Refined Program Extraction from Classical Proofs

Ulrich Berger, Wilfried Buchholz and Helmut Schwichtenberg*

March 24, 2001

1 Introduction

It is well known that it is undecidable in general whether a given program meets its specification. In contrast, it can be checked easily by a machine whether a formal proof is correct, and from a constructive proof one can automatically extract a corresponding program, which by its very construction is correct as well. This – at least in principle – opens a way to produce correct software, e.g. for safety-critical applications. Moreover, programs obtained from proofs are "commented" in a rather extreme sense. Therefore it is easy to maintain them, and also to adapt them to particular situations.

We will concentrate on the question of classical versus constructive proofs. It is known that any classical proof of a specification of the form $\forall x \exists y B$ with B quantifier-free can be transformed into a constructive proof of the same formula. However, when it comes to extraction of a program from a proof obtained in this way, one easily ends up with a mess. Therefore, some refinements of the standard transformation are necessary.

In this paper we develop a refined method of extracting reasonable and sometimes unexpected programs from classical proofs. We also generalize previously known results since B in $\forall x \exists y B$ no longer needs to be quantifier-free, but only has to belong to the strictly larger class of *goal formulas* defined in section 3. Furthermore we allow unproven lemmas D in the proof of $\forall x \exists y B$, where D is a *definite* formula (also defined in section 3).

Other interesting examples of program extraction from classical proofs have been studied by MURTHY [16], COQUAND's group (see e.g. [6]) in a type theoretic context and by KOHLENBACH [13] using a Dialectica-interpretation.

There is also a different line of research aimed at giving an algorithmic interpretation to (specific instances of) the classical double negation rule. It essentially started with GRIFFIN's observation [12] that FELLEISEN's control operator $\mathcal{C}$ [9, 10] can be given the type of the stability scheme $\neg\neg A \to A$. This initiated quite a bit of work aimed at extending the Curry-Howard correspon-

*The hospitality of the Mittag-Leffler Institute in the spring of 2001 is gratefully acknowledged.

dence to classical logic, e.g. by Barbanera and Berardi [1], Constable and Murthy [5], Krivine [14] and Parigot [17].

We now describe in more detail what the paper is about. In section 2 we fix our version of intuitionistic arithmetic for functionals, and recall how classical arithmetic can be seen as a subsystem. Then our argument goes as follows. It is well known that from a derivation of a classical existential formula $\tilde{\exists} yA := \forall y(A \to \bot) \to \bot$ one generally cannot read off an instance. A simple example has been given by Kreisel: Let R be a primitive recursive relation such that $\tilde{\exists} zR(x, z)$ is undecidable. Clearly we have – even logically –

$$\vdash \forall x \tilde{\exists} y \forall z.R(x, z) \to R(x, y).$$

But there is no computable f satisfying

$$\forall x \forall z.R(x, z) \to R(x, f(x)),$$

for then $\tilde{\exists} zR(x, z)$ would be decidable: it would be true if and only if $R(x, f(x))$ holds.[1]

However, it is well known that in case $\tilde{\exists} yG$ with G quantifier-free one *can* read off an instance. Here is a simple idea of how to prove this: replace $\bot$ anywhere in the proof by $\exists^* yG$ (we use $\exists^*$ for the constructive existential quantifier). Then the end formula $\forall y(G \to \bot) \to \bot$ is turned into $\forall y(G \to \exists^* yG) \to \exists^* yG$, and since the premise is trivially provable, we have the claim.

Unfortunately, this simple argument is not quite correct. First, G may contain $\bot$, and hence is changed under the substitution $\bot \mapsto \exists^* yG$. Second, we may have used axioms or lemmata involving $\bot$ (e.g. $\bot \to P$), which need not be derivable after the substitution. But in spite of this, the simple idea can be turned into something useful.

To take care of lemmata we normally want to use in a derivation of $\tilde{\exists} yG$, let us first slightly generalize the situation we are looking at. Let a derivation (in minimal logic) of $\tilde{\exists} yG$ from $\vec{D}$ and axioms

$$\begin{aligned}
\mathsf{Ind}_{n,A}:& \quad A[n := 0] \to \forall n(A \to A[n := n + 1]) \to \forall nA \\
\mathsf{Ind}_{p,A}:& \quad A[p := \mathsf{true}] \to A[p := \mathsf{false}] \to \forall pA \\
\mathsf{ax}_{\mathsf{true}}:& \quad \mathsf{atom}(\mathsf{true}) \\
\mathsf{ax}_{\mathsf{false},A}:& \quad \mathsf{atom}(\mathsf{false}) \to A
\end{aligned}$$

be given. Here atom is a unary predicate symbol taking one argument of the type o of booleans. The intended interpretation of atom is the set $\{\mathsf{true}\}$; hence "$\mathsf{atom}(t)$" means "$t = \mathsf{true}$". Assume the lemmata $\vec{D}$ and the goal formula G are such that

$$\vdash_{\mathsf{int}} \vec{D} \to D_i[\bot := \exists^* yG], \tag{1}$$
$$\vdash_{\mathsf{int}} G[\bot := \exists^* yG] \to \exists^* yG; \tag{2}$$

[1] Notice our slightly unusual formula notation: the scope of a quantifier followed by a dot extends as far as the surrounding parentheses allow. Otherwise we follow the standard convention that quantifiers bind stronger than $\land$, which binds stronger than $\to$.

here $\vdash_{\mathsf{int}}$ means derivability in intuitionistic arithmetic, i.e. with the additional axioms $\mathsf{efq}_A : \bot \to A$. The substitution $\bot \mapsto \exists^* yG$ turns the axioms above (except efq_A) into instances of the same scheme with different formulas, and hence from our given derivation (in minimal logic) of $\vec{D} \to \forall y(G \to \bot) \to \bot$ we obtain

$$\vdash_{\mathsf{int}} \vec{D}[\bot := \exists^* yG] \to \forall y(G[\bot := \exists^* yG] \to \exists^* yG) \to \exists^* yG.$$

Now (1) allows to drop the substitution in $\vec{D}$, and by (2) the second premise is derivable. Hence we obtain as desired

$$\vdash_{\mathsf{int}} \vec{D} \to \exists^* yG.$$

A main contribution of the present paper is the identification of classes of formulas – to be called *definite* and *goal* formulas – such that slight generalizations of (1) and (2) hold. This will be done in section 3.

We will also give (in section 5) an explicit and useful representation of the program term extracted (by the well-known modified realizability interpretation, cf. [18]) from the derivation M of $\vec{D} \to \exists^* yG$ just constructed. The program term has the form $pt_1 \dots t_n s$, where p is extracted from M and $t_1, \dots, t_n, s$ are determined by the formulas $\vec{D}$ and G only.

Since the constructive existential quantifier $\exists^*$ only enters our derivation in the context $\exists^* yG$, it is easiest to replace this formula everywhere by a new propositional symbol X and stipulate that a term r realizes X iff $G[y := r]$. This allows for a short and self-contained exposition – in section 4 – of all we need about modified realizability, including the soundness theorem. In section 5 we then prove our main theorem about program extraction from classical proofs.

The final section 6 then contains some examples of our general machinery. From a classical proof of the existence of the FIBONACCI numbers we extract in 6.1 a short and surprisingly efficient program, where λ-expressions rather than pairs are passed. In 6.2 we treat as a further example a classical proof of the wellfoundedness of $<$ on $\mathbb{N}$. Finally in 6.3 we take up a suggestion of VELDMAN and BEZEM [19] and present a short classical proof of (the general form of) DICKSON's Lemma, as an interesting candidate for further study.

2 Arithmetic for Functionals

The system we consider is essentially (the negative fragment of) HEYTING's intuitionistic arithmetic in finite types as described e.g. in [8]. It is based on GÖDEL's system T and just adds the corresponding logical and arithmetical apparatus to it. Equations are treated on the meta level by identifying terms with the same normal form.

