UNIFORM HEYTING ARITHMETIC

ULRICH BERGER

Dedicated to Helmut Schwichtenberg on his 60th Birthday

Abstract. We present an extension of Heyting Arithmetic in finite types called *Uniform Heyting Arithmetic* (HA^{u}) that allows for the extraction of optimized programs from constructive and classical proofs. The system HA^{u} has two sorts of first-order quantifiers: ordinary quantifiers governed by the usual rules, and uniform quantifiers subject to stronger variable conditions expressing roughly that the quantified object is not computationally used in the proof. We combine a Kripke-style Friedman/Dragalin translation which is inspired by work of Coquand and Hofmann and a variant of the refined A-translation due to Buchholz, Schwichtenberg and the author to extract programs from a rather large class of classical first-order proofs while keeping explicit control over the levels of recursion and the decision procedures for predicates used in the extracted program.

§1. Introduction.

According to the Brouwer-Heyting-Kolmogorov interpretation of constructive logic a proof is a construction providing evidence for the proven formula [20]. Viewing this interpretation from a data-oriented perspective one arrives at the so-called proofs-as-programs paradigm associating a constructive proof with a program 'realizing' the proven formula. This paradigm has been proposed as the basis of a new methodology for the synthesis of provably correct software. In several proof systems supporting computer aided program synthesis from proofs (e.g. Agda, Coq, Minlog, NuPrL, PX) rather large case studies have been carried out which seem to indicate that this indeed opens a promising path to secure software. An objection often raised against this approach is that it seems to be too 'indirect' to be a viable programming technique, i.e. when proving a formula we do not 'see' the program and hence do not have enough control over it.

The 'gap' between a proof and the program it represents appears to be particularly large when the proofs-as-program paradigm is extended to (a restricted class of) classical, i.e. non-constructive, proofs (see e.g. [15], [1], [14], [7], [12], [8] for different methods of extracting computational content from classical proofs). For example, if we prove classically the well-foundedness of the usual ordering on the natural numbers, i.e.

$$\forall f\colon \mathbb{N} \to \mathbb{N}\, \exists n\colon \mathbb{N}\ f(n) \not> f(n+1),$$

by applying the least element principle to the range of f, it seems that the program contained in this proof should compute for each function f the point n where f assumes its minimum, which is impossible, of course. What the extracted program really does is to search from $n := f(0) + 1$ downwards until

University of Wales Swansea.

an n is found where $f(n) \not> f(n+1)$. This search strategy does not seem to be visible in the proof [6].

Nevertheless a lot of work has been done to develop the proofs-as-programs paradigm into a realistic programming methodology, and refined program extraction procedures adapted to particular situations have been developed yielding optimized programs and providing a fair amount of explicit control over essential parameters of the extracted programs (e.g. [12], [6], [17]). This paper intends to make contributions in a similar direction through a formal system called *Uniform Heyting Arithmetic*, HA^{u}. The system HA^{u} is an extension of Heyting arithmetic in finite types with two sorts of first-order quantifiers, ordinary quantifiers governed by the usual logical rules and *uniform quantifiers* which are subject to stronger variable conditions expressing roughly that the quantified object is not computationally used in the proof. As a technical device we also include second-order universal quantifiers with arithmetic comprehension.

Our work brings together ideas and results from the papers [4], [9] and [6]. From [4] we have taken the uniform quantifier $\{\forall x\}$ introduced there to extract the well-know normalization-by-evaluation program from the Tait/Troelstra proof of strong normalization of the simply typed lambda calculus. The effect of the uniform quantifier is that certain unnecessary parameters do not occur in the extracted program.

The second crucial idea is a Kripke-style translation introduced by Coquand and Hofmann [9] to give an elegant proof that with respect to Π^0_2 sentences classical Σ^0_1-arithmetic is conservative over its intuitionistic counterpart (which also follows from a result of Parsons' [16] that Σ^0_1-arithmetic is conservative over primitive recursive arithmetic). The benefit of the Kripke translation will be that the levels of recursion in extracted programs are lower than in the ordinary A-translation used in [6]. Although [9] is mainly concerned with foundational questions the authors of [9] also express their belief that their method "should give rise to a more efficient program extraction procedure". In this paper we address this issue. From a (practical) computational point of view one problem with the method in [9] is that in the translation quantifier free predicates are coded into functions which leads to a proliferation of case analyses in extracted programs and hence to inefficiency (see the Fibonacci example in section 8). We will get around this problem by allowing proper second-order quantification. Using a normalization argument we will, however, at the end get rid of second-order rules (theorem 2.3).

The main inputs from [6] are the concepts of 'definite' and 'goal' formulas which fit nicely with the Kripke translation and allow us to reduce the number of predicates that need to be double negated and hence to further reduce the number of predicates for which decidability is needed in the extracted programs.

We illustrate our methods of extracting programs form proofs by some simple examples.

§2. The system HA^{u}.

The types and terms of system HA^{u} are (a minor extension of) the terms of Gödel's system T in its formulation as an applied typed lambda-calculus.

Types, $\rho, \sigma, \ldots$, are built from the base types boole and nat (and possibly other base types), by products, $\rho\times\sigma$, and function spaces, $\rho \to \sigma$. The *level* of a type is defined as usual by $\mathsf{lev}(\tau) = 0$ if τ is a base type, $\mathsf{lev}(\rho\times\sigma) = \max(\mathsf{lev}(\rho), \mathsf{lev}(\sigma))$ and $\mathsf{lev}(\rho\to\sigma) = \max(1 + \mathsf{lev}(\rho), \mathsf{lev}(\sigma))$. If $\mathcal{T}$ is a set of nonempty types we set $\tau(\mathcal{T}) = \max\{\mathsf{lev}\rho \mid \rho \in \mathcal{T}\}$; $\tau(\emptyset) = -1$.

Terms, $r^\rho, s^\sigma, \ldots$, are built from typed variables, $x^\rho, \ldots$, typed constants, c^ρ, including 0^{nat}, $(+1)^{\mathsf{nat}\to\mathsf{nat}}$, $\mathsf{True}^{\mathsf{boole}}$, $\mathsf{False}^{\mathsf{boole}}$, $\mathsf{If}_\rho{}^{\mathsf{boole}\to\rho\to\rho\to\rho}$ (case analysis at type ρ) and $\mathsf{Rec}_\rho{}^{\rho\to(\mathsf{nat}\to\rho\to\rho)\to\mathsf{nat}\to\rho}$ (Gödel primitive recursion at type ρ), $\mathsf{Zero}^{\mathsf{nat}\to\mathsf{boole}}$, by pairing, $\mathsf{p}(r^\rho, s^\sigma)^{\rho\times\sigma}$, projection, $\mathsf{p}_i(r^{\rho_0\times\rho_1})^{\rho_i}$ $(i = 0, 1)$, abstraction, $(\lambda x^\rho.r^\sigma)^{\rho\to\sigma}$, and application, $(r^{\rho\to\sigma}s^\rho)^\sigma$. In addition to the usual β and η rules we assume rewrite rules for the constants including

$$\begin{array}{lll} \mathsf{If}_\rho\,\mathsf{True}\,r\,s \mapsto r & \mathsf{Rec}_\rho\,r\,s\,0 \mapsto r & \mathsf{Zero}\,0 \mapsto \mathsf{True} \\ \mathsf{If}_\rho\,\mathsf{False}\,r\,s \mapsto s & \mathsf{Rec}_\rho r\,s\,(t+1) \mapsto s\,t\,(\mathsf{Rec}_\rho\,r\,s\,t) & \mathsf{Zero}\,(t+1) \mapsto \mathsf{False} \end{array}$$

such that the resulting rewrite system is confluent and strongly normalizing. We let $\mathsf{nf}(r)$ be the normal form of r with respect to this rewrite system. We let $\mathsf{reclev}(r)$ (recursion level of r) be the maximum of all $\mathsf{lev}(\rho)$ such that the recursor Rec_ρ occurs in the term r provided r contains at least one recursor. Otherwise $\mathsf{reclev}(r) := -1$.

The *atomic formulas* of HA^{u} are $\bot$, $P(\vec{t})$ and $X(\vec{t})$ where $P^{(\vec{\rho})}$ is a predicate constant, $X^{(\vec{\rho})}$ is a predicate variable and $\vec{t}^{\vec{\rho}}$ are terms. We allow predicate constants and predicate variables of arbitrary arities $(\vec{\rho})$. The set of predicate constants is not fixed, but we assume that we have equality, $=_\rho$, of arity (ρ, ρ) for every type ρ.

The *formulas* A, B of HA^{u} are atomic formulas, $A \wedge B$, $A \to B$, $\forall x\, A$, $\exists x\, A$, $\{\forall x\}A$, $\{\exists x\}A$, $\forall_2 X\, A$. The quantifiers $\{\forall x\}$ and $\{\exists x\}$ are called *uniform quantifiers*.

