the replacement schema may contain existential quantifiers. Let **repl** stand for axioms of the form $s = t \to \varphi(s) \to \varphi(t)$. We will provide for realizers such that the correctness formula gets provable in **NH** enriched with the schema **repl**.

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:

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

which can be proved using the **repl** schema on **NH**-formulae. This means that we can use equality axioms within our proofs. Because they are realized by the empty sequence, or by the identity on sequences, we can discard their use when extracting the program. The correctness proofs contain the same axioms schema, which is regarded as valid.

4.3.4 Induction Axioms

It is straightforward to introduce induction in this context. Induction can be postulated by introducing axioms

$$ind_{\varphi} : \varphi(0) \to (\forall n \varphi(n) \to \varphi(Sn)) \to \forall n \varphi(n).$$

In the general case, induction can be realized by simultaneous primitive recursion operators (See [Tro73, § 1.6.16, § 3.4.5]). We will only deal with the special case that $\tau(\varphi) \equiv \sigma$, so the induction formula is realized by exactly one term. We only need this special case, for which the usual recursion operator is a potential realizer. However, for the correctness proof we have to extend **NH** in two directions: We have to add induction axioms to it and we have to consider object terms modulo βR-equality. Let $\tau(\varphi) \equiv \sigma$, then we show that in extended **NH**, R_{σ} **mr** ind_{φ} is provable.

$$R_{\sigma}\ \mathbf{mr}\ ind_{\varphi} \equiv \forall x.(x\ \mathbf{mr}\ \varphi(0)) \to \forall f.(f\ \mathbf{mr}\ \forall n \varphi(n) \to \varphi(Sn)) \to \forall n.(Rxfn)\ \mathbf{mr}\ \varphi(n)\ .$$

So assume x **mr** $\varphi(0)$ and f **mr** $\forall n \varphi(n) \to \varphi(Sn)$. With induction on n we prove $(Rxfn)$ **mr** $\varphi(n)$.

If $n = 0$, we identify $Rxf0$ with x, and the first assumption applies.

If $n = (Sm)$, we may assume that $(Rfxm)$ **mr** $\varphi(m)$ (IH). Our second hypothesis can be rewritten to $\forall n \forall y.(y\ \mathbf{mr}\ \varphi(n)) \to (fny)\ \mathbf{mr}\ \varphi(Sn)$. This can be applied to m and $(Rfxm)$ and after identification of $Rfx(Sm)$ with $fm(Rfxm)$, it follows that $Rxf(Sm)$ **mr** $\varphi(Sm)$.

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 [Tro73, § 2.3.11].

Another incompleteness arises, because some combinatorial results are 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 speak about simply typed lambda terms. We adopt a new sort for each type and function symbols for typed application and abstraction. In this way only well typed terms can be expressed. Because we stick to a first-order language, there will not be explicit bindings nor α-conversion in it. These features only exist in the intended model. Below we give the concrete language with the intended interpretation.

Sorts: nat, denoting the natural numbers. For any type ρ, a sort $\mathcal{V}_\rho$ for variables of type ρ. For each ρ a sort $\mathcal{T}_\rho$, representing the set of lambda terms of type ρ *modulo α-conversion.*

Function symbols: (schematic in types ρ, σ and τ_i)

0^{nat}, 1^{nat} and $\_ + \_$ of arity $\mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$, with their usual meaning;

A variable x^ρ is denoted by a constant $\mathbf{x}$ of sort $\mathcal{V}_\rho$;

$\mathsf{V}_\rho : \mathcal{V}_\rho \to \mathcal{T}_\rho$, to inject variables into terms;

$\_ \bullet_{\rho,\sigma} \_ : \mathcal{T}_{\rho\to\sigma} \times \mathcal{T}_\rho \to \mathcal{T}_\sigma$, denoting typed application;

$\lambda\!\!\lambda_{\rho,\sigma} : \mathcal{V}_\rho \times \mathcal{T}_\sigma \to \mathcal{T}_{\rho\to\sigma}$, denoting typed abstraction;

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 $s(\vec{x} := \vec{t})$ is the simultaneous substitution in s of x_i by 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 second is simply discarded).

Predicate symbols: $\_ =_\rho \_$ of arity $\mathcal{T}_\rho \times \mathcal{T}_\rho$ and SN_σ of arity $\mathcal{T}_\sigma \times \mathsf{nat}$. The first denotes equivalence modulo α-conversion (i.e. identity in the intended model); the second denotes the relation of Definition 2.4(2).

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 $s \bullet \vec{t}$. Type decoration is often suppressed.

Example 5.1. *The term* $(\lambda\!\!\lambda(\mathbf{x},\mathsf{V}(\mathbf{x}))\bullet\mathsf{V}(\mathbf{x}))(\mathbf{x},\mathbf{x}:=\mathsf{V}(\mathbf{y}),\mathsf{V}(\mathbf{z}))$ *is interpreted by the lambda term* $(\lambda xx)y$.