Types are built from ground types ι for the natural numbers and o for the boolean objects (and possibly other ground types) by $\rho \to \sigma$. The *constants* are

$$\mathsf{true}^{\mathsf{o}},\ \mathsf{false}^{\mathsf{o}},\ 0^{\iota},\ \mathrm{S}^{\iota\to\iota},\ \mathcal{R}_{\mathsf{o},\rho},\ \mathcal{R}_{\iota,\rho}.$$

$\mathcal{R}_{\iota,\rho}$ is the *primitive recursion operator* of type $\rho \to (\iota \to \rho \to \rho) \to \iota \to \rho$ and $\mathcal{R}_{\mathsf{o},\rho}$ is the recursion operator for the type o of booleans, i.e. is of type $\rho \to \rho \to \mathsf{o} \to \rho$ and represents definition by cases. *Terms* are

$$x^\rho,\ c^\rho\ (c^\rho \text{ a constant}),\ \lambda x^\rho r,\ rs$$

with the usual typing rules. The *conversions* are those for the simply typed lambda calculus, plus some new ones for the recursion operators. We write $t+1$ for $\mathrm{S}^{\iota\to\iota}t$.

$$\begin{aligned}
\mathcal{R}_{\mathsf{o},\rho} rs\,\mathsf{true} &\mapsto_{\mathcal{R}} r \\
\mathcal{R}_{\mathsf{o},\rho} rs\,\mathsf{false} &\mapsto_{\mathcal{R}} s \\
\mathcal{R}_{\iota,\rho} rs0 &\mapsto_{\mathcal{R}} r \\
\mathcal{R}_{\iota,\rho} rs(t+1) &\mapsto_{\mathcal{R}} st(\mathcal{R}_{\iota,\rho} rst)
\end{aligned}$$

It is well known that for this system of terms every term strongly normalizes, and that the normal form is uniquely determined; hence the relation $r =_{\beta\mathcal{R}} s$ is decidable (by normalizing r and s). By identifying $=_{\beta\mathcal{R}}$-equal terms (i.e. treating equations on the meta level) we can greatly simplify many formal derivations.

Let atom be a unary predicate symbol taking one argument of type o. The intended interpretation of atom is the set $\{\mathsf{true}\}$; hence "$\mathsf{atom}(t)$" means "$t = \mathsf{true}$". We also allow the propositional symbols $\bot$ and X (i.e. 0-ary predicate symbols). So *formulas* are

$$\bot,\ X,\ \mathsf{atom}(t^{\mathsf{o}}),\ A \to B,\ \forall x^\rho A; \quad \text{abbreviation:} \quad \neg A := A \to \bot.$$

As *axioms* we take the induction schemes $\mathsf{Ind}_{n,A}$ and $\mathsf{Ind}_{p,A}$ for the ground types ι and o, and in addition the "truth axiom" $\mathsf{ax}_{\mathsf{true}}$ and two schemes $\mathsf{ax}_{\mathsf{false},A}$ and efq_A for "ex-falso-quodlibet", one for each of the two possibilities $\mathsf{atom}(\mathsf{false})$ and $\bot$ to express falsity (see introduction). Note that every instance $\bot \to A$ of ex-falso-quodlibet is derivable from $\bot \to X$ and $\bot \to \mathsf{atom}(\mathsf{false})$; this will be useful in section 4 (when we define the extracted program $[\![M]\!]$ of a derivation M).

Derivations are within minimal logic. They are written in natural deduction style, i.e. as typed λ-terms via the well-known CURRY-HOWARD correspondence:

$$\begin{aligned}
&u^B \quad (\text{assumptions}), \quad \text{axioms}, \\
&(\lambda u^A M^B)^{A\to B}, \quad (M^{A\to B} N^A)^B, \\
&(\lambda x^\rho M^A)^{\forall x^\rho A}, \quad (M^{\forall x^\rho A} t^\rho)^{A[x^\rho := t^\rho]}
\end{aligned}$$

where in the $\forall$-introduction $\lambda x M^A$, x must not be free in any B with $u^B \in \mathsf{FA}(M)$; here $\mathsf{FA}(M)$ is the set of free assumption variables of M.

Let Z^X denote this system of intuitionistic arithmetic; Z is obtained from Z^X by omitting X. Z_0 (Z_0^X, resp.) is Z (Z^X, resp.) without the axioms efq_A. For every Z_0-derivation M let M^X denote the Z_0^X-derivation resulting from M by substituting X for $\bot$. Write $C^X := C[\bot := X]$. – $\mathcal{L}[X]$ ($\mathcal{L}$, resp.)

denotes the language of Z^X (Z, resp.). We use P for atomic $\mathcal{L}$-formulas and A, B, C, D, G for $\mathcal{L}[X]$-formulas. $\vdash$ denotes derivability in minimal logic.

Note that in our setting derivability in Z^X is essentially the same as in Z_0^X:

Lemma 2.1. *Let* $F := \mathsf{atom}(\mathsf{false})$ *and* $A^F := A[\bot := F]$. *Then*

$$Z^X \vdash A \iff Z_0^X \vdash A^F.$$

Proof. $\Rightarrow$ holds since efq_A^F is $\mathsf{ax}_{\mathsf{false},A^F}$.

$\Leftarrow$. We have $Z^X \vdash \bot \leftrightarrow F$ by efq_F and $\mathsf{ax}_{\mathsf{false},\bot}$. This implies the claim. $\square$

Since our formulas do not contain the constructive existential quantifier $\exists^*$, we can derive stability for all $\mathcal{L}$-formulas. Hence classical arithmetic (in all finite types) is a subsystem of our present system Z:

Lemma 2.2. *(Stability).* $Z \vdash \neg\neg A \to A$ *for every* $\mathcal{L}$*-formula* A.

Proof. Induction on A.

Case $\mathsf{atom}(t)$. We have $Z \vdash \forall p.\neg\neg\mathsf{atom}(p) \to \mathsf{atom}(p)$ by boolean induction, again using $Z \vdash \bot \leftrightarrow F$ and the truth axiom $\mathsf{ax}_{\mathsf{true}}\colon \mathsf{atom}(\mathsf{true})$.

Case $\bot$. Obviously $Z \vdash \neg\neg\bot \to \bot$.

Case $A \to B$. By induction hypothesis for B:

$$\dfrac{u\colon \neg\neg B \to B \qquad \dfrac{\dfrac{v\colon \neg\neg(A \to B) \qquad \dfrac{\dfrac{u_1\colon \neg B \qquad \dfrac{u_2\colon A \to B \qquad w\colon A}{B}}{F}}{\neg(A \to B)}\,{\to^+}u_2}{F}}{\neg\neg B}\,{\to^+}u_1}{B}$$

Case $\forall x A$. Clearly it suffices to show $Z \vdash (\neg\neg A \to A) \to \neg\neg\forall x A \to A$:

$$\dfrac{u\colon \neg\neg A \to A \qquad \dfrac{\dfrac{v\colon \neg\neg\forall x A \qquad \dfrac{\dfrac{u_1\colon \neg A \qquad \dfrac{u_2\colon \forall x A \qquad x}{A}}{F}}{\neg\forall x A}\,{\to^+}u_2}{F}}{\neg\neg A}\,{\to^+}u_1}{A}$$

This concludes the proof. $\square$

Lemma 2.3. *(Cases).* $Z^X \vdash (\neg C \to A) \to (C \to A) \to A$ *for every quantifier-free* $\mathcal{L}$*-formula* C.

Proof. We may assume that $\bot$ does not occur in C, since $Z \vdash \bot \leftrightarrow \mathsf{atom}(\mathsf{false})$. Note that for every such quantifier-free formula C we can easily construct a boolean term t_C such that $Z_0 \vdash \mathsf{atom}(t_C) \leftrightarrow C$. Hence it suffices to derive

$$\forall p.((\mathsf{atom}(p) \to \mathsf{atom}(\mathsf{false})) \to A) \to (\mathsf{atom}(p) \to A) \to A.$$

This is done by induction on p, using the truth axiom $\mathsf{ax}_{\mathsf{true}}\colon \mathsf{atom}(\mathsf{true})$. $\square$

3 Definite and Goal Formulas

A formula is *relevant* if it "ends" with $\bot$. More precisely, relevant formulas are defined inductively by the clauses

- $\bot$ is relevant,
- if C is relevant and B is arbitrary, then $B \to C$ is relevant, and
- if C is relevant, then $\forall x C$ is relevant.

A formula which is not relevant is called *irrelevant*.