We set $\forall(A) = \forall\vec{x}\, A$ where $\vec{x}$ is a listing of $\mathsf{FV}(A)$, the set of free object variables of A (excluding predicate variables). A formula is called *quantifier free* if it contains no predicate variables and no quantifiers other than $\forall p^{\mathsf{boole}}$ or $\exists p^{\mathsf{boole}}$ (note that uniform boolean quantifiers are excluded). Given a set P of predicate constants, $\mathsf{qf}(\mathsf{P})$ denotes the set of quantifier free formulas containing predicate constants from P only and no predicate variables. By $\mathsf{pc}(A)$ we denote the set of predicate constants occurring in A. By $A[t/x]$ we denote A where t is substituted for every free occurrence of x (renaming bound variables if necessary). Frequently, type information will be suppressed when derivable from the context or irrelevant. Terms and formulas coinciding up to renaming of bound variables are identified. In order to save brackets we will frequently write, e.g. $\forall x\,.\,A \to B$ for $\forall x\,(A \to B)$, whereas $\forall x\, A \to B$ means $(\forall x\, A) \to B$. We set

$$\begin{array}{rcl} \neg A & = & A \to \bot \\ A \vee B & = & \exists p.(p = \mathsf{True} \to A) \wedge (p = \mathsf{False} \to B) \end{array}$$

The quantifier $\{\forall x\}$ was first used in [4] to optimize the program extracted from the normalization proof for the simply typed lambda calculus based on computability predicates. The idea behind uniform quantifiers is to express constructions that are independent of the quantified variable. For example, a proof of $\{\exists x\}A$ represents a construction of $A[t/x]$ for some term t, 'forgetting' the witness t whereas the construction represented by a proof of $\exists x\, A$ includes the witnessing term t. A more precise explanation in terms of realizability will be given in section 3.

A formula is called a *first-order formula* if it doesn't contain predicate variables (and hence no second-order quantifiers), it is called an *arithmetic formula* if doesn't contain second-order quantifiers (but may contain free predicate variables), and it is called a *Harrop formula* if it contains neither the quantifier $\exists x$ nor any predicate variable in a strictly positive position [1] (it may contain $\{\exists x\}$ strictly positively though).

If $\vec{x}^{\vec{\rho}}$ are object variables and B is a formula, then $\lambda\vec{x}^{\vec{\rho}}B$ is called a *predicate of arity* $(\vec{\rho})$. If $\mathcal{P} = \lambda\vec{x}^{\vec{\rho}}B$ is a predicate and $\vec{r}^{\vec{\rho}}$ are terms then $\mathcal{P}(\vec{r})$ denotes $B[\vec{r}/\vec{x}]$. If A is a formula, X a predicate variable and $\mathcal{P}$ a predicate, both of arity $\vec{\rho}$, we denote by $A[\mathcal{P}/X]$ the result of replacing in A every subformula of the form $X(\vec{r})$ by $\mathcal{P}(\vec{r})$ (possibly renaming bound variables). We tacitly extend properties of formulas to properties of predicates in the obvious way. For example, an *arithmetic predicate* is a predicate $\lambda\vec{x}B$ where B is an arithmetic formula. A predicate of arity $()$, say λA, will be identified with the formula A. Also a propositional variable, that is, predicate variable X of arity $()$, will be identified with the formula $X()$.

The *derivation* rules of HA^{u} are displayed in figures 1 and 2. In a sequent $\Gamma \vdash M\colon A$ the context Γ is a finite set of assumptions $\{u_1^{B_1}, \dots, u_n^{B_n}\}$ where the symbols u_i are all different and will be sometimes omitted. Usually we will also omit the set-forming curly brackets and write a comma to denote the union of two sets. The variable conditions are as usual: $(\dagger)$ = 'x respectively X not free in Γ', $(\dagger\dagger)$ = 'x not free in B'. The rules are divided into *normal rules* (figure 1), which are as expected, and *uniform* and *second-order rules* (figure 2). The uniform rule $\{\forall\}^+$ refers to the set $\mathsf{CV}(M)$ of *computationally relevant variables* of a derivation M which in turn refers to the notion of a *computationally relevant term.* By the latter we mean a term t that is used in M as an instantiating term in an $\forall$-elimination rule, $\forall^-(N,t)$, or as a witnessing term in an $\exists$-introduction rule, $\exists^+_{\lambda xA}(t,N)$ where the occurrence of that rule is not within a subderivation deriving a Harrop formula. A *computationally relevant variable* is a free variable of a derivation [2] which occurs (free) in a computationally relevant term in M.

The rules of HA^{u} are designed such that when extracting programs from first-order derivations (section 3) we may ignore the uniform rules, except for the $\{\exists\}^-$ rule, which can be dealt with in a simplified way. In this paper the second-order

[1]A position in a formula is called *strictly positive* if it is not contained in the premise of an implication.

[2]An occurrence of a variable is free in a derivation if that occurrence is not bound by a quantifier or a λ-abstraction.

$\dfrac{}{\Gamma, u^A \vdash u^A\colon A}$	
$\dfrac{\Gamma \vdash M_0\colon A_0 \qquad \Gamma \vdash M_1\colon A_1}{\Gamma \vdash \wedge^+(M_0, M_1)\colon A_0 \wedge A_1}$	$\dfrac{\Gamma \vdash M\colon A_0 \wedge A_1}{\Gamma \vdash \wedge_i^-(M)\colon A_i}$
$\dfrac{\Gamma, u^A \vdash M\colon B}{\Gamma \vdash \to^+ (\lambda u^A M)\colon A \to B}$	$\dfrac{\Gamma \vdash M\colon A \to B \qquad \Gamma \vdash N\colon A}{\Gamma \vdash \to^- (M, N)\colon B}$
$\dfrac{\Gamma \vdash M\colon A}{\Gamma \vdash \forall^+(\lambda x M)\colon \forall x\, A}\ (\dagger)$	$\dfrac{\Gamma \vdash M\colon \forall x\, A}{\Gamma \vdash \forall^-(M, t)\colon A[t/x]}$
$\dfrac{\Gamma \vdash M\colon A[t/x]}{\Gamma \vdash \exists^+_{\lambda x A}(t, M)\colon \exists x\, A}$	$\dfrac{\Gamma \vdash M\colon \exists x\, A \;\; \Gamma \vdash N\colon \forall x\, .A \to B}{\Gamma \vdash \exists^-(M, N)\colon B}\ (\dagger\dagger)$
$\dfrac{\mathsf{nf}(r) =_\alpha \mathsf{nf}(s)}{\Gamma \vdash \mathsf{Refl}(\mathsf{r}, \mathsf{s})\colon r = s}$	$\dfrac{\Gamma \vdash M\colon A[r/x] \qquad \Gamma \vdash N\colon r = s}{\Gamma \vdash \mathsf{Comp}_{\lambda x A}(M, N)\colon A[s/x]}$
$\dfrac{\Gamma \vdash M\colon \mathsf{True} = \mathsf{False}}{\Gamma \vdash \mathsf{Cont}(M)\colon \bot}$	$\dfrac{\Gamma \vdash M\colon \bot}{\Gamma \vdash \mathsf{Efq}_A(M)\colon A}$
$\dfrac{\Gamma \vdash M{:}A[\mathsf{True}/p] \;\; \Gamma \vdash N{:}A[\mathsf{False}/p]}{\Gamma \vdash \mathsf{Case}_{\lambda p A}(M, N)\colon \forall p\, A}$	
$\dfrac{\Gamma \vdash M{:}A[0/n] \;\; \Gamma \vdash N{:}\forall n\, .A{\to}A[n{+}1/n]}{\Gamma \vdash \mathsf{Ind}_{\lambda n A}(M, N)\colon \forall n\, A}$	

FIGURE 1. The normal rules of HA^{u}

part of HA^{u} will only be used as a technical tool to analyze its first-order part and for that purpose among the predicate variables only propositional variables will be needed.

Remark: An interesting variant of HA^{u} is obtained when in the comprehension rule $\forall_2^-(M, \mathcal{P})$ arbitrary Harrop predicates $\mathcal{P}$ are allowed. Although this system

$\dfrac{\Gamma \vdash M{:}A \quad x \notin \mathsf{CV}(M)}{\Gamma \vdash \{\forall\}^+(\lambda x M)\colon \{\forall x\}A}\ (\dagger)$	$\dfrac{\Gamma \vdash M\colon \{\forall x\}A}{\Gamma \vdash \{\forall\}^-(M,t)\colon A[t/x]}$
$\dfrac{\Gamma \vdash M\colon A[t/x]}{\Gamma \vdash \{\exists\}^+_{\lambda x A}(t,M)\colon \{\exists x\}A}$	$\dfrac{\Gamma \vdash M{:}\{\exists x\}A \quad \Gamma \vdash N{:}\{\forall x\}.A \to B}{\Gamma \vdash \{\exists\}^-(M,N)\colon B}\ (\dagger\dagger)$
$\dfrac{\Gamma \vdash M\colon A}{\Gamma \vdash \forall_2^+(\lambda X\, M)\colon \forall_2 X\, A}\ (\dagger)$	$\dfrac{\Gamma \vdash M\colon \forall_2 X\, A \qquad \mathcal{P} \text{ arithmetic}}{\Gamma \vdash \forall_2^-(M,\mathcal{P})\colon A[\mathcal{P}/X]}$

FIGURE 2. The uniform and second-order rules of HA^{u}

is proof-theoretically equivalent to full second-order arithmetic it still has a realizability interpretation by Gödel primitive recursive functionals (see also the remarks in section 3).