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 $\exists$-free axiom schemata. See for the meaning of the underlinings Section 4.

1. $\underline{\forall x}.\ \mathrm{SN}_\rho(\mathsf{V}(x),0)$
2. $\underline{\forall x,\vec{t},s,m,n}.\ \mathrm{SN}_{\rho\to\sigma}(\mathsf{V}(x)\bullet\vec{t},m)\to\mathrm{SN}_\rho(s,n)\to\mathrm{SN}_\sigma(\mathsf{V}(x)\bullet\langle\vec{t},s\rangle,m+n)$
3. $\underline{\forall s,x,m}.\ \mathrm{SN}_\sigma(s\bullet\mathsf{V}(x),m)\to\mathrm{SN}_{\rho\to\sigma}(s,m)$
4. $\underline{\forall r,y,\vec{x},s,\vec{t},\vec{r},m,n}.\ \mathrm{SN}_\iota(r(y,\vec{x}:=s,\vec{t})\bullet\vec{r},m)$
   $\to\mathrm{SN}_\rho(s,n)$
   $\to\mathrm{SN}_\iota(\lambda\!\!\lambda(y,r)(\vec{x}:=\vec{t})\bullet\langle s,\vec{r}\rangle,m+n+1)$
5. $\underline{\forall\vec{t}}.\ t_i=\mathsf{V}(\mathbf{x}_i)(\vec{\mathbf{x}}:=\vec{t})$, provided that i is the first occurrence of $\mathbf{x}_i$ in $\vec{\mathbf{x}}$.
6. $\underline{\forall r,s,\vec{x},\vec{t}}.\ r(\vec{x}:=\vec{t})\bullet s(\vec{x}:=\vec{t})=(r\bullet s)(\vec{x}:=\vec{t})$
7. $\underline{\forall s,\vec{x}}.\ s(\vec{x}:=\mathsf{V}(\vec{x}))=s$, where $\mathsf{V}(\vec{x})$ 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, $(\lambda xr)[\vec{x}:=\vec{t}]$ equals $\lambda x(r[\vec{x}:=\vec{t}])$, because we can perform an α-conversion, renaming x. However, we cannot postulate the similar equation

$$\forall x,\vec{x},\vec{t},r.\ \lambda\!\!\lambda(x,r)(\vec{x}:=\vec{t})=\lambda\!\!\lambda(x,r(\vec{x}:=\vec{t}))$$

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 will occur both bound and free[1]. Now in the proof of Lemma 3.3 it is shown how the reduction length of $(\lambda y.t)s\vec{r}$ can be estimated from the reduction lengths of s and $t[y:=s]\vec{r}$. After substituting $r[\vec{x}:=\vec{t}]$ for t, and using the abovementioned equation (thus avoiding that variables in $\vec{t}$ become bound), we get Axiom 4.

[1]Strictly speaking, [Ber93] erroneously ignores this subtlety.

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.

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

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.

5.2.1 Formalizing the SC Lemma

We proceed by formalizing Lemma 3.2. We will define proofs

$$\Phi_\rho : \underline{\forall t}.\, \mathrm{SC}_\rho(t) \to \exists n \mathrm{SN}_\rho(t,n) \quad \text{and}$$

$$\Psi_\rho : \underline{\forall x \vec{t}}.\, (\exists m \mathrm{SN}_\rho(\mathsf{V}(x) \bullet \vec{t}, m)) \to \mathrm{SC}_\rho(\mathsf{V}(x) \bullet \vec{t})$$

with simultaneous induction on ρ:

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

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

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 $\mathbf{ep}(\Psi_\rho) : \mathsf{nat} \to \rho$,

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

5.2.2 Formalizing the Abstraction Lemma

We proceed by formalizing Lemma 3.3, which deals with abstractions. Let r have sort $\mathcal{T}_{\vec{\rho}\to\rho}$, and each r_i sort $\mathcal{T}_{\rho_i}$ (so $r \bullet \vec{r}$ 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

$$\Lambda_{\rho,\sigma,\vec{\rho},\vec{\tau}} : \underline{\forall r, y, \vec{x}, s, \vec{t}, \vec{r}}.\, \mathrm{SC}_\rho(r(y, \vec{x} := s, \vec{t}\,) \bullet \vec{r}\,) \to \mathrm{SC}_\sigma(s) \to \mathrm{SC}_\rho(\lambda\!\!\!\lambda(y, r)(\vec{x} := \vec{t}\,) \bullet \langle s, \vec{r}\rangle)$$

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 subscript will be written in the sequel.