We define *goal formulas* G and *definite formulas* D inductively. These notions are related to similar ones common under the same name in the context of extensions of logic programming. Recall that P ranges over atomic $\mathcal{L}$-formulas (including $\bot$).

$$\begin{array}{rcll} G & := & P \mid D \to G & \text{provided } D \text{ irrelevant} \Rightarrow D \text{ quantifier-free} \\ & & \mid \forall x G & \text{provided } G \text{ irrelevant,} \\[1ex] D & := & P \mid G \to D & \text{provided } D \text{ irrelevant} \Rightarrow G \text{ irrelevant} \\ & & \mid \forall x D. & \end{array}$$

Lemma 3.1. *For definite formulas D and goal formulas G we have*

$$Z^X \vdash (\neg D \to X) \to D^X \qquad \textit{for } D \textit{ relevant,} \tag{3}$$

$$Z^X \vdash D \to D^X, \tag{4}$$

$$Z^X \vdash G^X \to G \qquad \textit{for } G \textit{ irrelevant,} \tag{5}$$

$$Z^X \vdash G^X \to (G \to X) \to X. \tag{6}$$

Proof. Simultaneous induction on formulas.

(3). Let D be relevant. *Case* $\bot$. Clearly $((\bot \to \bot) \to X) \to X$ is derivable.

Case $G \to D$.

$$\dfrac{\dfrac{\begin{matrix}\mid\\ (\neg D\to X)\to D^X\end{matrix} \qquad \dfrac{\dfrac{\dfrac{\begin{matrix}\mid\\ G^X\to(G\to X)\to X\end{matrix} \quad G^X}{(G\to X)\to X} \qquad \dfrac{\dfrac{\neg(G\to D)\to X \quad \dfrac{\dfrac{\neg D \quad \dfrac{G\to D \quad G}{D}}{\bot}}{\neg(G\to D)}}{X}}{G\to X}}{X}}{\neg D\to X}}{D^X}}{(\neg(G\to D)\to X)\to G^X\to D^X}$$

Here we have used the induction hypotheses (3) for D and (6) for G.

Case $\forall x D$.

$$\cfrac{\cfrac{\cfrac{\begin{array}{c}\mid\\ (\neg D\to X)\to D^X\end{array}\quad \cfrac{\cfrac{\neg\forall x D\to X \quad \cfrac{\cfrac{\neg D\quad \cfrac{\forall x D}{D}}{\bot}}{\neg\forall x D}}{X}}{\neg D\to X}}{D^X}}{\forall x D^X}}{(\neg\forall x D\to X)\to\forall x D^X}$$

Here we have used the induction hypothesis (3) for D.

(4). *Case* D relevant.

$$\cfrac{\cfrac{\begin{array}{c}\mid\\ (\neg D\to X)\to D^X\end{array}\quad \cfrac{\cfrac{\bot\to X \quad \cfrac{\neg D\quad D}{\bot}}{X}}{\neg D\to X}}{D^X}}{D\to D^X}$$

Here we have used (3) and $\bot\to X$.

Case D irrelevant. *Subcase* P. Then $P^X = P$ and the claim is obvious. *Subcase* $G\to D$. Then D is irrelevant, hence also G is irrelevant.

$$\cfrac{\cfrac{\begin{array}{c}\mid\\ D\to D^X\end{array}\quad \cfrac{G\to D\quad \cfrac{\begin{array}{c}\mid\\ G^X\to G\end{array}\quad G^X}{G}}{D}}{D^X}}{(G\to D)\to G^X\to D^X}$$

Here we have used the induction hypotheses (5) for G and (4) for D.

Subcase $\forall x D$. By the induction hypothesis (4) for D we have $D\to D^X$, which clearly implies $\forall x D\to\forall x D^X$.

(5). Let G be irrelevant. *Case* P. Then $P^X = P$ and the claim is obvious.

Case $D\to G$.

$$\cfrac{\cfrac{\begin{array}{c}\mid\\ G^X\to G\end{array}\quad \cfrac{D^X\to G^X\quad \cfrac{\begin{array}{c}\mid\\ D\to D^X\end{array}\quad D}{D^X}}{G^X}}{G}}{(D^X\to G^X)\to D\to G}$$

Here we have used the induction hypotheses (5) for G and (4) for D.

Case $\forall x G$.

$$\dfrac{\dfrac{\dfrac{\begin{matrix}|\\ G^X \to G\end{matrix} \qquad \dfrac{\forall x G^X}{G^X}}{G}}{\forall x G}}{\forall x G^X \to \forall x G}$$

Here we have used the induction hypothesis (5) for G.

(6). We may assume that G is relevant, for otherwise the claim clearly follows from (5). *Case* $\bot$. Obvious, since $\bot^X = X$.

Case $D \to G$. Our goal is $(D^X \to G^X) \to ((D \to G) \to X) \to X$. Let $\mathcal{D}_1[D^X \to G^X, (D \to G) \to X]$ be

$$\dfrac{\dfrac{\dfrac{\begin{matrix}|\\ G^X \to (G \to X) \to X\end{matrix} \qquad \dfrac{D^X \to G^X \quad D^X}{G^X}}{(G \to X) \to X} \qquad \dfrac{\dfrac{(D \to G) \to X \qquad \dfrac{G}{D \to G}}{X}}{G \to X}}{X}}{D^X \to X}$$

(using the induction hypothesis (6) for G) and $\mathcal{D}_2[(D \to G) \to X]$ be

$$\dfrac{\dfrac{(D \to G) \to X \qquad \dfrac{\dfrac{\dfrac{\neg D \qquad D}{\bot}}{G}}{D \to G}}{X}}{\neg D \to X}$$

Note that the passage from $\bot$ to G can be done by means of introduction rules, since G is relevant.

Subcase D relevant.

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1[D^X \to G^X, (D \to G) \to X]\\ |\\ D^X \to X\end{matrix} \qquad \dfrac{\begin{matrix}|\\ (\neg D \to X) \to D^X\end{matrix} \qquad \begin{matrix}\mathcal{D}_2[(D \to G) \to X]\\ |\\ \neg D \to X\end{matrix}}{D^X}}{X}}{(D^X \to G^X) \to ((D \to G) \to X) \to X}$$

Here we have used the induction hypothesis (3) for D.

Subcase D irrelevant. Then D is quantifier-free. We use case distinction on D from lemma 2.3, in the form $(D \to X) \to (\neg D \to X) \to X$. So it suffices to derive from $D^X \to G^X$ and $(D \to G) \to X$ both premises; recall that our goal was $(D^X \to G^X) \to ((D \to G) \to X) \to X$. The negative case is provided by

$\mathcal{D}_2[(D \to G) \to X]$, and the positive case by

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1[D^X \to G^X, (D \to G) \to X] \\ \mid \\ D^X \to X\end{matrix} \qquad \dfrac{\begin{matrix} \mid \\ D \to D^X\end{matrix} \qquad D}{D^X}}{X}}{D \to X}$$

Here we have used the induction hypothesis (4) for D. ∎

Lemma 3.2. *For goal formulas $\vec{G} = G_1, \dots, G_n$ we have*

$$Z^X \vdash (\vec{G} \to X) \to \vec{G}^X \to X.$$

Proof. By lemma 3.1(6) we have

$$Z^X \vdash G_i^X \to (G_i \to X) \to X$$

for all $i = 1, \dots, n$. Now the assertion follows by minimal logic: Assume $\vec{G} \to X$ and $\vec{G}^X$; we must show X.

Because of $G_1^X \to (G_1 \to X) \to X$ it suffices to prove $G_1 \to X$. Assume G_1.

Because of $G_2^X \to (G_2 \to X) \to X$ it suffices to prove $G_2 \to X$. Assume G_2.

Repeating this pattern, we finally have assumptions $G_1, \dots, G_n$ available, and obtain X from $\vec{G} \to X$. ∎

Theorem 3.3. *Assume that for definite formulas $\vec{D}$ and goal formulas $\vec{G}$ we have*

$$Z_0 \vdash \vec{D} \to \forall \vec{y}(\vec{G} \to \bot) \to \bot.$$

Then we also have

$$Z^X \vdash \vec{D} \to \forall \vec{y}(\vec{G} \to X) \to X.$$

In particular, substitution of the formula

$$\exists^* \vec{y}.\vec{G} := \exists^* \vec{y}.G_1 \wedge \ldots \wedge G_n$$

for X yields

$$Z \vdash \vec{D} \to \exists^* \vec{y}.\vec{G}.$$

Proof. Substitution of X for $\bot$ in the given derivation yields

$$Z_0^X \vdash \vec{D}^X \to \forall \vec{y}(\vec{G}^X \to X) \to X.$$

Now by lemma 3.1(4) we can drop X in $\vec{D}^X$, and by lemma 3.2 also in $\vec{G}^X$.