A main goal of this paper is to control the recursions occurring in a program extracted from a derivation in terms of the inductions used and the recursion operators occurring in computationally relevant terms. We write

$$\Gamma \vdash^{\mathcal{I}} M\colon A$$

if $\Gamma \vdash M\colon A$ and $\mathcal{I}$ is a set of predicates and types such that the following two conditions are satisfied. (1) in M every induction rule is of the form $\mathsf{Ind}_{\lambda n C}(N_0, N_1)$ where C is an *arithmetic instance* of $\mathcal{I}$, i.e. $C = B[\vec{\mathcal{P}}, \vec{t}/\vec{X}, \vec{x}]$ with $\lambda n B \in \mathcal{I}$, $\vec{\mathcal{P}}$ arithmetic predicates, $\vec{t}$ terms of appropriate arities and types respectively, (2) all recursors, Rec_ρ, occurring in a computationally relevant term in M have a type $\rho \in \mathcal{I}$ (it may seem a bit crude to throw predicates and types into one set, but we find this appropriate in order not to let notations become too cumbersome). We write $\Gamma \vdash A$ respectively $\Gamma \vdash^{\mathcal{I}} A$ if $\Gamma \vdash M\colon A$ respectively $\Gamma \vdash^{\mathcal{I}} M\colon A$ for some derivation M. We also write

$$\Gamma \vdash^{\mathcal{I}}_1 M : A \text{ (or } \Gamma \vdash^{\mathcal{I}}_1 A)$$

if $\Gamma \vdash^{\mathcal{I}} M : A$ where M is a *first-order derivation*, that is, in M no second-order rules occur.

It will be convenient to have two ways of (intuitionistically) expressing decidability of a predicate. The first is to postulate the law of excluded middle.

$$\mathsf{LEM}(P) := \forall \vec{x}\,.\, P(\vec{x}) \vee \neg P(\vec{x})$$

The second possibility is to explicitely state the existence of the characteristic function of a predicate. To this end we fix for every predicate constant P of arity $(\vec{\sigma})$ a variable f_P of type $\vec{\sigma} \to \mathsf{boole}$ and set

$$\mathsf{DEC}(P) := \forall \vec{x}\ .\ P(\vec{x}) \leftrightarrow f_P \vec{x} = \mathsf{True}.$$

Using the variables f_P we can in the usual way define for every quantifier free formula C a characteristic term t_C of type boole as follows. $t_{P(\vec{t})} = f_P(\vec{t})$, $t_\perp = \mathsf{False}$, $t_{A\wedge B} = \mathsf{If}\, t_A\, t_B\, \mathsf{False}$, $t_{A\to B} = \mathsf{If}\, t_A\, t_B \mathsf{True}$, $t_{\forall p\, A} = \mathsf{If}\, t_{A[\mathsf{True}/p]}\, t_{A[\mathsf{False}/p]} \mathsf{False}$, $t_{\exists p\, A} = \mathsf{If}\, t_{A[\mathsf{True}/p]}\, \mathsf{True}\, t_{A[\mathsf{False}/p]}$. Clearly t_C is recursion free.

In the following we let P be a set of predicate constants and set $\mathsf{DEC}(\mathsf{P}) := \{\mathsf{DEC}(P) \mid P \in \mathsf{P}\}$.

Lemma 2.1. (i) $\mathsf{LEM}(\mathsf{P}) \vdash_1^\emptyset B \vee \neg B$ *for each* $B \in \mathsf{qf}(\mathsf{P})$.
(ii) $\mathsf{DEC}(P) \vdash_1^\emptyset \mathsf{LEM}(P)$.
(iii) $\mathsf{DEC}(\mathsf{pc}(C)) \vdash_1^\emptyset C \leftrightarrow t_c = \mathsf{True}$ *for every quantifier free formula* C.

Proof. *(i).* Easy induction on formulas in $\mathsf{qf}(\mathsf{P})$. *(ii).* Obvious. *(iii).* Induction on C. ⊣

Above we extended the operations $\mathsf{LEM}(\cdot)$ and $\mathsf{DEC}(\cdot)$ from predicates to set of predicates. In a similar way we will often tacitly extend an operation on a certain kind of syntactic objects (e.g. types, predicate symbols, formulas, predicates) to sets of such or similar objects. For example, if $\mathcal{I}$ is a set of predicates or types (as occurring in '$\Gamma \vdash^{\mathcal{I}} A$'), then $\neg\mathcal{I}$ naturally denotes $\{\lambda x \neg\mathcal{P}(x) \mid \mathcal{P} \in \mathcal{I} \text{ a predicate}\} \cup \{\rho \mid \rho \in \mathcal{I} \text{ a type}\}$.

We close this section with a simple but important conservativity result. Note that we can view our system HA^{u} as an extension of the intuitionistic finite type analogue to the classical system ACA of second-order arithmetic with arithmetic comprehension and unrestricted induction. If in ACA one restricts induction to arithmetic predicates one obtains the system ACA_0 which is conservative over Peano Arithmetic [18]. Below we show that the analogous system, that is, HA^{u} with induction restricted to arithmetic predicates, is conservative over the first-order part of HA^{u}. However we will not argue model theoretically (as done in [18]), but use proof-theoretic methods instead. We will need the following substitution lemma.

Lemma 2.2. *If* $\Gamma \vdash M : A$, *then* $\Gamma[t/x] \vdash M[t/x] : A[t/x]$ *for any term* t *and* $\Gamma[\mathcal{P}/X] \vdash M[\mathcal{P}/X] : A[\mathcal{P}/X]$ *for every arithmetic predicate* $\mathcal{P}$.

Proof. Induction on M. Here it is important that in a Harrop formula no predicate variable occurs strictly positively. ⊣

Theorem 2.3. *Assume* $\Gamma \vdash^{\mathcal{I}} A$ *where* Γ, A *are arithmetic formulas and* $\mathcal{I}$ *consists of arithmetic predicates. Then* $\Gamma \vdash_1^{\mathcal{I}} A$.

Proof. Let us (just for this proof) call a derivation 'admissible' if it uses only arithmetic instances of the rules $\mathsf{Comp}_{\lambda x A}$, Efq_A and $\mathsf{Ind}_{\lambda n A}$. Since arbitrary instances of Comp and Efq can be derived (without induction) from arithmetic (even atomic) instances of these rules we may well assume that our given derivation of $\Gamma \vdash^{\mathcal{I}} A$ is admissible. Using normalization for second-order logic (due to Girard [10]) we can, by repeatedly applying the usual β-conversions for second-order natural deduction, transform the given derivation into a derivation $\Gamma \vdash^{\mathcal{I}} M \colon A$ which is still admissible and which in addition is 'normal' in the sense that in a $*$-elimination, $* \in \{\wedge, \to, \forall, \exists, \{\forall\}, \{\exists\}, \forall_2\}$, the main premise deriving the formula with $*$ as outermost constructor is not a $*$-introduction. By lemma 2.2, β-conversion does not spoil the special side conditions for the uniform and second-order rules and clearly β-conversion also preserves admissibility. Now, by a straightforward induction on normal admissible derivations $\Delta \vdash N\colon B$ one simultaneously shows (1) if Δ, B are arithmetic, then N is first-order; (2) if Δ is arithmetic and N does not end with a $*$-introduction where $*$ is as above, then B is arithmetic and N is first-order. Applying (1) to M we see that M is first-order. $\dashv$

§3. Realizability and program extraction. To every first-order non-Harrop formula A we assign a simple type $\tau(A)$, which will be the type of programs extracted from derivations with end formula A: $\tau(A_0 \wedge A_1) = \tau(A_0) \times \tau(A_1)$ if both conjuncts are non-Harrop and $= \tau(A_{1-i})$ if A_i is Harrop. $\tau(A \to B) = \tau(A) \to \tau(B)$ if A is non-Harrop and $= \tau(B)$ if A is Harrop. $\tau(\forall x^\rho\, A) = \rho \to \tau(A)$. $\tau(\exists x^\rho\, A) = \rho \times \tau(A)$ if A is non-Harrop and $= \rho$ if A is Harrop. Finally, $\tau(\{\forall x\}A) = \tau(\{\exists x\}A) = \tau(A)$. If $\mathcal{B}$ is a set of formulas we set $\tau(\mathcal{B}) = \{\tau(B) \mid B \in \mathcal{B}, B \text{ non-Harrop}\}$.