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

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

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

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

5.2.3 Formalizing the Main Lemma

The main lemma (3.4) states that every term s is strongly computable, even after substituting strongly computable terms for variables. The informal proof of Lemma 3.4 is with induction on s. Therefore, we can only give a formal proof for each s separately. Given a lambda term s, we write $\overline{s}$ for its representation, consisting of constants $\mathbf{x}, \mathbf{y}, \ldots$ and function symbols V, $\bullet$ and $\lambda\!\!\!\lambda$. This is unique, up to the names of the bound variables. Given a term s with all free variables among $\vec{x}$, we construct by induction on the term structure a proof

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

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

where in the last equation, we assume that $\overline{\lambda x r} = \lambda\!\!\!\lambda(\mathbf{y}, r')$, with $x : \sigma$ and $r : \rho$. $\Lambda_{\rho,\sigma}$ is defined in Section 5.2.2.

Again we extract the programs from these formal proofs. As applications of **repl** are realized by the identity, we can safely drop them from the extracted program. For terms s^σ with free variables among $\vec{x}$, each $x_i : \tau_i$, we get $\mathbf{ep}(\Pi_{s,\vec{x}}) : \vec{\tau} \to \sigma$,

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

where again it is assumed that $\overline{\lambda xr} = \lambda\!\!\lambda(\mathbf{y}, r')$, $x : \sigma$ and $r : \rho$.

5.2.4 Formalization of the Theorem

Now we are able to give a formal proof of $\exists n \mathrm{SN}(\overline{s}, n)$, 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 $\Omega_s : \exists n\mathrm{SN}(\overline{s}, n)$ for each term s ($\overline{s}$ denotes the representation of s). Let $\vec{x}$ be the sequence of free variables in $s : \sigma$, each $x_i : \tau_i$.

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

where $\Psi_i := (\Psi_{\tau_i}\underline{\mathsf{V}(\mathbf{x}_i)}\exists^+[0; (ax_1\underline{\mathbf{x}_i})])$ is a proof of $\mathrm{SC}(\mathsf{V}(\mathbf{x}_i))$ (Ψ is defined in Section 5.2.1) and $\mathsf{V}(\vec{\mathbf{x}})$ stands for $\mathsf{V}(\mathbf{x}_1), \cdots, \mathsf{V}(\mathbf{x}_n)$. As extracted program, we get $\mathbf{ep}(\Omega_s) : \mathsf{nat}$,

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

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 [Gan80], we recapitulate these programs and introduce a readable notation for them.

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

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:

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

The Soundness Theorem 4.8 guarantees that SN($\bar{t}$,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 [Gan80] 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 $[\![t]\!]_{\vec{x}\mapsto\vec{S}}$, 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 are some slight differences. 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 $\sigma \equiv \sigma_1 \to \cdots \to \sigma_n \to \mathsf{nat}$)

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

This function is defined using 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 **ep**(Φ) and **ep**(Ψ)), thus avoiding the + functional and the implicit powers of 2.

We conclude this section by stating that program extraction provides a tool to compare two strong normalization proofs for simply typed lambda calculus, namely the proof à la Tait and Gandy's proof. The functionals involved in both are the same, up to a rather arbitrary choice of some details. In the next section we will treat the case of Gödel's T. It will turn out that again the extracted functionals from the proof à la Tait follow the same scheme as Gandy's functionals. Again the extracted functionals are more efficient than the ones found by Gandy.

6 Application to Gödel's T

In this section, we extend our result to Gödel's T. This system extends simply typed lambda calculus with constants and rewrite rules for higher-order primitive recursion.

The proof à la Tait can be extended by proving that the new constants are strongly computable. We present the version with concrete upper bounds (Section 6.3). Unlike in the simply typed case, it turns out to be rather cumbersome to give a concrete number. Some effort has been put in identifying and proving the right axioms (Sections 6.1, 6.2). The informal proof is formalized and the computational content extracted (Section 6.4). The obtained result is compared with the functionals used by Gandy (Section 6.5).

6.1 Changing the Interpretation of $\mathrm{SN}(t, n)$

Let us 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 version of this uniformity principle appears in [dV87, § 2.3.4]).

Note that the uniformity principle doesn't hold if we substitute $R_o st$ for r: Although $\mathrm{SN}(S^k 0, 0)$ holds for each k, $Rst(S^k0)$ 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 3.1. 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 6.1. *Second interpretation of* SN*:* $\mathrm{SN}(t,n)$ *holds if and only if for all reduction sequences of the form* $t \equiv s_0 \to_{\beta R} s_1 \to_{\beta R} \cdots \to_{\beta R} s_m \equiv S^k(r)$*, we have* $m + k \leq n$*. Note that* k *can only be non-zero for terms of type* o*.*

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 3.3. The following observation is used: If $(\lambda x.s)t\vec{r} \to^*_{\beta R} S^\ell(q)$, then at some point we contract the outermost β-redex, say $(\lambda x.s')t'\vec{r'} \to_\beta s'[x := t']\vec{r'}$. The latter is also a reduct of $s[x := t]\vec{r}$, so ℓ is bounded by the upper bound for the numerical value of this term.