The second assertion follows from the first one since $\forall \vec{y}.\vec{G} \to \exists^* \vec{y}.\vec{G}$ clearly is derivable. ∎

The theorem can be viewed in the standard way to yield a method for program extraction from classical proofs. However, in section 5 we give a finer analysis of the extracted programs, and an explanation of the role of definite and goal formulas.

Example Let us check the mechanism of working with definite and goal formulas for Kreisel's "non-example" given in the introduction. There we gave a trivial proof in classical logic of a $\forall\exists$-formula that cannot be realized by a computable function, and we better make sure that our general result also does not provide such a function. The example amounts to a proof in minimal logic of

$$\forall z(\neg\neg R(x,z) \to R(x,z)) \to \forall y\big((R(x,y) \to \forall z R(x,z)) \to \bot\big) \to \bot.$$

Here $R(x,y) \to \forall z R(x,z)$ is a goal formula, but the premise $\forall z.\neg\neg R(x,z) \to R(x,z)$ is *not* definite. Replacing R by $\neg S$ (to get rid of the stability assumption) does not help, for then $\neg S(x,y) \to \forall z \neg S(x,z)$ is *not* a goal formula. A third possibility would be to use the fact that R is primitive recursive and write $\mathsf{atom}(rxy)$ instead of $R(x,y)$. However, then $\forall y\big((\mathsf{atom}(rxy) \to \forall z\, \mathsf{atom}(rxz)) \to \bot\big) \to \bot$ could only be proved in Z, *not* in Z_0 as required in theorem 3.3.

How to obtain definite and goal formulas

To apply these results we have to know that our assumptions are definite formulas and our goal is given by goal formulas. For quantifier-free formulas this clearly can always be achieved by inserting double negations in front of every atom (cf. the definitions of definite and goal formulas). This corresponds to the original (unrefined) so-called A-translation of Friedman [11] (or Leivant [15]). However, in order to obtain reasonable programs which do not unneccessarily use higher types or case analysis we want to insert double negations only at as few places as possible.

We describe a more economical general way to obtain definite and goal formulas, following [2, 3]. It consists in singling out some predicate symbols as being "critical", and then double negating only the atoms formed with critical predicate symbols; call these *critical* atoms.

Assume we have a proof in minimal arithmetic Z of

$$\forall \vec{x}_1 C_1 \to \cdots \to \forall \vec{x}_n C_n \to \forall \vec{y}(\vec{B} \to \bot) \to \bot$$

with $\vec{C}, \vec{B}$ quantifier-free (among the premises $\forall \vec{x}_i C_i$ we may have efq-axioms for quantifier-free formulas, hence in fact the situation described applies to intuitionistic logic). Let

$$L := \{\, C_1, \ldots, C_n, \vec{B} \to \bot \,\}$$

The set of *L-critical* predicate symbols is defined to be the smallest set satisfying

(i) $\bot$ is critical.

(ii) If $(\vec{C}_1 \to R_1(\vec{s}_1)) \to \cdots \to (\vec{C}_m \to R_m(\vec{s}_m)) \to R(\vec{s})$ is a positive subformula of L, and if some R_i is L-critical, then R is L-critical.

Now if we double negate every L-critical atom different from $\bot$ we clearly obtain definite assumptions $\vec{C}'$ and goal formulas $\vec{B}'$. Furthermore the proof term of the given derivation can easily be transformed into a correct derivation of the translated formula from the translated assumptions (by inserting the obvious proofs of the translated axioms).

However, in particular cases we might be able to obtain definite and goal formulas with still fewer double negations: it may not be necessary to double negate *every* critical atom.

Of course this method will be really useful only if besides atom and $\bot$ there are other predicate symbols available. Our results could be easily adapted to a language with free predicate symbols.

4 Program Extraction

We assign to every formula A an object $\tau(A)$ (a type or the symbol $*$). $\tau(A)$ is intended to be the type of the program to be extracted from a proof of A, assuming that a proof of X carries computational content of some given type ν.

$$\begin{aligned}
\tau(X) &:= \nu \\
\tau(P) &:= * \quad \text{(in particular } \tau(\bot) = *) \\
\tau(\forall x^\rho A) &:= \begin{cases} * & \text{if } \tau(A) = * \\ \rho \to \tau(A) & \text{otherwise} \end{cases} \\
\tau(A \to B) &:= \begin{cases} \tau(B) & \text{if } \tau(A) = * \\ * & \text{if } \tau(B) = * \\ \tau(A) \to \tau(B) & \text{otherwise} \end{cases}
\end{aligned}$$

We now define, for a given derivation M of a formula A with $\tau(A) \neq *$, its *extracted program* $[\![M]\!]$ of type $\tau(A)$.

$$\begin{aligned}
[\![u^A]\!] &:= u^{\tau(A)} \\
[\![\lambda u^A M]\!] &:= \begin{cases} [\![M]\!] & \text{if } \tau(A) = * \\ \lambda u^{\tau(A)}[\![M]\!] & \text{otherwise} \end{cases} \\
[\![M^{A\to B} N]\!] &:= \begin{cases} [\![M]\!] & \text{if } \tau(A) = * \\ [\![M]\!][\![N]\!] & \text{otherwise} \end{cases} \\
[\![\lambda x^\rho M]\!] &:= \lambda x^\rho [\![M]\!] \\
[\![Mt]\!] &:= [\![M]\!]t
\end{aligned}$$

We also need extracted programs for the axioms.

$$\begin{aligned}
[\![\mathsf{Ind}_{p,A}]\!] &:= \mathcal{R}_{\mathsf{o},\rho} \colon \rho \to \rho \to \mathsf{o} \to \rho & \text{with } \rho := \tau(A) \neq *, \\
[\![\mathsf{Ind}_{n,A}]\!] &:= \mathcal{R}_{\iota,\rho} \colon \rho \to (\iota \to \rho \to \rho) \to \iota \to \rho & \text{with } \rho := \tau(A) \neq *, \\
[\![\mathsf{efq}_X]\!] &:= \mathsf{dummy}^\nu
\end{aligned}$$

where dummy^ν is an arbitrary closed term of type ν. For derivations M of A with $\tau(A) = *$ we define $[\![M]\!] := \varepsilon$ (ε some new symbol). This applies in particular if A is an $\mathcal{L}$-formula.

Finally we define *modified realizability* for formulas in $\mathcal{L}[X]$. For the propositional symbol X we need a comprehension term $\mathcal{A} := \lambda y^\nu A_0$ with an $\mathcal{L}$-formula A_0; write $\mathcal{A}(r)$ for $A_0[y^\nu := r]$. More precisely, we define formulas $r\ \mathbf{mr}_{\mathcal{A}}\ A$, where r is either a term of type $\tau(A)$ if the latter is a type, or the symbol ε if $\tau(A) = *$.

$$\begin{aligned}
r\ \mathbf{mr}_{\mathcal{A}}\ X &= \mathcal{A}(r) \\
r\ \mathbf{mr}_{\mathcal{A}}\ P &= P \\
r\ \mathbf{mr}_{\mathcal{A}}\ \forall x A &= \begin{cases} \forall x.\varepsilon\ \mathbf{mr}_{\mathcal{A}}\ A & \text{if } \tau(A) = * \\ \forall x.rx\ \mathbf{mr}_{\mathcal{A}}\ A & \text{otherwise} \end{cases} \\
r\ \mathbf{mr}_{\mathcal{A}}\ (A \to B) &= \begin{cases} \varepsilon\ \mathbf{mr}_{\mathcal{A}}\ A \ \to\ r\ \mathbf{mr}_{\mathcal{A}}\ B & \text{if } \tau(A) = * \\ \forall x.x\ \mathbf{mr}_{\mathcal{A}}\ A \ \to\ \varepsilon\ \mathbf{mr}_{\mathcal{A}}\ B & \text{if } \tau(A) \neq * = \tau(B) \\ \forall x.x\ \mathbf{mr}_{\mathcal{A}}\ A \ \to\ rx\ \mathbf{mr}_{\mathcal{A}}\ B & \text{otherwise} \end{cases}
\end{aligned}$$

Note that for $\mathcal{L}$-formulas A we have $\tau(A) = *$ and $\varepsilon\ \mathbf{mr}_{\mathcal{A}}\ A = A$. For the formulation of the soundness theorem it will be useful to let $u^{\tau(A)} := \varepsilon$ if u^A is an assumption variable with $\tau(A) = *$.