For every first-order non-Harrop formula A and every term r of type $\tau(A)$ we define by structural recursion on A a formula $r\,\mathbf{mr}\,A$, to be read as r *modified realizes* A. Simultaneously we define for every first-order Harrop formula A a formula $\epsilon\,\mathbf{mr}\,A$ where ϵ should be thought of as the inhabitant of a singleton type. Our definition is taken from [2] and corresponds to Kreisel's modified realizability [13] (see also [19]). For non-Harrop formulas we define

$$\begin{aligned}
r\,\mathbf{mr}\,(A_0 \wedge A_1) &= \begin{cases} \mathsf{p}_0(r)\,\mathbf{mr}\,A_0 \wedge \mathsf{p}_1(r)\,\mathbf{mr}\,A_1 & \text{if } A_0,\ A_1 \text{ are non-Harrop}\\ \epsilon\,\mathbf{mr}\,A_i \wedge r\,\mathbf{mr}\,A_{1-i} & \text{if } A_i \text{ is Harrop}\end{cases}\\
r\,\mathbf{mr}\,(A \to B) &= \begin{cases} \forall x\,.x\,\mathbf{mr}\,A \to rx\,\mathbf{mr}\,B & \text{if } A \text{ is non-Harrop}\\ \epsilon\,\mathbf{mr}\,A \to r\,\mathbf{mr}\,B & \text{if } A \text{ is Harrop}\end{cases}\\
r\,\mathbf{mr}\,\forall x^\rho\, A &= \forall x\,.rx\,\mathbf{mr}\,A\\
r\,\mathbf{mr}\,\exists x^\rho\, A &= \begin{cases} \mathsf{p}_1(r)\,\mathbf{mr}\,A[\mathsf{p}_0(r)/x] & \text{if } A \text{ is non-Harrop}\\ \epsilon\,\mathbf{mr}\,A[r/x] & \text{if } A \text{ is Harrop}\end{cases}\\
r\,\mathbf{mr}\,\mathsf{Q}A &= \mathsf{Q}(r\,\mathbf{mr}\,A) \text{ where } \mathsf{Q} \in \{\{\forall x\},\{\exists x\}\}
\end{aligned}$$

and for Harrop formulas

$$\begin{aligned}
\epsilon\,\mathbf{mr}\,(A) &= A \text{ if } A \text{ is atomic}\\
\epsilon\,\mathbf{mr}\,(A_0 \wedge A_1) &= \epsilon\,\mathbf{mr}\,A_0 \wedge \epsilon\,\mathbf{mr}\,A_1\\
\epsilon\,\mathbf{mr}\,(A \to B) &= \begin{cases} \forall x\,.x\,\mathbf{mr}\,A \to \epsilon\,\mathbf{mr}\,B & \text{if } A \text{ is non-Harrop}\\ \epsilon\,\mathbf{mr}\,A \to \epsilon\,\mathbf{mr}\,B & \text{if } A \text{ is Harrop}\end{cases}\\
\epsilon\,\mathbf{mr}\,\mathsf{Q}A &= \mathsf{Q}(\epsilon\,\mathbf{mr}\,A) \text{ where } \mathsf{Q} \in \{\forall x\,,\{\forall x\},\{\exists x\}\}
\end{aligned}$$

To every first-order derivation M of a non-Harrop formula A we assign a simply typed lambda-term $[\![M]\!]$, the *program extracted from* M. The assignment is relative to a given one-to-one assignment of assumption variables u^B, where B is non-Harrop, to object variables $x_u^{\tau(B)}$.

The definition of $[\![M]\!]$ for derivations M ending with a normal rule is as expected and is shown in figure 3. We write '$r \equiv s$' to express that r and s are syntactically identical (so no confusion with the formula '$r = s$' is possible). For uniform rules $[\![M]\!]$ is very simple and shown in figure 4. For convenience we set $[\![M]\!] := \epsilon$ if M derives a Harrop formula.

$[\![u^A]\!] \equiv x_u$		
$[\![\wedge^+(M_0^{A_0}, M_1^{A_1})]\!]$ $\equiv \mathsf{p}([\![M_0]\!], [\![M_1]\!])$ $\equiv [\![M_{1-i}]\!]$	$[\![\wedge_i^-(M^{A_0 \wedge A_1})]\!]$ $\equiv \mathsf{p}_i([\![M]\!])$ $\equiv [\![M]\!]$	A_0, A_1 non-Harrop A_i Harrop
$[\![\to^+(\lambda u^A M)]\!]$ $\equiv \lambda x_u.[\![M]\!]$ $\equiv [\![M]\!]$	$[\![\to^-(M^{A \to B}, N^A)]\!]$ $\equiv [\![M]\!][\![N]\!]$ $\equiv [\![M]\!]$	A non-Harrop A Harrop
$[\![\forall^+(\lambda x M)]\!] \equiv \lambda x.[\![M]\!]$	$[\![\forall^-(M, t)]\!] \equiv [\![M]\!]t$	
$[\![\exists^+_{\lambda x A}(t, M)]\!]$ $\equiv \mathsf{p}(t, [\![M]\!])$ $\equiv t$	$[\![\exists^-(M^{\exists x\, A}, N)]\!]$ $\equiv [\![N]\!]\mathsf{p}_0([\![M]\!])\mathsf{p}_1([\![M]\!])$ $\equiv [\![N]\!][\![M]\!]$	A non-Harrop A Harrop
$[\![\mathsf{Case}_{\lambda p A}(M, N)]\!]$ $\equiv \lambda p.\mathsf{If}_{\tau(\mathsf{A})} p [\![M]\!][\![N]\!]$	$[\![\mathsf{Ind}_{\lambda n A}(M, N)]\!]$ $\equiv \mathsf{Rec}_{\tau(\mathsf{A})}[\![M]\!][\![N]\!]$	
$[\![\mathsf{Comp}_{\lambda x A}(M, N)]\!]$ $\equiv [\![M]\!]$	$[\![\mathsf{Efq}_A(M)]\!] \equiv 0^{\tau(A)}$ (any term of type $\tau(A)$)	

FIGURE 3. Program extraction for the normal rules of HA^{u}

In the following when we write '$x_u \,\mathbf{mr}\, B$' where B is a Harrop formula we mean in fact '$\epsilon \,\mathbf{mr}\, B$'. If $\mathcal{P}$ is a predicate of arity (nat), then we set $\mathbf{mr}(\mathcal{P})$ to

$[\![\{\forall\}^+(\lambda x M)]\!] \equiv [\![M]\!]$	$[\![\{\forall\}^-(M,t)]\!] \equiv [\![M]\!]$
$[\![\{\exists\}^+_{\lambda x A}(t,M)]\!] \equiv [\![M]\!]$	$[\![\{\exists\}^-(M^{\exists x\, A},N)]\!]$ $\equiv [\![N]\!][\![M]\!]$ if A is non-Harrop $\equiv [\![N]\!]$ if A is Harrop

FIGURE 4. Program extraction for the uniform rules of HA^{u}

be $\lambda n.fn\,\mathbf{mr}\,\mathcal{P}(n)$ where f is a fresh variable of appropriate type, or $\epsilon\,\mathbf{mr}\,\mathcal{P}(n)$ depending on whether $\mathcal{P}$ is non-Harrop or Harrop.

THEOREM 3.1 (Soundness of realizability). *Assume* $\Gamma \vdash^{\mathcal{I}}_1 M\colon A$. *Then we have* $\{x_u\,\mathbf{mr}\,B \mid u^B \in \Gamma\} \vdash^{\mathbf{mr}(\mathcal{I})}_1 [\![M]\!]\,\mathbf{mr}\,A$, *with* $\mathsf{reclev}([\![M]\!]) \leq \mathsf{lev}(\tau(\mathcal{I}))$ *if A is non-Harrop.*

PROOF. Induction on M. ⊣

Remark. Of course $\tau(\mathcal{I})$ denotes $\{\tau(B) \mid \lambda nB \in \mathcal{I},\, B \text{ non-Harrop}\} \cup \{\rho \mid \rho \in \mathcal{I} \text{ type}\}$. A similar remark applies to $\mathbf{mr}(\mathcal{I})$.

In order to use theorem 3.1 for program extraction we need to relate realizability to truth. We call a first-order formula A *Q-negative* respectively *Q-positive* where $Q \in \{\forall, \exists\}$, if A contains the quantifier Q at negative respectively positive positions only. Note that $\exists$-negative formulas A are Harrop.

LEMMA 3.2. *Let A be a first-order formula.*
(i) *If A is $\exists$-negative, then* $\vdash^{\emptyset}_1 A \to \epsilon\,\mathbf{mr}\,A$.
(ii) *If A is $\exists$-positive, then* $\vdash^{\emptyset}_1 x\,\mathbf{mr}\,A \to A$.

PROOF. Straightforward simultaneous induction on A. ⊣

THEOREM 3.3 (Program extraction). *Let Γ be a set of $\exists$-negative formulas and A an $\exists$-positive formula. Assume* $\Gamma \vdash^{\mathcal{I}}_1 M\colon \exists x^{\rho} A$. *Let* $r \equiv [\![M]\!]$ *if A is a Harrop formula and* $r \equiv \mathsf{p}_0([\![M]\!])$ *otherwise. Then* $\Gamma \vdash^{\mathbf{mr}(\mathcal{I})}_1 A[r/x]$ *with* $\mathsf{reclev}(r) \leq \mathsf{lev}(\tau(\mathcal{I}))$.