6.2 Basic Facts

To prove $\mathrm{SC}(0)$, $\mathrm{SC}(S)$ and $\mathrm{SC}(R)$, we surely need some new axioms, expressing basic truths about SN. In this section, $\to$ is written for $\to_{\beta R}$. About 0 and S we need:

Lemma 6.2. *(To be used as axiom)*
$\mathrm{SN}(0^o, 0^{\mathsf{nat}})$*; 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 $(Sp) \to^n S^k(r)$ for some n, k and r, then $p \to^n S^{k-1}(r)$. 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 not so clear which facts we need for the recursion operator. To prove $\mathrm{SC}(R_\sigma)$ (See Lemma 6.7), 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 6.3. *(To be used as axiom)*
$\forall s,t,\vec{r},p,\ell,n.\ \mathrm{SN}_\iota(s\vec{r},\ell) \to \mathrm{SN}_{o\to\sigma\to\sigma}(t,n) \to \mathrm{SN}_o(p,0) \to \mathrm{SN}_\iota(R_\sigma stp\vec{r}, \ell+n+1)$

Proof: Assume $\mathrm{SN}(s\vec{r},\ell)$, $\mathrm{SN}(t,n)$ and $\mathrm{SN}(p,0)$. The latter assumption tells that p is normal and can't be a successor. If $p \not\equiv 0$, then reductions in $Rstp\vec{r}$ can only occur inside s, t and $\vec{r}$, and these are bounded by $\ell+n$. If $p \equiv 0$, then a maximal reduction of $Rstp\vec{r}$ will consist of first some steps within s, t and $\vec{r}$ (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:

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

We can construct a reduction sequence from $s\vec{r}$ via $s'\vec{r'}$ 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 $\mathrm{SN}(Rstp\vec{r}, \ell+n+1)$. ⊠

Now we come up with a first proposal for an axiom, enabling the induction step in Lemma 6.7. 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 6.4. *(To be used as axiom)*

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

Proof: Assume $\mathrm{SN}_\iota(s\vec{r},\ell)$, $\forall q.\mathrm{SN}(q,m) \to \mathrm{SN}_\iota(tq(Rstq)\vec{r},n)$ and $\mathrm{SN}(p,m+1)$, for arbitrary $s,t,\vec{r},\ell,m,n$ and p. Consider an arbitrary maximal reduction sequence from $Rstp\vec{r}$. It consists of reduction steps inside s, t, p and $\vec{r}$ (of a, b, c and d steps to the terms s', t', p' and $\vec{r'}$, 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:

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

We can construct a reduction from $s\vec{r}$ 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 $\mathrm{SN}(t0(Rst0)\vec{r}, n)$, 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:

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

First, $\mathrm{SN}(q,m)$ holds, because if $q \to^{j} S^k(q')$, then $p \to^{c+j} S^{k+1}(q')$, so $c+j+k+1 \leq m+1$, hence $j+k \leq m$. Next note, that there is a reduction from $tq(Rstq)\vec{r}$ 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 don't 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 $\mathrm{SN}(Rstp\vec{r}, \ell+m+n+2)$ 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 $\forall q.\mathrm{SN}(q,m) \to \mathrm{SC}(Rstq)$ as induction hypothesis, and we may assume $\mathrm{SN}(p, m+1)$. In order to apply Lemma 6.4, we are obliged to give an n, such that $\forall q.\mathrm{SN}(q,m) \to \mathrm{SN}(tq(Rstq)\vec{r}, n)$ holds, but using the induction hypothesis we can only find an n for a fixed q.

We give two solutions of this problem. Both solutions rely on the fact that the upper bound n above doesn't really depend on q. In the formalism of Section 4, this is expressed by the $\underline{\forall q}$-quantifier.

The first solution uses the axioms IP and IU, enabling us to derive $\exists n \underline{\forall q}.\mathrm{SN}(q,m) \to \mathrm{SN}(tq(Rstq)\vec{r}, n)$ from $\underline{\forall q}.\mathrm{SN}(q,m) \to \exists n \mathrm{SN}(tq(Rstq)\vec{r}, n)$. The advantage is that we use a general method. The disadvantage is that after removing the underlinings in the obtained derivation, the proof is no longer valid.

The other solution changes the axiom, by relaxing the second hypothesis of Lemma 6.4 and to weaken its conclusion consequently. We then get an axiom which can be used in the proof of $\mathrm{SC}(R)$, but it contains existential quantifiers and consequently we have to plug in a realizer by hand. This leads to:

Lemma 6.5. *(To be used as axiom)*

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

Strictly speaking, we don't need a proof of this lemma. In Section 6.4 we show that its underlined version is realizable (by the identity) and the correctness proof relies on Lemma 6.4, so only Lemma 6.4 is crucial. But we motivated this lemma by the wish to obtain a valid proof after removing the underlining, and in that case it is important that the axioms are true.