Theorem 4.1. *(Soundness). Assume that M is a Z^X-derivation of B. Then there is a Z-derivation of $[\![M]\!]\ \mathbf{mr}_{\mathcal{A}}\ B$ from the assumptions $\{\, u^{\tau(C)}\ \mathbf{mr}_{\mathcal{A}}\ C \mid u^C \in \mathsf{FA}(M)\,\}$.*

Proof. Induction on M. *Case* $\mathsf{Ind}_{n,A}$. Take $\mathcal{R}_{\iota,\rho}$. *Case* $\mathsf{Ind}_{p,A}$. Take $\mathcal{R}_{\mathbf{o},\rho}$. *Case* $\mathsf{efq}_A \colon \bot \to A$. Then

$$[\![\mathsf{efq}_A]\!]\ \mathbf{mr}_{\mathcal{A}}\ (\bot \to A) = \bot \to [\![\mathsf{efq}_A]\!]\ \mathbf{mr}_{\mathcal{A}}\ A,$$

which is an instance of the same axiom scheme. The inductive steps are straightforward. $\square$

5 Computational Content of Classical Proofs

For a smooth formulation of the following theorem when writing an application ts where s is of type $*$, we mean simply t. Similarly abstractions of the form $\lambda w^* t$ stand for t.

Theorem 5.1. *Let $\vec{D} = D_1, \ldots, D_n$ and $\vec{G} = G_1, \ldots, G_m$ be arbitrary $\mathcal{L}$-formulas. Assume that we have terms $t_1, \ldots, t_n, s_1, \ldots, s_m, r$ such that*

$$Z \vdash \vec{D} \to t_j\ \mathbf{mr}_{\mathcal{A}}\ D_j^X \qquad \textit{for } 1 \le j \le n, \tag{7}$$

$$Z \vdash \vec{D} \to w_i\ \mathbf{mr}_{\mathcal{A}}\ G_i^X \to (G_i \to \mathcal{A}(v_i)) \to \mathcal{A}(s_i w_i v_i) \qquad \textit{for } 1 \le i \le m, \tag{8}$$

$$Z \vdash \vec{D} \to \forall \vec{y}.\vec{G} \to \mathcal{A}(r\vec{y}). \tag{9}$$

Let M be a Z_0-derivation of $\vec{D} \to \forall\vec{y}(\vec{G} \to \bot) \to \bot$, and

$$s := \lambda\vec{y}\lambda\vec{w}.s_1 w_1(\dots(s_m w_m(r\vec{y}))\dots).$$

Then

$$Z \vdash \vec{D} \to \mathcal{A}([\![M^X]\!] t_1 \dots t_n s).$$

Proof. From the Z_0-derivation M we obtain by the substitution $\bot \mapsto X$ a Z_0^X-derivation $M^X \colon \vec{D}^X \to \forall\vec{y}(\vec{G}^X \to X) \to X$. The soundness theorem 4.1 yields

$$\begin{aligned}
&[\![M^X]\!]\ \mathbf{mr}_{\mathcal{A}}\ \big(\vec{D}^X \to \forall\vec{y}(\vec{G}^X \to X) \to X\big)\\
&= \forall\vec{u}\forall v.\vec{u}\ \mathbf{mr}_{\mathcal{A}}\ \vec{D}^X \to (v\ \mathbf{mr}_{\mathcal{A}}\ \forall\vec{y}.\vec{G}^X \to X) \to \mathcal{A}([\![M^X]\!]\vec{u}v)\\
&= \forall\vec{u}\forall v.\vec{u}\ \mathbf{mr}_{\mathcal{A}}\ \vec{D}^X \to \forall\vec{y}\forall\vec{w}\big(\vec{w}\ \mathbf{mr}_{\mathcal{A}}\ \vec{G}^X \to \mathcal{A}(v\vec{y}\vec{w})\big) \to \mathcal{A}([\![M^X]\!]\vec{u}v). \qquad (10)
\end{aligned}$$

Instantiate (10) with $\vec{t}$ for $\vec{u}$ and s for v. Clearly $\vec{t}\ \mathbf{mr}_{\mathcal{A}}\ \vec{D}^X$ is derivable from $\vec{D}$ by (7), so it remains to show $\vec{D} \to \vec{w}\ \mathbf{mr}_{\mathcal{A}}\ \vec{G}^X \to \mathcal{A}(s\vec{y}\vec{w})$.

Let $a_{m+1} := r\vec{y}$ and $a_i := s_i w_i a_{i+1}$, hence $s = \lambda\vec{y}\lambda\vec{w}\, a_1$. We show by induction on $j := m - i$

$$\vec{D} \to G_1 \to \dots \to G_i \to w_{i+1}\ \mathbf{mr}_{\mathcal{A}}\ G_{i+1}^X \to \dots \to w_m\ \mathbf{mr}_{\mathcal{A}}\ G_m^X \to \mathcal{A}(a_{i+1}). \qquad (11)$$

Basis. For $j = 0$ we have $i = m$ and (11) holds by (9). *Step.* From the IH (11) and the assumption (8) we obtain

$$\vec{D} \to G_1 \to \dots \to G_{i-1} \to w_i\ \mathbf{mr}_{\mathcal{A}}\ G_i^X \to \dots \to w_m\ \mathbf{mr}_{\mathcal{A}}\ G_m^X \to \mathcal{A}(s_i w_i a_{i+1}).$$

For $j = m$ we have $i = 0$ and hence we obtain from (11)

$$\vec{D} \to w_1\ \mathbf{mr}_{\mathcal{A}}\ G_1^X \to \dots \to w_m\ \mathbf{mr}_{\mathcal{A}}\ G_m^X \to \mathcal{A}(a_1),$$

which was to be shown. ∎

In order to apply theorem 5.1, we need $\mathcal{A} = \lambda y A_0$ and terms t_j, s_i, r such that (7)–(9) hold. The choice of $\mathcal{A}$ and r of course depends on the application at hand and should be done such that (9) holds. The rest follows from lemma 3.1 by the soundness theorem 4.1:

Theorem 5.2. *For every definite formula D and goal formula G we have terms t, s such that for an arbitrary $\mathcal{A} = \lambda y A_0$ with an $\mathcal{L}$-formula A_0:*

$$Z \vdash D \to t\ \mathbf{mr}_{\mathcal{A}}\ D^X, \qquad (12)$$

$$Z \vdash w\ \mathbf{mr}_{\mathcal{A}}\ G^X \to (G \to \mathcal{A}(v)) \to \mathcal{A}(swv) \qquad (13)$$

Proof. (12). Let N_D be the Z^X-derivation of $D \to D^X$ from lemma 3.1(4). The soundness theorem yields

$$Z \vdash [\![N_D]\!] \ \mathbf{mr}_{\mathcal{A}} \ (D \to D^X), \quad \text{i.e.} \quad Z \vdash D \to [\![N_D]\!] \ \mathbf{mr}_{\mathcal{A}} \ D^X.$$

(13). Let H_G be the Z^X-derivation of $G^X \to (G \to X) \to X$ from lemma 3.1(6). By the soundness theorem

$$Z \vdash [\![H_G]\!] \ \mathbf{mr}_{\mathcal{A}} \ (G^X \to (G \to X) \to X), \quad \text{i.e.}$$
$$Z \vdash w \ \mathbf{mr}_{\mathcal{A}} \ G^X \to (G \to \mathcal{A}(v)) \to \mathcal{A}([\![H_G]\!] wv).$$

$\square$

Corollary 5.3. *Let $\vec{D} = D_1, \dots, D_n$ be definite formulas and G a goal formula. Let M be a Z_0-derivation of $\vec{D} \to \forall y(G \to \bot) \to \bot$. Then*

$$Z \vdash \vec{D} \to G[y := [\![M^X]\!] t_1 \dots t_n s],$$

where $t_1, \dots, t_n, s$ are determined by the formulas $\vec{D}, G$ only.

6 Examples

We now want to give some simple examples of how to apply theorems 5.1 and 5.2. Here we will always have a single goal formula G and $\mathcal{A}$ will always be chosen as $\lambda y G$. Hence (9) trivially holds with $r := \lambda yy$.