PROOF. Theorem 3.1 and lemma 3.2. ⊣

Remarks. 1. Our modified realizability interpretation could be easily extended to a similar system that allows comprehension for *arbitrary* (i.p. possibly second-order) Harrop predicates. The second-order quantifier $\forall_2$ could be treated like a uniform quantifier, that is, we would set $\tau(\forall_2 X\, A) = \tau(A)$, $r\,\mathbf{mr}\,\forall_2 X\, A = \forall_2 X\, r\,\mathbf{mr}\,A$ and $[\![\forall_2^+(\lambda X\, M)]\!] \equiv [\![M]\!]$, $[\![\forall_2^-(M,\mathcal{P})]\!] \equiv [\![M]\!]$. Essentially, the interpretation works because the type of a formula is not changed by substituting a Harrop predicate into it.

2. Consider 'uniform induction' which is like induction except for the step formula which becomes $\{\forall n\}.\, A \to A[n+1/n]$ (note that uniform induction is

weaker than induction). Obviously, uniform induction is realized by the 'iterator' $\mathsf{It}_\rho^{\rho\to(\rho\to\rho)\to\mathsf{nat}\to\rho}$ with the conversion rules $\mathsf{It}_\rho\, r\, s\, 0 \mapsto r$ and $\mathsf{It}_\rho r\, s\,(t+1) \mapsto s\,(\mathsf{It}_\rho\, r\, s\, t)$. One can derive induction from uniform induction and extract from that an implementation of primitive recursion in terms of iteration. Uniform induction also suffices to prove the classical least element principle, $\neg\neg\exists n\, A \to \neg\neg\exists n\,(A \wedge \forall m < n\, \neg A[m/n])$.

In section 7 we will be concerned with the problem of constructing a realizer of an instance of induction on a predicate of the form $\lambda n(A \vee B)$ where n is not free in B. According to the soundness theorem 3.1 we immediately get a realizer that uses a primitive recursion of type $\mathsf{boole} \times \tau(A) \times \tau(B)$ and hence has recursion level $= \max(\mathsf{lev}(\tau(A)), \mathsf{lev}(\tau(B)))$ (assuming A and B are both non-Harrop). We now show that the recursion level can be lowered to $\mathsf{lev}(\tau(A))$.

In the following we set $\mathsf{Ind}(\mathcal{P}) := \mathcal{P}(0) \wedge \forall n\,(\mathcal{P}(n) \to \mathcal{P}(n+1)) \to \forall n\,\mathcal{P}(n)$ for any predicate $\mathcal{P}$ of arity nat.

LEMMA 3.4. *Let A, B be first-order formulas where A is non-Harrop and $n\colon \mathsf{nat}$ a variable not free in B. Let $f, g\colon \mathsf{nat} \to \mathsf{boole}$ be fresh variables and h a fresh variable of type $\mathsf{nat} \to \tau(A)$. Then*

$$\vdash_1^{\{\lambda n.(fn=\mathsf{True}\to hn\,\mathbf{mr}\,A)\wedge gn=\mathsf{True}\}}\; r\,\mathbf{mr}\,\mathsf{Ind}(\lambda n.A \vee B)$$

for some term r with $\mathsf{reclev}(r) = \mathsf{lev}(\tau(A))$.

PROOF. Let us assume B is non-Harrop as well (the other case is similar). Let $\tau(A) = \rho$ and $\tau(B) = \sigma$. We argue informally. To produce a realizer of $\mathsf{Ind}(\lambda n.A \vee B)$ we assume $(p_0, y_0, z_0)\,\mathbf{mr}\,(A[0/n] \vee B)$, that is,

$$(p_0 = \mathsf{True} \to y_0\,\mathbf{mr}\,A[0/n]) \wedge (p_0 = \mathsf{False} \to z_0\,\mathbf{mr}\,B),$$

and $s\,\mathbf{mr}\,\forall n\,(A \vee B \to A[n+1/n] \vee B)$, that is,

$$\forall n\,((p, y, z)\,\mathbf{mr}\,(A \vee B) \to sn(p, y, z)\,\mathbf{mr}\,(A[n+1/n] \vee B)).$$

We need to realize $\forall n\,(A \vee B)$. If $p_0 = \mathsf{False}$, then we know $z_0\,\mathbf{mr}\,B$ and hence $\lambda n(\mathsf{False}, 0^\rho, z_0)\,\mathbf{mr}\,\forall n\,(A \vee B)$. If $p_0 = \mathsf{True}$ we define a sequence of triples $t_n = (p_n, y_n, k_n)\colon \mathsf{boole} \times \rho \times \mathsf{nat}$ such that

(1) if $p_n = \mathsf{True}$ then $y_n\,\mathbf{mr}\,A$,
(2) if $p_n = \mathsf{False}$ then $p_{k_n} = \mathsf{True}$ and $\mathsf{p}_0(sk_n(\mathsf{True}, y_{k_n}, 0^\sigma)) = \mathsf{False}$

(where $\mathsf{p}_i(a_0, a_1, a_2) = a_i$). Once we have defined the sequence $(t_n)_{n\in\mathbb{N}}$ we are done since then $\lambda n(p_n, y_n, \mathsf{p}_2(sk_n(\mathsf{True}, y_{k_n}, 0^\sigma))\,\mathbf{mr}\,\forall n\,(A \vee B)$

We define the triples t_n by primitive recursion on n as follows (note that $\mathsf{lev}(\mathsf{boole} \times \rho \times \mathsf{nat}) = \mathsf{lev}(\rho)$). $t_0 := (p_0, y_0, 0)$ where p_0, y_0 are given above. In order to define t_{n+1} from t_n we consider p_n. If $p_n = \mathsf{False}$ we set $t_{n+1} :\equiv t_n$. If $p_n = \mathsf{True}$ we compute $(p, y, z) :\equiv sn(\mathsf{True}, y_n, 0^\sigma)$. If $p = \mathsf{True}$ we set $t_{n+1} :\equiv (\mathsf{True}, y, 0)$. If $p = \mathsf{False}$ we set $t_{n+1} :\equiv (\mathsf{False}, 0^\rho, n)$.

By induction on n one verifies that the triples t_n satisfy (1) and (2). Clearly the conjunction of (1) and (2) is equivalent (provable without induction) to an instance of the formula $(fn = \mathsf{True} \to hn\,\mathbf{mr}\,A) \wedge gn = \mathsf{True}$, as required. ⊣

Remark. A similar lemma and a similar construction appear in [9] in the context of bounded arithmetic.

§4. Minimal fragments of HA^ω and partial Gödel translations. One of the main goals of this paper is to extract optimized programs from classical proofs of first-order existential formulas. The extraction process can be roughly divided into three phases. First the given classical proof is transformed via Gödel's negative translation (double negating atomic and existential formulas) into a proof in minimal logic of a formula of the form $\neg\neg\exists x\, B$. Then a Kripke translation further transforms this into an intuitionistic proof of $\exists x\, B$ from which we finally extract a program via the realizability interpretation described in section 3. In this paper we do not carry out the first phase, but rather assume that a classical proof of an existence formula is given to us as proof of a formula $\neg\neg\exists x\, B$ in the minimal first-order fragment of HA^ω.

In general the minimal fragment of an intuitionistic system is obtained by restricting it such that the formula $\bot$ can be treated as a placeholder for an arbitrary formula. In order to achieve this for our system it is not sufficient to abandon the rule Efq. We must also restrict the $\{\forall\}$-introduction in order to prevent the side condition '$x \notin \mathsf{CV}(M)$' from becoming violated when $\bot$ is replaced by an arbitrary formula.

We call a first-order formula a *strong Harrop formula* if it contains neither the quantifier $\exists$ nor the formula $\bot$ in a strictly positive position. Obviously if A is a strong Harrop formula, then $A[B/\bot]$ is a strong Harrop formula for arbitrary first-order formulas B.

We define $\mathsf{PCV}(M)$ (potentially computationally relevant variables) like $\mathsf{CV}(M)$, but replacing in the definition 'Harrop formula' by 'strong Harrop formula'. We write

$$\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} M \colon A$$

if $\Gamma \vdash^{\mathcal{I}} M \colon A$ and

(1) Γ, A are first-order,
(2) the rule Efq is not used in M,
(3) for every occurrence of a rule $\{\forall\}^+(\lambda y N)$ we have $y \notin \mathsf{PCV}(N)$.

For any given set P of predicate constants we define a partial form of Gödel's double negation translation by

$$\mathsf{g_P}(A) = A[\lambda\vec{x}\neg\neg P(\vec{x})/P \mid P \in \mathsf{P}].$$

Lemma 4.1. (i) *Let* P *be a set of predicate constants and* A *a first-order formula such that all predicate constants occurring strictly positively in* A *are in* P*, then* $\vdash^{\emptyset}_{\mathrm{m}} \neg\neg\mathsf{g_P}(A) \to \mathsf{g_P}(A)$ *(and hence also* $\vdash^{\emptyset}_{\mathrm{m}} \bot \to \mathsf{g_P}(A)$*).*

(ii) *If* A *contains neither implications nor universal quantifiers, then we have* $\vdash^{\emptyset}_{\mathrm{m}} \neg\mathsf{g_P}(A) \leftrightarrow \neg A$ *(with respect to any* P*).*

(iii) *If* $A \in \mathsf{qf}(\mathsf{P})$*, then* $\mathsf{DEC}(\mathsf{P}) \vdash^{\emptyset}_{\mathrm{m}} \mathsf{g_P}(A) \leftrightarrow \neg\neg(t_A = \mathsf{True})$ *(and therefore also* $\mathsf{DEC}(\mathsf{P}) \vdash^{\emptyset}_{\mathrm{m}} \mathsf{g_P}(\neg\neg\exists x\, A) \leftrightarrow \neg\neg\exists x\, t_A = \mathsf{True}$*).*

Proof. Inductions on A. In (iii) one uses (i). $\dashv$

Unfortunately a partial Gödel translation can be unsound for the rule Comp and for uniform rules. We therefore we need to further restrict the relation $\vdash_{\mathrm{m}}$. We write

$$\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}_0} M \colon A$$

if $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} M \colon A$ and in M neither the rule Comp nor a uniform rule occur.