Proof: Assume $\mathrm{SN}_\iota(s\vec{r},\ell)$, $\forall q.\mathrm{SN}(q,m) \to \exists n \mathrm{SN}_\iota(tq(Rstq)\vec{r},n)$ and $\mathrm{SN}(p,m+1)$. Consider a reduction sequence from $Rstp\vec{r}$ of i steps to a term $S^j(r)$, such that $i+j$ is maximal. For this sequence one of the cases A, B or C in the proof of Lemma 6.4 applies. In all cases we find an appropriate q with $\mathrm{SN}(q,m)$, for which we need the second assumption. This gives an n, for which $\mathrm{SN}_\iota(tq(Rstq)\vec{r},n)$ holds. Now $i+j$ can be bounded by $\ell+m+n+2$, just as in the applicable case of Lemma 6.4. ⊠

6.3 Informal Decorated Proof

In this section we prove that the new constants are strongly computable. The proofs here serve as explanations of the formal proofs in Section 6.4. The Numeral Lemma is a direct consequence of Lemma 6.2. The Recursor Lemma uses Lemmas 6.3 and 6.5, Lemma 3.2 and the definition of SC (Definition 3.1).

Lemma 6.6. (Numeral Lemma) $\mathrm{SC}(0)$ *and* $\mathrm{SC}(S)$.

Lemma 6.7. (Recursor Lemma) *For all σ, $\mathrm{SC}(R_\sigma)$ is strongly computable.*

Proof: Note that R_σ has type $\sigma \to (o \to \sigma \to \sigma) \to o \to \sigma$. We assume $\mathrm{SC}(s)$, $\mathrm{SC}(t)$ and $\mathrm{SC}(p)$ 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 $\exists m \mathrm{SN}(p,m)$. We prove $\forall m \forall p.\mathrm{SN}(p,m) \to \mathrm{SC}(Rstp)$ by induction on m, which finishes the proof.

Case 0**:** Let $\mathrm{SN}(p,0)$. Let arbitrary, strongly computable $\vec{r}$ be given. We have to prove $\exists k \mathrm{SN}(Rstp\vec{r},k)$. From $\mathrm{SC}(s)$ and $\mathrm{SC}(\vec{r})$ we get $\mathrm{SC}(s\vec{r})$, hence $\mathrm{SN}(s\vec{r},\ell)$ for some ℓ (using the definition of SC repeatedly). Lemma 3.2 and the assumption $\mathrm{SC}(t)$ imply $\mathrm{SN}(t,n)$ for some n. Now Lemma 6.3 applies, yielding $\mathrm{SN}(Rstp\vec{r},\ell+n+1)$. So we put $k := \ell+n+1$.

Case $m+1$**:** Assume $\forall q.\mathrm{SN}(q,m) \to \mathrm{SC}(Rstq)$ (IH) and $\mathrm{SN}(p,m+1)$. Let arbitrary, strongly computable $\vec{r}$ be given. We have to prove $\exists k \mathrm{SN}(Rstp\vec{r},k)$. As in Case 0, we obtain $\mathrm{SN}(s\vec{r},\ell)$ for some ℓ. In order to apply Lemma 6.5, we additionally have to prove $\forall q.\mathrm{SN}(q,m) \to \exists n \mathrm{SN}(tq(Rstq)\vec{r},n)$.

So assume $\mathrm{SN}(q,m)$ for arbitrary q. This implies $\mathrm{SC}(q)$ and, by IH, $\mathrm{SC}(Rstq)$. Now by definition of $\mathrm{SC}(t)$, we have $\mathrm{SC}(tq(Rstq)\vec{r})$, i.e. $\mathrm{SN}(tq(Rstq)\vec{r},n)$ for some n. Now Lemma 6.5 applies, yielding $\mathrm{SN}(Rstp\vec{r},\ell+m+n'+2)$ for some n'. We put $k := \ell+m+n'+2$. ⊠

6.4 Formalized Proof

In order to formalize the proof of Section 6.3, we extend the language of Section 5.1 with constants $0^{\mathcal{T}_o}$ and infinitely many R_σ of sort $\mathcal{T}_{\sigma\to(o\to\sigma\to\sigma)\to o\to\sigma}$. Note the difference between $0^{\mathcal{T}_o}$ and 0^{nat}. Only on sort nat we will postulate induction axioms.

6.4.1 List of Additional Axioms

In the formalized proof, we use instances of induction (for formulae with a single realizer) and the axioms below, which are underlined versions of Lemma 6.2–6.5. Axioms 10, 11a and 11b are schematic in σ. To enhance readability, we don't write the infix application symbol $\bullet$, so e.g. $s\vec{r}$ denotes $s \bullet \vec{r}$.