6.1 Fibonacci Numbers

Let α_n be the n-th Fibonacci number, i.e.

$$\alpha_0 := 0, \quad \alpha_1 := 1, \quad \alpha_n := \alpha_{n-2} + \alpha_{n-1} \quad \text{for } n \geq 2.$$

We want to give a (classical) existence proof for the Fibonacci numbers. So we need to prove

$$\forall n \exists k \, G(n,k), \quad \text{i.e.} \quad \forall k(G(n,k) \to \bot) \to \bot$$

from assumptions expressing that G is the graph of the Fibonacci function, i.e.

$$G(0,0), \quad G(1,1), \quad \forall n \forall k \forall l. G(n,k) \to G(n+1,l) \to G(n+2,k+l).$$

Clearly the assumption formulas are definite and $G(n,k)$ is a goal formula. So theorems 5.1 and 5.2 can be applied without inserting double negations.

To construct a derivation, assume

$$\begin{aligned}
v_0 &: G(0,0), \\
v_1 &: G(1,1), \\
v_2 &: \forall n \forall k \forall l. G(n,k) \to G(n+1,l) \to G(n+2,k+l) \\
u &: \forall k. G(n,k) \to \bot.
\end{aligned}$$

Our goal is $\bot$. To this end we first prove a strengthened claim in order to get the induction through:

$$\forall n B \quad \text{with } B := \forall k \forall l (G(n,k) \to G(n+1,l) \to \bot) \to \bot.$$

This is proved by induction on n. The base case follows from v_0 and v_1. In the step case we can assume that we have k, l satisfying $G(n,k)$ and $G(n+1,l)$. We need k', l' such that $G(n+1,k')$ and $G(n+2,l')$. Using v_2 simply take $k' := l$ and $l' := k+l$. – To obtain our goal $\bot$ from $\forall n B$, it clearly suffices to prove its premise $\forall k \forall l. G(n,k) \to G(n+1,l) \to \bot$. So let k, l be given and assume $u_1 \colon G(n,k)$ and $u_2 \colon G(n+1,l)$. Then u applied to k and u_1 gives our goal $\bot$.

The derivation term is

$$\begin{aligned} M = & \lambda v_0^{G(0,0)} \lambda v_1^{G(1,1)} \lambda v_2^{\forall n \forall k \forall l. G(n,k) \to G(n+1,l) \to G(n+2,k+l)} \lambda u^{\forall k. G(n,k) \to \bot} \\ & \mathsf{Ind}_{n,B} M_{\text{base}} M_{\text{step}} n (\lambda k \lambda l \lambda u_1^{G(n,k)} \lambda u_2^{G(n+1,l)}. u k u_1) \end{aligned}$$

where

$$\begin{aligned} M_{\text{base}} = & \lambda w_0^{\forall k \forall l. G(0,k) \to G(1,l) \to \bot}. w_0 0 1 v_0 v_1 \\ M_{\text{step}} = & \lambda n \lambda w^B \lambda w_1^{\forall k \forall l. G(n+1,k) \to G(n+2,l) \to \bot}. \\ & \quad w(\lambda k \lambda l \lambda u_3^{G(n,k)} \lambda u_4^{G(n+1,l)}. w_1 l (k+l) u_4 (v_2 k l u_3 u_4)). \end{aligned}$$

Now let $\mathcal{A} := \lambda k G(n,k)$, and M^X be obtained from M by replacing every occurrence of $\bot$ by X. Therefore

$$[\![M^X]\!] = \lambda u^{\iota \to \iota}. \mathcal{R}_{\iota, (\iota \to \iota \to \iota) \to \iota} [\![M^X_{\text{base}}]\!] [\![M^X_{\text{step}}]\!] n (\lambda k \lambda l. u k)$$

where

$$\begin{aligned} [\![M^X_{\text{base}}]\!] &= \lambda w_0^{\iota \to \iota \to \iota}. w_0 0 1 \\ [\![M^X_{\text{step}}]\!] &= \lambda n \lambda w^{(\iota \to \iota \to \iota) \to \iota} \lambda w_1^{\iota \to \iota \to \iota}. w(\lambda k \lambda l. w_1 l (k+l)) \end{aligned}$$

Since there are no relevant formulas involved, the extracted term according to theorem 5.1 is

$$[\![M^X]\!] (\lambda x x) = \mathcal{R}_{\iota, (\iota \to \iota \to \iota) \to \iota} [\![M^X_{\text{base}}]\!] [\![M^X_{\text{step}}]\!] n (\lambda k \lambda l. k)$$

This algorithm might be easier to understand if we write it as a Scheme program:

```
(define (fibo n) (fibo1 n (lambda (k l) k)))

(define (fibo1 n1 f)
  (if (= n1 0)
      (f 0 1)
      (fibo1 (- n1 1) (lambda (k l) (f l (+ k l))))))
```

This is a linear algorithm in tail recursive form. It is somewhat unexpected since it passes λ-expressions (rather than pairs, as one would ordinarily do), and hence uses functional programming in a proper way. This clearly is related to the use of classical logic, which by its use of double negations has a functional flavour.

To remove some of the tedium of doing all that by hand, we certainly want machine help. We have done such an implementation within our system MINLOG; here is the original printout of the extracted term, with only some indentation added.

```
(lambda (n^1)
  (((((nat-rec-at
        (quote (arrow (arrow nat (arrow nat nat)) nat)))
      (lambda (hh^2) ((hh^2 (num 0)) (num 1))))
     (lambda (n^2)
       (lambda (ff^3)
         (lambda (hh^4)
           (ff^3 (lambda (n^5)
                   (lambda (n^6)
                     ((hh^4 n^6) ((plus-nat n^5) n^6)))))))))
    n^1) (lambda (n^2) (lambda (n^3) n^2))))
```

It is rather obvious that this can be translated into the SCHEME program above.

Remark. Of course, in this example there is no need to do the proof classically; in fact, it is more natural to work with the constructive existential quantifier $\exists^*$ instead. Here is the term extracted from this proof (original output of MINLOG).

```
(lambda (n^1)
  (car ((((nat-rec-at (quote (star nat nat)))
           (cons (num 0) (num 1)))
          (lambda (n^2)
            (lambda (nat*nat^3)
              (cons (cdr nat*nat^3)
                    ((plus-nat (car nat*nat^3))
                     (cdr nat*nat^3)))))) n^1)))
```

A more readable Scheme program is

```
(define (constr-fibo n) (car (constr-fibo-aux n)))

(define (constr-fibo-aux n)
  (if (= 0 n)
      (cons 0 1)
      (let ((prev (constr-fibo-aux (- n 1))))
        (cons (cdr prev)
              (+ (car prev) (cdr prev))))))
```

So the resulting algorithm is linear again, but passes pairs rather than λ-expressions.

6.2 Wellfoundedness of $\mathbb{N}$

There is an interesting phenomenon which may occur if we extract a program from a classical proof which uses the minimum principle. Consider as a simple example the wellfoundedness of $<$ on $\mathbb{N}$, i.e.

$$\forall f^{\iota \to \iota} \exists k. f(k+1) < f(k) \to \bot.$$

If one formalizes the classical proof "choose k such that $f(k)$ is minimal" and extracts a program one might expect that it computes a k such that $f(k)$ is minimal. But this is impossible! In fact the program computes the least k such that $f(k+1) < f(k) \to \bot$ instead. This discrepancy between the classical proof and the extracted program can of course only show up if the solution is not uniquely determined.

We begin with a rather detailed exposition of the classical proof, since we need a complete formalization. Our goal is $\exists k\, f(k) \le f(k+1)$, and the classical proof consists in using the minimum principle to choose a minimal element in $\mathsf{ran}(f) := \{\, y \mid \exists x\, f(x) = y \,\}$, the range of f. This suffices, for if we have such a minimal element, say y_0, then it must be of the form $f(x_0)$, and by the choice of y_0 we have $f(x_0) \le f(x)$ for every x, so in particular $f(x_0) \le f(x_0+1)$.

Next we need to prove the minimum principle from ordinary zero-successor-induction. The minimum principle

$$\exists k\, R(k) \to \exists k. R(k) \wedge \forall l{<}k. R(l) \to \bot \tag{14}$$

is to be applied with $R(k) := k \in \mathsf{ran}(f)$. Now (14) is logically equivalent to

$$\forall k \big(R(k) \to \forall l{<}k (R(l) \to \bot) \to \bot\big) \to \forall k. R(k) \to \bot \tag{15}$$

The premise of (15) expresses the "progressiveness" of $R(k) \to \bot$ w.r.t. $<$; we abbreviate it to

$$\mathsf{Prog} := \forall k. \forall l{<}k (R(l) \to \bot) \to R(k) \to \bot$$

We prove (15) by zero-successor-induction on n w.r.t. the formula

$$B := \forall k{<}n. R(k) \to \bot.$$

Base. $B[n := 0]$ follows easily from the lemma

$$v_1 \colon \forall m. m < 0 \to \bot.$$

Step. Let n be given and assume $w_2 \colon B$. To show $B[n := n+1]$ let k be given and assume $w_3 \colon k < n+1$. We will derive $R(k) \to \bot$ by using $w_1 \colon \mathsf{Prog}$ at k. Hence we have to prove

$$\forall l{<}k. R(l) \to \bot.$$

So, let l be given and assume further $w_4 \colon l < k$. From w_4 and $w_3 \colon k < n+1$ we infer $l < n$ (using an arithmetical lemma). Hence, by induction hypothesis $w_2 \colon B$ at l we get $R(l) \to \bot$.