LEMMA 4.2. *If* $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}_0} M \colon A$*, then* $\mathsf{g_P}(\Gamma) \vdash^{\mathsf{g_P}(\mathcal{I})}_{\mathrm{m}_0} \mathsf{g_P}(A)$.

PROOF. Straightforward induction on M. ⊣

Remark. Abandoning the rule Comp might seem to result in a rather useless calculus. However we may add corresponding assumptions, e.g. $\forall(r = s \to A[r/x] \to A[s/x])$ (A atomic suffices) to the assumptions Γ. A similar remark applies to the rule Efq. This means that in fact we still have full intuitionistic logic available. The point of $\vdash_{\mathrm{m}_0}$ is that according to the lemma above these assumptions need to be Gödel translated.

§5. A Kripke-translation.

For every first-order formula A and every arithmetic formula K we define a formula A^K.

$$\begin{aligned}
\bot^K &= K\\
P(\vec{t})^K &= P(\vec{t}) \quad (P \text{ a predicate constant})\\
(A \wedge B)^K &= A^K \wedge B^K\\
(A \to B)^K &= \forall_2 Y\,.\,(K \to Y) \to A^Y \to B^Y \quad (Y \text{ a fresh prop. var.})\\
(\mathsf{Q}A)^K &= \mathsf{Q}A^K \quad (\mathsf{Q} \text{ any of the four first-order quantifiers})
\end{aligned}$$

For the remainder of this section A and B range over first-order and K over arithmetic formulas. Throughout the rest of the paper X, Y, Z stand for sufficiently fresh propositional variables.

The obvious proofs of the next two lemmas are omitted.

LEMMA 5.1. (i) $A^X[K/X] = A^K$.
(ii) $A[t/x]^K = A^K[t/x]$.
(iii) *If A is strongly Harrop then A^K is Harrop.*

LEMMA 5.2. (i) $\vdash^{\emptyset} (K \to L) \to A^K \to A^L$*, for any A.*
(ii) *If C is $\bot$-free, i.e. $\bot$ does not occur in C, then* $\vdash^{\emptyset} C^K \leftrightarrow C$.

THEOREM 5.3 (Soundness of Kripke translation). *If* $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} A$ *then* $\Gamma^K \vdash^{\mathcal{I}^Y} A^K$.

PROOF. Note that, by definition of $\vdash_{\mathrm{m}}$, Γ and A are first-order, and therefore, by theorem 2.3 we may assume that the given derivation of $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} A$ is first-order. We argue by induction on derivations using lemma 5.1 and lemma 5.2. For every derivation $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} M \colon A$ one constructs a derivation $\Gamma^K \vdash^{\mathcal{I}^Y} M' \colon A^K$ such that $\mathsf{CV}(M') \subseteq \mathsf{PCV}(M)$. We only sketch some of the more interesting cases.

$\to^+$: By induction hypothesis we have $\Gamma^Z, A^Z \vdash^{\mathcal{I}^Y} B^Z$. With lemma 5.2 (i) it follows $\Gamma^K, K \to Z, A^Z \vdash^{\mathcal{I}^Y} B^Z$. Hence $\Gamma^K \vdash^{\mathcal{I}^Y} (A \to B)^K$.

$\forall^-$: We are given $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} \forall^-(M,t) : A[t/x]$ derived from $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} M : \forall x\, A$. By induction hypothesis we have $\Gamma^K \vdash^{\mathcal{I}^Y} M' : \forall x\, A^K$ with $\mathsf{PCV}(M') = \mathsf{PCV}(M)$. With lemma 5.1 (ii) it follows $\Gamma^K \vdash^{\mathcal{I}^Y} \forall^-(M',t) : A[t/x]^K$. By virtue of lemma 5.1 (iii) we also have $\mathsf{CV}(\forall^-(M',t)) \subseteq \mathsf{PCV}(\forall^-(M,t))$.

Induction: By (meta) induction hypothesis we have $\Gamma^K \vdash^{\mathcal{I}^Y} A^K[0/n]$ and $\Gamma^K \vdash^{\mathcal{I}^Y} \forall n\, \forall_2 Y\, .\, (K \to Y) \to A^Y \to A[n{+}1/n]^Y$ with $A \in \mathcal{I}$. With lemma 5.1 (i) it follows $\Gamma^K \vdash^{\mathcal{I}^Y} \forall n\, (A^K \to A^K[n+1/n])$ because K is arithmetic. This entails $\Gamma^K \vdash^{\mathcal{I}^Y} \forall n\, A^K$, because $\lambda n A^K$ is an arithmetic instance of $\mathcal{I}^Y$.

$\{\forall\}^+$: We assume we derived $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} \{\forall\}^+(\lambda y M) \colon \{\forall y\} A$ from $\Gamma \vdash^{\mathcal{I}}_{\mathrm{m}} M \colon A$ where $y \notin \mathsf{FV}(\Gamma) \cup \mathsf{PCV}(M)$. By i.h. we have $\Gamma^K \vdash^{\mathcal{I}^Y} M' \colon A^K$ and $y \notin \mathsf{FV}(\Gamma^K) \cup \mathsf{CV}(M')$. Hence $\Gamma^K \vdash^{\mathcal{I}^Y} \{\forall\}^+(\lambda y M') \colon \{\forall y\} A^K$. ⊣

The following lemma shows how the Kripke translation can be used to remove double negation.

Lemma 5.4. (i) $\forall_2 Y\, ((B \to Y) \to B^Y \to Y) \vdash^{\emptyset} (\neg\neg B)^{\perp} \to B$.
(ii) *If B is $\perp$-free then* $\vdash^{\emptyset} (\neg\neg B)^K \leftrightarrow (B \vee K)$.

Proof. We argue informally. (i): Assume $\forall_2 Y\, ((B \to Y) \to B^Y \to Y)$ and $(\neg\neg B)^{\perp}$. The second assumption is intuitionistically equivalent to

$$\forall_2 X\, .\, \forall_2 Y\, ((X \to Y) \to B^Y \to Y) \to X.$$

Since B is arithmetic we may instantiate X with it and obtain

$$\forall_2 Y\, ((B \to Y) \to B^Y \to Y) \to B$$

which yields B by the first assumptions.

(ii): Since B doesn't contain $\perp$, we have, by lemma 5.2(ii), $\vdash^{\emptyset} (\neg B)^X \leftrightarrow B \to X$ and therefore

$$\vdash^{\emptyset} (\neg\neg B)^K \leftrightarrow \forall_2 X\, ((K \to X) \to (B \to X) \to X).$$

But $\vdash^{\emptyset} \forall_2 X\, ((K \to X) \to (B \to X) \to X) \leftrightarrow (B \vee K)$ since $B \vee K$ is arithmetic. ⊣

§6. Definite and goal formulas.

In [6] a translation similar to our Kripke translation and a class of 'goal formulas' were defined for which a property similar to the general hypothesis of lemma 5.4 is provable. It turns out that the same (even slightly extended) class can be used here.

We call a formula *relevant* if $\perp$ is the only atomic subformula in a strictly positive position, and *co-relevant* if $\perp$ does not occur in a strictly positive position (the terminology is inspired from [6]; note that a strong Harrop formula is the same as a first-order co-relevant Harrop formula). Relatively to a given set P of predicate constants we simultaneously define *definite formulas* D and *goal formulas* G. We let D_0 respectively G_0 range over $\perp$-free formulas that are $\exists$-negative respectively $\exists$-positive. We also use $\Box$ as a placeholder for the first-order universal quantifiers $\forall x$, $\{\forall x\}$ and $\Diamond$ for the existential quantifiers $\exists x$, $\{\exists x\}$.

$$\begin{aligned} D := {} & D_0 \mid \bot \mid D \wedge D \mid \Box D \\ & \mid G \to D \text{ provided } G \text{ co-relevant or } D \text{ relevant} \end{aligned}$$

$$\begin{aligned} G := {} & G_0 \mid \bot \mid G \wedge G \mid \Diamond G \\ & \mid \Box G \text{ provided } G \text{ co-relevant} \\ & \mid D \to G \text{ provided } D \text{ relevant or } D \in \mathsf{qf}(\mathsf{P}) \end{aligned}$$

Since this definition depends on the set P we will sometimes attach the prefix 'P-' to the words 'definite' and 'goal'. Note that definite respectively goal formulas are $\exists$-negative respectively $\exists$-positive.