8. $\mathrm{SN}(0^{\mathcal{T}_o}, 0^{\mathsf{nat}})$

9. $\underline{\forall pm}.\ \mathrm{SN}(p,m) \to \mathrm{SN}(Sp, m+1)$

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

11a. $$\begin{aligned}\underline{\forall s,t,\vec{r},\ell,m,n}.\quad & \mathrm{SN}_\iota(s\vec{r},\ell)\to\\ & \Big(\underline{\forall q}.\mathrm{SN}_o(q,m)\to\mathrm{SN}_\iota(tq(Rstq)\vec{r},n)\Big)\to\\ & \Big(\underline{\forall p}.\mathrm{SN}_o(p,m+1)\to\mathrm{SN}_\iota(R_\sigma stp\vec{r},\ell+m+n+2)\Big)\end{aligned}$$

11b. $$\begin{aligned}\underline{\forall s,t,\vec{r},\ell,m}.\quad & \mathrm{SN}_\iota(s\vec{r},\ell)\to\\ & \Big(\underline{\forall q}.\mathrm{SN}_o(q,m)\to\exists n\mathrm{SN}_\iota(tq(Rstq)\vec{r},n)\Big)\to\\ & \Big(\underline{\forall p}.\mathrm{SN}_o(p,m+1)\to\exists n\mathrm{SN}_\iota(R_\sigma stp\vec{r},\ell+m+n+2)\Big)\end{aligned}$$

6.4.2 Realization of Axiom 11

By the underlining, it becomes clear that in Axiom 11b, the existentially quantified n, doesn't depend on q and p. We show that Axiom 11a is equivalent in **NH** to the correctness statement "the identity realizes Axiom 11b":

$$\begin{aligned}
&\lambda n^{\mathsf{nat}} n\ \mathbf{mr}\ \underline{\forall s,t,\vec{r},\ell,m}.\quad && \mathrm{SN}_\iota(s\vec{r},\ell)\to\\
&&& \Big(\underline{\forall q}.\mathrm{SN}(q,m)\to\exists n\mathrm{SN}_\iota(tq(Rstq)\vec{r},n)\Big)\to\\
&&& \Big(\underline{\forall p}.\mathrm{SN}(p,m+1)\to\exists n\mathrm{SN}_\iota(Rstp\vec{r},\ell+m+n+2)\Big)\\
\equiv\quad & \forall s,t,\vec{r},\ell,m.\quad && \mathrm{SN}_\iota(s\vec{r},\ell)\to\\
&&& \forall n.\Big(\ n\ \mathbf{mr}\ (\underline{\forall q}.\mathrm{SN}(q,m)\to\exists n\mathrm{SN}_\iota(tq(Rstq)\vec{r},n))\to\\
&&& \qquad n\ \mathbf{mr}\ (\underline{\forall p}.\mathrm{SN}(p,m+1)\to\exists n\mathrm{SN}_\iota(Rstp\vec{r},\ell+m+n+2))\Big)\\
\equiv\quad & \forall s,t,\vec{r},\ell,m.\quad && \mathrm{SN}_\iota(s\vec{r},\ell)\to\\
&&& \forall n.\Big(\ (\forall q.\mathrm{SN}(q,m)\to\mathrm{SN}_\iota(tq(Rstq)\vec{r},n))\to\\
&&& \qquad (\forall p.\mathrm{SN}(p,m+1)\to\mathrm{SN}_\iota(Rstp\vec{r},\ell+m+n+2))\Big)
\end{aligned}$$

6.4.3 Formalization of the Numeral and Recursor Lemma

We first give the formal proof Σ_0 of the formula $\mathrm{SC}(0)$, which abbreviates $\exists n \mathrm{SN}(0, n)$:

$$\begin{aligned} \Sigma_0 &:= \exists^+[0, ax_8] \\ \mathbf{ep}(\Sigma_0) &\equiv 0 \end{aligned}$$

Now the formal proof Σ_S of the formula $\mathrm{SC}(S)$, which is equal to $\underline{\forall p}.(\exists m \mathrm{SN}(p, m)) \rightarrow \exists n \mathrm{SN}(Sp, n)$ follows:

$$\begin{aligned} \Sigma_S &:= \underline{\lambda p} \lambda u^{\mathrm{SC}(p)}.\exists^-[u; m; u_1^{\mathrm{SN}(p,m)}; \exists^+[m+1; (ax_9 \underline{pm} u_1)]] \\ \mathbf{ep}(\Sigma_S) &\equiv \lambda x_u .\, x_u + 1 \end{aligned}$$

Finally, we define formal proofs Σ_R of $\mathrm{SC}(R_\rho)$, schematic in ρ. As mentioned in Section 6.2, we give two alternatives. Both use Axiom 10 in the base case of the induction. In the induction step, the first uses Axiom 11a, IP and IU and the other uses Axiom 11b. The latter one is closest to the informal proof of Section 6.3. To enhance readability, we will write $\underline{\lambda \vec{r}} \lambda \vec{w}$ instead of the more correct $\underline{\lambda r_1} \lambda w_1 \underline{\lambda r_2} \lambda w_2 \cdots$.