Now a complete formalization is easy. We express $x \leq y$ by $y < x \to \bot$ and take $\forall x\, f(x) \neq k$ for $R(k) \to \bot$. The derivation term is

$$\begin{aligned} M := &\lambda v_1^{\forall m. m<0 \to \bot} \\ &\quad \lambda u^{\forall k. (f(k+1)<f(k)\to\bot)\to\bot}. \\ &\quad M_{\mathrm{cvind}}^{\mathsf{Prog}\to\forall y \forall x x. f(x)\neq y} M_{\mathrm{prog}}(f0)0L^{f0=f0} \end{aligned}$$

where

$$\begin{aligned} M_{\mathrm{cvind}} &= \lambda w_1^{\mathsf{Prog}} \lambda k. \mathsf{Ind}_{n,B} M_{\mathrm{base}} M_{\mathrm{step}}(k+1)kL^{k<k+1}, \\ M_{\mathrm{base}} &= \lambda k \lambda w_0^{k<0} \lambda x \lambda \tilde{w_0}^{f(x)=k}. v_1 k w_0, \\ M_{\mathrm{step}} &= \lambda n \lambda w_2^B \lambda k \lambda w_3^{k<n+1}. w_1 k (\lambda l \lambda w_4^{l<k}. w_2 l (L^{l<n}[w_4, w_3])), \end{aligned}$$

$$\begin{aligned} M_{\mathrm{prog}} = &\lambda k \lambda u_1^{\forall l. l<k \to \forall x f(x)\neq l} \lambda x \lambda u_2^{f(x)=k}. \\ &\quad u x \lambda w_5^{f(x+1)<f(x)}. u_1(f(x+1))L^{f(x+1)<k}[w_5, u_2](x+1)L^{f(x+1)=f(x+1)} \end{aligned}$$

Here we have used the abbreviations

$$\begin{aligned} \mathsf{Prog} &= \forall k. [\forall l. l < k \to \forall x. f(x) \neq l] \to \forall x. f(x) \neq k \\ B &= \forall k. k < n \to \forall x. f(x) \neq k \end{aligned}$$

For term extraction let

$$\mathcal{A} := \lambda k. f(k+1) < f(k) \to F,$$

and let M^X denote the result of replacing every formula C in the derivation M by C^X. Then

$$[\![M^X]\!] = \lambda v_1^{\iota\to\iota} \lambda u^{\iota\to\iota\to\iota}. [\![M_{\mathrm{cvind}}^X]\!][\![M_{\mathrm{prog}}^X]\!](f0)0$$

where

$$\begin{aligned} [\![M_{\mathrm{cvind}}^X]\!] &= \lambda w_1^{\iota\to(\iota\to\iota\to\iota)\to\iota\to\iota} \lambda k. \mathcal{R}_{\iota,\iota\to\iota\to\iota}[\![M_{\mathrm{base}}^X]\!][\![M_{\mathrm{step}}^X]\!](k+1)k \\ [\![M_{\mathrm{base}}^X]\!] &= \lambda k \lambda x. v_1 k \\ [\![M_{\mathrm{step}}^X]\!] &= \lambda n \lambda w_2^{\iota\to\iota\to\iota} \lambda k. w_1 k (\lambda l. w_2 l), \end{aligned}$$

$$[\![M_{\mathrm{prog}}^X]\!] = \lambda k \lambda u_1^{\iota\to\iota\to\iota} \lambda x. u x (u_1(f(x+1))(x+1)).$$

Note that k is not used in $[\![M_{\mathrm{prog}}^X]\!]$; this is the reason why the optimization below is possible.

Now by (12) we generally have $D \to [\![N_D]\!]\ \mathbf{mr}_{\mathcal{A}}\ D^X$ for every relevant definite formula D. In our case for $D = \forall k. k < 0 \to \bot$ we clearly can derive directly

$$\forall k(k < 0 \to \bot) \to (\lambda n 0)\ \mathbf{mr}_{\mathcal{A}}\ \forall k. k < 0 \to X,$$

since we can use ex-falso. So we may assume $[\![N_D]\!] = \lambda n 0$. Also, by (13) we generally have

$$w\ \mathbf{mr}_{\mathcal{A}}\ G^X \to (G \to \mathcal{A}(v)) \to \mathcal{A}([\![H_G]\!] wv).$$

In our case, with $G = f(k+1) < f(k) \to \bot$, we can derive directly

$$\begin{aligned}&(f(k+1) < f(k) \to \mathcal{A}(w)) \to ((f(k+1) < f(k) \to \bot) \to \mathcal{A}(v)) \to \\ &\mathcal{A}(\mathbf{if}\ f(k+1) < f(k)\ \mathbf{then}\ w\ \mathbf{else}\ v).\end{aligned}$$

So we may assume $[\![H_G]\!] = \lambda w \lambda v.\mathbf{if}\ f(k+1) < f(k)\ \mathbf{then}\ w\ \mathbf{else}\ v$. Now let

$$s := \lambda k \lambda w.[\![H_G]\!] wk = \lambda k \lambda w.\mathbf{if}\ f(k+1) < f(k)\ \mathbf{then}\ w\ \mathbf{else}\ k.$$

Then the extracted term according to theorem 5.1 is

$$[\![M^X]\!][\![N_D]\!] s =_\beta [\![M^X_{\mathrm{cvind}}]\!]' [\![M^X_{\mathrm{prog}}]\!]' (f0)0$$

where $'$ indicates substitution of $[\![N_D]\!]$, s for v_1, u, so

$$\begin{aligned}[\![M^X_{\mathrm{cvind}}]\!]' &=_{\beta\eta} \lambda w_1 \lambda k'.\mathcal{R}(\lambda k \lambda x 0)(\lambda n \lambda w_2 \lambda k.w_1 k w_2)(k'+1)k', \\ [\![M^X_{\mathrm{prog}}]\!]' &=_\beta \lambda k \lambda x u_1 \lambda x.\mathbf{if}\ f(x+1) < f(x)\ \mathbf{then}\ u_1(f(x+1))(x+1)\ \mathbf{else}\ x\end{aligned}$$

Therefore we obtain as extracted algorithm

$$\begin{aligned}&[\![M^X]\!][\![N_D]\!] s =_\beta \\ &\mathcal{R}(\lambda k \lambda x.0) \\ &\quad(\lambda n \lambda w_2^{\iota\to\iota\to\iota} \lambda k \lambda x.\mathbf{if}\ f(x+1) < fx\ \mathbf{then}\ w_2(f(x+1))(x+1)\ \mathbf{else}\ x) \\ &\quad((f0)+1)(f0)0.\end{aligned}$$

To make this algorithm more readable we may write

$$[\![M^X]\!][\![N_D]\!] s = h(f(0)+1, f(0), 0),$$

where

$$\begin{aligned}h(0,k,x) &= 0 \\ h(n+1,k,x) &= \mathbf{if}\ f(x+1) < fx\ \mathbf{then}\ h(n, f(x+1), x+1)\ \mathbf{else}\ x\end{aligned}$$

The extracted program (original output of Minlog) is

```
(lambda (h^1)
  ((((((nat-rec-at (quote (arrow nat (arrow nat nat))))
       (lambda (n^2)
         (lambda (n^3) n000)))
      (lambda (n^2)
        (lambda (hh^3)
          (lambda (n^4)
```

```
          (lambda (n^5)
            (((if-nat
               ((<-strict-nat (h^1 ((plus-nat n^5) (num 1))))
                (h^1 n^5)))
              ((hh^3 (h^1 ((plus-nat n^5) (num 1))))
               ((plus-nat n^5) (num 1))))
             n^5))))))
     ((plus-nat (h^1 (num 0))) (num 1)))
    (h^1 (num 0)))
   (num 0)))
```

We can rewrite this as a SCHEME program as follows.

```
(define (wf f) (wf-aux f (+ (f 0) 1) (f 0) 0))

(define (wf-aux f n k x)
  (if (= 0 n)
      0
      (if (< (f (+ x 1)) (f x))
          (wf-aux f (- n 1) (f (+ x 1)) (+ x 1))
          x)))
```

Note that k is not used here (this will always happen if the induction principle is used in the form of the minimum principle only), and hence we may optimize our program to

```
(define (wf1 f) (wf1-aux f (+ (f 0) 1) 0))

(define (wf1-aux f n x)
  (if (= 0 n)
      0
      (if (< (f (+ x 1)) (f x))
          (wf-aux f (- n 1) (+ x 1))
          x)))
```

Now it is immediate to see that the program computes the least k such that $f(k+1) < f(k) \to \bot$, where $f(0)+1$ only serves as an upper bound for the search.