LEMMA 6.1. *Set $\Delta := \mathsf{LEM}(\mathsf{P})$ and let D and G range over P-definite and P-goal formulas respectively. Then from $\mathsf{LEM}(\mathsf{P})$ we can derive without induction:*
(i) $(\neg D \to K) \to D^K$ *for D relevant.*
(ii) $D \to D^K$.
(iii) $G^K \to G$ *for G co-relevant.*
(iv) $(G \to K) \to G^K \to K$.

PROOF. The four claims are proved simultaneously by induction on the built-up of formulas. The proof is similar to the corresponding proof in [6]. Since the definitions have slightly changed we sketch the arguments anyway.

(i). The case D_0 does not apply since any relevant formula must contain $\bot$ at least once. For $\bot$ the assertion holds trivially, since $\bot^K = K$.

Case $D_1 \wedge D_2$. Since $D_1 \wedge D_2$ is relevant so are D_1 and D_2. Therefore the assertion follows directly from the induction hypothesis (i).

Cases $\Box D$. Assume $\neg\Box D \to K$. In order to prove $\Box D^K$ it suffices to prove D^K (using an introduction rule). Since $\Box D \to D$ our assumption entails $\neg D \to K$ and we can apply the induction hypothesis (i).

Case $G \to D$. Since $G \to D$ is relevant, so is D. Assume $\neg(G \to D) \to K$, $K \to Y$ and G^Y. We have to show D^Y. By i.h. (i) it suffices to show Y under the extra assumption $\neg D$. By i.h. (iv) and the assumption G^Y, showing Y reduces to showing $G \to Y$: From G and $\neg D$ we obtain $\neg(G \to D)$ and hence K, by our first assumption. Since $K \to Y$ it follows Y.

(ii). For $\bot$-free formula we use lemma 5.2 (ii). To derive $\bot \to \bot^K$ we use the rule Efq. The cases $D_1 \wedge D_2$ and $\Box D$ follow directly from i.h. (ii).

Case $G \to D$ where G is co-relevant or D is relevant. Assume $G \to D$ and G^Y. We have to show D^Y. If G is co-relevant we use i.h. (iii) and i.h. (ii). If D is relevant we use i.h. (i) to reduce our goal to Y under the extra assumption $\neg D$. Now i.h. (iv) and G^Y allow us to add the further assumption G. Since the assumptions G, $G \to D$ and $\neg D$ are contradictory, we are done with Efq.

(iii). For a $\bot$-free formula we use again lemma 5.2 (ii). The case $\bot$ does not apply. The cases $G_1 \wedge G_2$, $\Diamond$ and $\Box D$ follow directly from i.h. (iii). The case $D \to G$ follows easily from i.h. (ii) and i.h. (iii).

(iv). The cases G_0 and $\bot$ are easy. The cases $G_1 \wedge G_2$ and $\Diamond G$ follow directly from i.h. (iv).

Case $\Box G$ where G is co-relevant. Direct by i.h. (iii).

Case $D \to G$ where D is relevant or $\in \mathsf{qf}(\mathsf{P})$. Assume $(D \to G) \to K$ and $(D \to G)^K$. We have to show K. From the first assumption we obtain $\neg D \to K$ (+) using Efq. We also have $D^K \to K$ (++) using both assumptions and i.h. (iv). If D is relevant we are done by i.h. (i), (+) and (++). If $D \in \mathsf{qf}(\mathsf{P})$ we argue by case analysis on D (using lemma 2.1)). In case D we use i.h. (ii) and (++), whereas in case $\neg D$ we are done by (+). ⊣

THEOREM 6.2. *Let Γ be a set of of definite formulas and G a goal formula. Then $\Gamma \vdash_{\mathrm{m}}^{\mathcal{I}} \neg\neg G$ implies $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash^{\mathcal{I}^X} G$.*

PROOF. By theorem 5.3 we have $\Gamma^{\perp} \vdash^{\mathcal{I}^X} (\neg\neg G)^{\perp}$. By lemma 6.1 (ii) we may replace $\Gamma^{\perp}$ by $\mathsf{LEM}(\mathsf{P}) \cup \Gamma$. By lemma 5.4 (i) and lemma 6.1 (iv) we may replace $(\neg\neg G)^{\perp}$ by G. ⊣

THEOREM 6.3. *Let Γ, G be as in theorem 6.2 and $\mathcal{I}$ a set of $\perp$-free first-order predicates. Then $\Gamma \vdash_{\mathrm{m}}^{\neg\neg\mathcal{I}} \neg\neg G$ implies $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash_1^{\mathcal{I} \vee X} G$.*

PROOF. By theorem 6.2 the assumption implies $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash^{(\neg\neg\mathcal{I})^X} G$. With lemma 5.4 (ii) and theorem 2.3 the assertion follows. ⊣

Note that in $\mathcal{I} \vee X$ above the variable X may be instantiated in the derivation by an arbitrary arithmetic predicate. The following theorem shows that under certain circumstances the '$\vee X$' may be omitted.

THEOREM 6.4. *Let Γ, G, $\mathcal{I}$ be as in theorem 6.2 Assume $\Gamma \vdash_{\mathrm{m}}^{\neg\neg\mathcal{I}} \neg\neg G$ where in the derivation for every occurrence of an induction, $\mathsf{Ind}_{\lambda n \neg\neg B}(M, N)$, it holds that in M and N only assumptions from Γ and no inductions are used. Then $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash_1^{\mathcal{I}} G$.*

PROOF. The assumptions imply that for every instance of induction over a predicate $\lambda n \neg\neg B_i$ we have in fact $\Gamma \vdash_{\mathrm{m}}^{\emptyset} \neg\neg B_i[0/n]$, $\Gamma \vdash_{\mathrm{m}}^{\emptyset} \forall n\,.\neg\neg B_i \to \neg\neg B_i[n+1/n]$ and $\Gamma, \wedge_i \forall n\, B_i \vdash_{\mathrm{m}}^{\emptyset} \neg\neg G$. By theorem 5.3 together with lemma 5.4 (ii) and lemma 6.1 (ii) we obtain from the first two groups of derivations $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash^{\emptyset} B_i[0/n]$ and $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash^{\emptyset} \forall n\,.\, B_i \to B_i[n+1/n]$ which together with the third derivation and induction imply $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash^{\mathcal{I}} G$. With theorem 2.3 we obtain first-order derivability. ⊣

§7. Programs from classical proofs.

We now combine the Kripke-translation and the realizability interpretation to extract programs from classical first-order proofs. Since, as explained in section 4, we view classical logic as a subsystem of of minimal logic, classical proofs of existential formulas are subsumed by proofs of the form $\Gamma \vdash_{\mathrm{m}} \neg\neg\exists x\, B$. In the formulations of our results we will give bounds for levels of recursions occurring in extracted programs and state for which predicate constants decision procedures are needed. We define the level of a formula, $\mathsf{lev}(A)$, recursively as follows. $\mathsf{lev}(A) = 0$ if A is an atomic formula, otherwise $\mathsf{lev}(A \wedge B) = \max(\mathsf{lev}(A), \mathsf{lev}(B))$, $\mathsf{lev}(A \to B) = \max(\mathsf{lev}(A) + 1, \mathsf{lev}(B))$, $\mathsf{lev}(\forall x^{\rho} A) = \max(\mathsf{lev}(\rho) + 1, \mathsf{lev}(A))$, $\mathsf{lev}(\exists x^{\rho} A) = \max(\mathsf{lev}(\rho), \mathsf{lev}(A))$, $\mathsf{lev}(\{\forall x\}A) = \mathsf{lev}(\{\exists x\}A) = \mathsf{lev}(A)$.

THEOREM 7.1. *Let* P *be a set of predicate constants and* Γ *a set of* P*-definite formulas and* G *a* P*-goal formula. Assume*

$$\Gamma \vdash_{\mathrm{m}}^{\neg\neg\mathcal{I}} M \colon \neg\neg\exists x^{\rho}\, G$$

where all predicates in $\mathcal{I}$ *are* $\bot$*-free and all predicates and types in* $\mathcal{I}$ *have level* $\leq k$*. Then one can extract from* M *a term* r *with* $\mathsf{reclev}(r) \leq k$ *such that*

$$\mathsf{DEC}(\mathsf{P}), \Gamma \vdash_{1}^{\mathcal{J}} G[r/x]$$

where $\mathcal{J}$ *consists of predicates of the form* $\lambda n.(E \to (s\,\mathbf{mr}\,C)) \wedge F$ *with equations* E, F *of type* boole *and* $\lambda n.C \in \mathcal{I}$*.*

PROOF. By theorem 6.3 we have $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash^{\mathcal{I} \vee X} \exists x\, G$. Lemma 2.1 (ii), lemma 3.4, lemma 3.2, theorem 3.3, lemma 2.1 (iii) and theorem 2.3 give us a term r with $\mathsf{reclev}(r) \leq k$ such that $\mathsf{DEC}(\mathsf{P}), \Gamma \vdash_{1}^{\mathcal{J}} G[r/x]$ where the predicates in $\mathcal{J}$ are of the required form. ⊣

Remark. If in the proof above one uses theorem 6.4 instead of theorem 6.3 (provided the additional conditions are satisfied, of course), then one does not need to refer to lemma 3.4 and hence obtains slightly improved programs (see the example of the Fibonacci numbers in section 8).