$$\Sigma_R := \underline{\lambda s} \lambda u^{\mathrm{SC}(s)} \underline{\lambda t} \lambda v^{\mathrm{SC}(t)} \underline{\lambda p} \lambda w^{\mathrm{SC}(p)}.\exists^-[w; m; w_1^{\mathrm{SN}(p,m)};\ (ind\ Base\ Step\ m\ \underline{p}\ w_1)],$$

where *ind* is induction w.r.t. $\forall m \underline{\forall p}.\mathrm{SN}(p, m) \rightarrow \mathrm{SC}_\rho(Rstp)$.

$$\begin{aligned} Base := \underline{\lambda p} \lambda a^{\mathrm{SN}(p,0)} \underline{\lambda \vec{r}} \lambda \vec{w}^{\mathrm{SC}(\vec{r})}.&\exists^-[(u \underline{\vec{r}} \vec{w}); \ell; u_1^{\mathrm{SN}(s\vec{r},\ell)}; \\ &\exists^-[(\Phi_{o\rightarrow\rho\rightarrow\rho} \underline{t} v); n; v_1^{\mathrm{SN}(t,n)}; \\ &\exists^+[\ell + n + 1; (ax_{10} \underline{s t \vec{r} p \ell n} u_1 v_1 a)]]] \end{aligned}$$

and

$$\begin{aligned} Step := \lambda m \lambda IH^{\underline{\forall q}.\mathrm{SN}(q,m)\rightarrow \mathrm{SC}(Rstq)} & \\ \underline{\lambda p} \lambda a^{\mathrm{SN}(p,m+1)} \underline{\lambda \vec{r}} \lambda \vec{w}^{\mathrm{SC}(\vec{r})}.&\exists^-[(u \underline{\vec{r}} \vec{w}); \ell; u_1^{\mathrm{SN}(s\vec{r},\ell)}; \\ &\exists^-[(ax_{11b} \underline{s t \vec{r} \ell m} u_1 \\ &\qquad \Big(\underline{\lambda q} \lambda b^{\mathrm{SN}(q,m)}.(v \underline{q} (\exists^+[m, b]) \underline{(Rstq)} (IH \underline{q} b) \underline{\vec{r}} \vec{w})\Big) \\ &\qquad \underline{p} a); \\ &\quad n; c^{\mathrm{SN}(\overline{(R}stp\vec{r}, \ell+m+n+2)}; \\ &\exists^+[\ell + m + n + 2, c]]]. \end{aligned}$$

In the alternative proof, only the induction step differs. We can exchange *Step* for:

$$\begin{aligned} Step' := \lambda m \lambda IH^{\underline{\forall q}.\mathrm{SN}(q,m)\rightarrow \mathrm{SC}(Rstq)} & \\ \underline{\lambda p} \lambda a^{\mathrm{SN}(p,m+1)} \underline{\lambda \vec{r}} \lambda \vec{w}^{\mathrm{SC}(\vec{r})}.&\exists^-[(u \underline{\vec{r}} \vec{w}); \ell; u_1^{\mathrm{SN}(s\vec{r},\ell)}; \\ &\exists^-[(IU\ (IP\ \underline{\lambda q} \lambda b^{\mathrm{SN}(q,m)}.\Big(v \underline{q} (\exists^+[m, b]) \underline{(Rstq)} (IH \underline{q} b) \underline{\vec{r}} \vec{w}\Big))); \\ &\quad n; c^{\underline{\forall q} \mathrm{SN}(q,m)\rightarrow \mathrm{SN}(tq(Rstq)\vec{r},n)}; \\ &\exists^+[\ell + m + n + 2; (ax_{11a} \underline{s t \vec{r} \ell m n} u_1 c \underline{p} a)]]] \end{aligned}$$

The proof uses induction, so the extracted program will use recursion. The structure of the induction formula reveals that $\mathbf{ep}(ind) = R_\rho$. The extracted program of the Recursor Lemma is:

$$\mathbf{ep}(\Sigma_R) \equiv \lambda x_u \lambda x_v \lambda x_w.(R_\rho\ \mathbf{ep}(Base)\ \mathbf{ep}(Step)\ x_w),$$

where

$$\mathbf{ep}(Base) \equiv \lambda \vec{x_w}.(x_u \vec{x_w}) + (\mathbf{ep}(\Phi_{o\to\rho\to\rho})x_v) + 1$$

$$\mathbf{ep}(Step) =_\beta \mathbf{ep}(Step') =_\beta \lambda m \lambda x_{IH} \lambda \vec{x_w}.(x_u \vec{x_w}) + m + (x_v m x_{IH} \vec{x_w}) + 2$$

The extracted program of *Step* contains $\mathbf{ep}(ax_{11b})$, that of *Step'* contains $\mathbf{ep}(IU)$ and $\mathbf{ep}(IP)$. All these axioms are realized by the identity, which we left out.