6.3 Towards More Interesting Examples

VELDMAN and BEZEM [19] suggested DICKSON's Lemma [7] as an interesting case study for program extraction from classical proofs. It states that for k given infinite sequences $f_1, \dots, f_k$ of natural numbers and a given number l there are indices $i_1, \dots, i_l$ such that *every* sequence f_κ increases on $i_1, \dots, i_l$, i.e. $f_\kappa(i_1) \leq \dots \leq f_\kappa(i_l)$ for $\kappa = 1, \dots, k$. Here is a short classical proof, using the minimum principle for undecidable sets.

Call a unary predicate (or set) $Q \subseteq \mathbb{N}$ *unbounded* if $\forall x \exists y. Q(y) \wedge x < y$.

Lemma 6.1. *Let Q be unbounded and f a function from a superset of Q to $\mathbb{N}$. Then the set Q_f of left f-minima w.r.t. Q is unbounded; here*

$$Q_f(x) := Q(x) \wedge \forall y.Q(y) \to x < y \to f(x) \leq f(y).$$

Proof. Let x be given. We must find y with $Q_f(y)$ and $x < y$. The minimum principle for $\{ y \mid Q(y) \wedge x < y \}$ with measure f yields

$$(\exists y.Q(y) \wedge x < y) \to \exists y.Q(y) \wedge x < y \wedge \forall z.Q(z) \to x < z \to f(y) \leq f(z). \tag{16}$$

Since Q is assumed to be unbounded, the premise is true. We show that the y provided by the conclusion satisfies $Q_f(y)$, i.e.

$$Q(y) \wedge \forall z.Q(z) \to y < z \to f(y) \leq f(z).$$

So let z with $Q(z)$ and $y < z$ be given. From $x < y$ we obtain $x < z$, hence $f(y) \leq f(z)$ by the conclusion of (16). □

Lemma 6.2. *Let Q be unbounded and $f_1, \dots, f_k$ be functions from a superset of Q to $\mathbb{N}$. Then there is an unbounded subset Q_1 of Q such that $f_1, \dots, f_k$ increase on Q_1, i.e.*

$$Q_1(x) \wedge Q_1(y) \wedge x < y \to \bigwedge_{\kappa=1}^{k} f_\kappa(x) \leq f_\kappa(y).$$

Proof. By induction on k. Let Q_2 be Q if $k = 1$, and in case $k \geq 2$ be an unbounded subset of Q where $f_2, \dots, f_k$ increase (i.e. given by the induction hypothesis for $f_2, \dots, f_k$). Let Q_1 be the set of left f_1-minima w.r.t. Q_2, i.e.

$$Q_1(x) := Q_2(x) \wedge \forall y.Q_2(y) \to x < y \to f_1(x) \leq f_1(y).$$

By lemma 6.1 Q_1 is an unbounded subset of Q_2. Now on Q_1 f_1 increases, and because of $Q_1 \subseteq Q_2$ also $f_2, \dots, f_k$ increase. □

Corollary 6.3. *For every k, l we have*

$$\forall f_1, \dots, f_k \exists i_0, \dots, i_l \bigwedge_{\lambda < l} .i_\lambda < i_{\lambda+1} \wedge \bigwedge_{\kappa=1}^{k} f_\kappa(i_\lambda) \leq f_\kappa(i_{\lambda+1}). \quad \square$$

For $k = 2$ (i.e. two sequences) this example has been treated in [4]. However, it is interesting to look at the general case, since then the brute force search takes time $O(n^k)$, and we can hope that the program extracted from the classical proof is better.

Acknowledgements Klaus Weich originally proposed the functional algorithm computing the Fibonacci numbers. Monika Seisenberger – apart from being a coauthor of [4] – and Felix Joachimski have contributed a lot to the Minlog system, particularly to the implementation of the translation of classical proofs into constructive ones. We also benefitted from helpful comments by Peter Selinger and Matteo Slanina, who presented this material in a seminar in Stanford, in the fall of 2000.

References

[1] Franco Barbanera and Stefano Berardi. Extracting constructive content from classical logic via control–like reductions. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, pages 45–59. LNCS Vol. 664, 1993.

[2] Ulrich Berger. Programs from classical proofs. In M. Behara, R. Fritsch, and R.G. Lintz, editors, *Symposia Gaussiana. Proceedings of the 2nd Gauss Symposium. Conference A: Mathematics and Theoretical Physics. Munich, Germany, August 2-7, 1993*, pages 187–200, Berlin, New York, 1995. Walter de Gruyter.

[3] Ulrich Berger and Helmut Schwichtenberg. Program extraction from classical proofs. In D. Leivant, editor, *Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, IN, USA, October 1994*, volume 960 of *LNCS*, pages 77–97. Springer Verlag, Berlin, Heidelberg, New York, 1995.

[4] Ulrich Berger, Helmut Schwichtenberg, and Monika Seisenberger. The Warshall Algorithm and Dickson's Lemma: Two Examples of Realistic Program Extraction. To appear: Journal of Automated Reasoning.

[5] Robert L. Constable and Chetan Murthy. Finding computational content in classical proofs. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 341–362. Cambridge University Press, 1991.

[6] Thierry Coquand and Hendrik Persson. Gröbner Bases in Type Theory. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Types for Proofs and Programs*, volume 1657 of *LNCS*. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[7] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Am. J. Math*, 35:413–422, 1913.

[8] Anne S. Troelstra (editor). *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1973.

[9] Matthias Felleisen, Daniel P. Friedman, E. Kohlbecker, and B.F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[10] Matthias Felleisen and R. Hieb. The revised report on the syntactic theory of sequential control and state. *Theoretical Computer Science*, 102:235–271, 1992.

[11] Harvey Friedman. Classically and intuitionistically provably recursive functions. In D.S. Scott and G.H. Müller, editors, *Higher Set Theory*, volume

669 of *Lecture Notes in Mathematics*, pages 21–28. Springer Verlag, Berlin, Heidelberg, New York, 1978.

[12] Timothy G. Griffin. A formulae–as–types notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, 1990.

[13] Ulrich Kohlenbach. Analysing proofs in analysis. In W. Hodges, M. Hyland, C. Steinhorn, and J. Truss, editors, *Logic: from Foundations to Applications. European Logic Colloquium (Keele, 1993)*, pages 225–260. Oxford University Press, 1996.

[14] Jean-Louis Krivine. Classical logic, storage operators and second-order lambda-calculus. *Annals of Pure and Applied Logic*, 68:53–78, 1994.

[15] Daniel Leivant. Syntactic translations and provably recursive functions. *The Journal of Symbolic Logic*, 50(3):682–688, September 1985.

[16] Chetan Murthy. Extracting constructive content from classical proofs. Technical Report 90–1151, Dep.of Comp.Science, Cornell Univ., Ithaca, New York, 1990. PhD thesis.

[17] Michel Parigot. $\lambda\mu$–calculus: an algorithmic interpretation of classical natural deduction. In *Proc. of Log. Prog. and Automatic Reasoning, St. Petersburg*, volume 624 of *LNCS*, pages 190–201. Springer Verlag, Berlin, Heidelberg, New York, 1992.

[18] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics. An Introduction*, volume 121, 123 of *Studies in Logic and the Foundations of Mathematics*. North–Holland, Amsterdam, 1988.

[19] Wim Veldman and Marc Bezem. Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics. Logic Group Preprint Series 72, University of Utrecht, Dept of Philosophy, January 1992.