Remark: In [Tro73, § 2.2.18] König's Lemma (or intuitionistically the Fan Theorem) is used to prove that in the reduction tree of a strongly normalizing term, the maximal value is bounded. To avoid this, one can either prove uniqueness of normal forms, or strengthen SC by stating properties of reduction trees, which is rather cumbersome. In our proof König's Lemma is avoided by having a binary SN-predicate, which gives an upper bound on the numerical value.

6.5 Comparison with Gandy's Functionals

Using the notation of Section 5.3, the extracted functionals read:

$$\begin{array}{rcl}
[\![0]\!] & = & 0 \\
[\![S]\!](m) & = & m+1 \\
[\![R_\rho]\!](x, f, 0, \vec{z}) & = & x(\vec{z}) + M_{o\to\rho\to\rho}(f) + 1 \\
[\![R_\rho]\!](x, f, m+1, \vec{z}) & = & x(\vec{z}) + m + f(m, [\![R_\rho]\!](x, f, m), \vec{z}) + 2
\end{array}$$

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

We extended the SN proof for simply typed lambda calculus à la Tait, by proving SC(R). In Gandy's proof, this corresponds to finding a strictly monotonic interpretation R^* of R, such that the recursion rules are decreasing. The functional used by Gandy resembles the one above, but is less efficient. It obeys the following equations:

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

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, but 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. This addition of the result of step 0 is necessary to achieve strict monotonicity of $[\![R]\!]$ in its third argument.

We conclude by stating that also for Gödel's T, program extraction reveals a similarity between the SN proof à la Tait and the SN proof of Gandy. However, the extracted functional from Tait's proof gives a sharper upper bound than the functional given by Gandy. Moreover, because we changed our interpretation of $\mathrm{SN}(t, n)$ to verify the axioms, we know that this sharper upper bound holds for the sum of the length and the numerical value of each reduction sequence. Both results however, could have been easily obtained when using functionals directly.

7 Conclusion

We used modified realizability to demonstrate a similarity of SN proofs using strong computability and SN proofs using monotonic functionals. Although these proofs follow different patterns, we showed that the arithmetic behind both is the same: they produce almost the same upper bound and they compute it in the same way (with the same expression). For simply typed lambda calculus this result follows smoothly. For Gödel's T some effort was involved in discovering and proving the right axioms, expressing the basic combinatorics. This effort has paid off, because we found sharper upper bounds than in [Gan80, vdPS95]. 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.

This research aims at a convenient method to prove termination of higher-order rewrite systems. There are a number of examples where theory about strictly monotonic functionals has been applied successfully [Gan80, vdP94, vdPS95]. The connection found in this paper helps to find appropriate functionals by inspecting proofs using computability predicates. These functionals can be used to give upper bounds for reduction lengths and also to deal with extensions of the initial system in a flexible way. In [vdPS95], upper bounds for a reduction relation including permutative reductions are given. It seems impossible to extract such functionals from Prawitz's proof, because his notion of strong validity requires a more general form of inductive definitions.

Future research should focus on the automatic detection of strict monotonicity for restricted classes of functionals. The connection with strong computability can help to identify such classes.

References

[Ber93] Ulrich Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote, editors, *Proc. of TLCA '93,* Utrecht, volume 664 of *LNCS*, pages 91–106. Springer Verlag, 1993.

[dV87] Roel de Vrijer. Exactly estimating functionals and strong normalization. *Proc. of the Koninklijke Nederlandse Akademie van Wetenschappen*, 90(4):479–493, Dec 1987.

[Gan80] R.O. Gandy. Proofs of strong normalization. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, London, 1980.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[Gir87] J.-Y. Girard. *Proof theory and Logical Complexity, volume I.* Studies in Proof Theory. Bibliopolis, Napoli, 1987.

[Kre59] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, 1959.

[Pra71] D. Prawitz. Ideas and results in proof theory. In Jens Erik Fenstad, editor, *Proc. of the Second Scandinavian Logic Symposium*, pages 235–307, Amsterdam, 1971. North–Holland.

[Tai67] W.W. Tait. Intensional interpretation of functionals of finite types I. *JSL*, 32:198–212, 1967.

[Tro73] A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis.* Number 344 in LNM. Springer Verlag, Berlin, second, corrected edition, 1973. appeared as ILLC X-93-05, University of Amsterdam.

[vdP94] Jaco van de Pol. Termination proofs for higher-order rewrite systems. In J. Heering et al., editor, *Proc. of HOA '93*, volume 816 of *LNCS*, pages 305–325. Springer Verlag, 1994.

[vdPS95] Jaco van de Pol and Helmut Schwichtenberg. Strict functionals for termination proofs. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proc. of TLCA'95*, volume 902 of *LNCS*, pages 350–364. Springer Verlag, 1995.