In order to make theorem 7.1 easier to apply it is desirable to replace the rather complicatedly defined classes of definite and goal formulas by simpler ones. For example, the class of definite formulas could be replaced by the smaller class of formulas containing $\bot$ only positively and $\diamond$ only negatively. However such a restriction on occurrences of $\bot$ excludes many natural situations. For example, in order to make an intuitionistic proof minimal one one might have to add assumptions of the form $\forall(\bot \to A)$ to Γ, which have $\bot$ at a negative position. We will now identify classes of formulas that allow unrestricted occurrences of $\bot$, are easier to recognize than definite and goal formulas, do not depend on a given class P of predicates and can easily be transformed into definite respectively goal formulas preserving provability.

We call a first-order formula *quasi-negative* if it is $\exists$-negative and $\forall$-positive, that is, contains existential quantifiers only negatively and universal quantifiers only positively. A formula is called *quasi-positive* if it is $\exists$-positive and $\forall$-negative.

For any first-order formula A we inductively define a set $\mathsf{Crit}(A)$ of so-called "A-critical" predicate symbols by the following two rules:

(i) Any predicate symbol occurring non-strictly positively (i.e. positively, but not strictly positively) in A is A-critical.
(ii) If $B \to C$ is a strictly positive subformula of A, and $\bot$ or an A-critical predicate symbol occur strictly positively in B, then all predicate symbols occurring strictly positively in C are A-critical.

LEMMA 7.2. *Let* A *be quasi-negative and* $\mathsf{P} := \mathsf{Crit}(A)$*. Then, with respect to* P*,* $\mathsf{g}_{\mathsf{P}}(C)$ *is definite for every positive subformula* C *of* A*, and* $\mathsf{g}_{\mathsf{P}}(B)$ *is a goal formula for every negative subformula* B *of* A*.*

Proof. Induction on subformulas of A.

Case $\bot$. This is a definite and a goal formula.

Case $P(\vec{t})$ positive. If $P \in \mathsf{P}$, then $P(\vec{t}) \in \mathsf{qf}(\mathsf{P})$. Hence $\neg P(\vec{t})$ is a goal formula and therefore $\mathsf{g}_{\mathsf{P}}(P(\vec{t})) = \neg\neg P(\vec{t})$ is definite. If $P \notin \mathsf{P}$, then $\mathsf{g}_{\mathsf{P}}(P(\vec{t})) = P(\vec{t})$ is definite as well.

Case $P(\vec{t})$ negative. The formulas $P(\vec{t})$ and $\neg\neg P(\vec{t})$ are both goal formulas (with respect to any P).

Case $B \wedge C$ (positive or negative). Direct by induction hypothesis.

Case $B \to C$ positive. Since $\mathsf{g}_{\mathsf{P}}(B \to C) = \mathsf{g}_{\mathsf{P}}(B) \to \mathsf{g}_{\mathsf{P}}(C)$, by induction hypothesis, it suffices to show that $\mathsf{g}_{\mathsf{P}}(B)$ is co-relevant or $\mathsf{g}_{\mathsf{P}}(C)$ is relevant. If $\mathsf{g}_{\mathsf{P}}(B)$ is not co-relevant, then $\bot$ or a critical P must occur in B strictly positively. But then all predicate symbols occurring strictly positively in C are critical, and therefore all strictly positive positions in C are occupied by critical predicate symbols. Hence $\mathsf{g}_{\mathsf{P}}(C)$ is relevant because $\mathsf{P} \supseteq \mathsf{Crit}(A)$.

Case $C \to B$ negative. By a similar argument as above $\mathsf{g}_{\mathsf{P}}(B)$ is relevant. Hence the induction hypotheses can be applied.

Case $\Box C$ positive, $\Diamond B$ negative. Direct by induction hypothesis. $\dashv$

For a set Γ of first-order formulas we set $\mathsf{Crit}(\Gamma) = \mathsf{Crit}(A)$ where A is the conjunction of all formulas in Γ.

Theorem 7.3. *Let Γ be a set of quasi-negative formulas, B a quasi-positive formula and assume*

$$\Gamma \vdash^{\neg\neg\mathcal{I}}_{\mathrm{m}_0} M \colon \neg\neg\exists x^\rho\, B$$

where the predicates in $\mathcal{I}$ are free of $\bot$, $\to$ and $\forall$ and have level $\leq k$. Then from M one can extract a term r with $\mathsf{lev}(r) \leq k$ such that

$$\mathsf{DEC}(\mathsf{Crit}(\Gamma, \neg B)), \Gamma \vdash^{\mathcal{J}}_{1} B[r/x]$$

where $\mathcal{J}$ consists of quantifier free predicates.

Proof. Let $\mathsf{P} = \mathsf{Crit}(\Gamma, \neg B)$. We have $\mathsf{g}_{\mathsf{P}}(\Gamma) \vdash^{\mathsf{g}_{\mathsf{P}}(\neg\neg\mathcal{I})}_{\mathrm{m}_0} \neg\neg\exists x^\rho\, \mathsf{g}_{\mathsf{P}}(B)$, by lemma 4.2, and with lemma 4.1 (ii) it follows $\mathsf{DEC}(\mathsf{P}), \mathsf{g}_{\mathsf{P}}(\Gamma) \vdash^{\neg\neg\mathcal{I}}_{\mathrm{m}} \neg\neg\exists x^\rho\, \mathsf{g}_{\mathsf{P}}(B)$. Since, by lemma 7.2, with respect to P the formulas in $\mathsf{g}_{\mathsf{P}}(\Gamma)$ are definite and $\mathsf{g}_{\mathsf{P}}(B)$ is a goal formula, and also $\mathsf{DEC}(\mathsf{P})$ consists of definite formulas, we can apply theorem 7.1 and get a term r with $\mathsf{reclev}(r) \leq k$ and $\mathsf{DEC}(\mathsf{P}), \mathsf{g}_{\mathsf{P}}(\Gamma) \vdash^{\mathcal{J}}_{1} \mathsf{g}_{\mathsf{P}}(B)[r/x]$ where $\mathcal{J}$ consists of predicates that are 'propositional in' formulas of the form $s\,\mathbf{mr}\,C$ with $\lambda n C \in \mathcal{I}$. From the assumptions on $\mathcal{I}$ it follows that the latter formulas are quantifier free. Finally, in presence of the assumptions $\mathsf{DEC}(\mathsf{P})$ we may remove the Gödel translation and obtain $\mathsf{DEC}(\mathsf{P}), \Gamma \vdash^{\mathcal{J}}_{1} B[r/x]$. $\dashv$

Theorem 7.4. *Let Γ,B be as in theorem 7.3 and assume*

$$\Gamma \vdash^{\neg\neg\mathcal{I}}_{\mathrm{m}_0} M \colon \neg\neg\exists x^\rho\, B$$

where all formulas in $\mathcal{I}$ are of the form $\exists y^\sigma\, C$ with C quantifier free and $\mathsf{lev}(\sigma) \leq k$. Then from M one can extract a term r with $\mathsf{lev}(r) \leq k$ such that

$$\mathsf{DEC}(\mathsf{Crit}(\Gamma, \neg B) \cup \mathsf{pc}(\mathcal{I})), \Gamma \vdash^{\mathcal{J}}_{1} B[r/x]$$

where $\mathcal{J}$ consists of equations of type boole.

PROOF. Let $\mathsf{P} = \mathsf{Crit}(\Gamma, \neg B) \cup \mathsf{pc}(\mathcal{I})$. By lemma 4.2 we have $\mathsf{g_P}(\Gamma) \vdash_{\mathrm{m}_0}^{\neg\neg\mathsf{g_P}(\mathcal{I})} \neg\neg\exists x^\rho\, \mathsf{g_P}(B)$ and with lemma 4.1 (iii) it follows $\mathsf{DEC}(\mathsf{P}), \mathsf{g_P}(\Gamma) \vdash_{\mathrm{m}}^{\neg\neg\mathcal{I}'} \neg\neg\exists x^\rho\, \mathsf{g_P}(B)$ where $\mathcal{I}'$ consists of predicates of the form $\exists y^\sigma\, E$ with E a boolean equation and $\mathsf{lev}(\sigma) \leq k$. As in the proof of theorem 7.3 we may use theorem 7.1 to obtain a term r with $\mathsf{reclev}(r) \leq k$ such that $\mathsf{DEC}(\mathsf{P}), \mathsf{g_P}(\Gamma) \vdash_1^{\mathcal{J}} \mathsf{g_P}(B)[r/x]$, but where now $\mathcal{J}$ consists of boolean equations. Hence $\mathsf{DEC}(\mathsf{P}), \Gamma \vdash_1^{\mathcal{J}} B[r/x]$. ⊣

Remarks. 1. Theorems 7.3 and 7.4 can be viewed as a generalizations of Parsons' result [16] on the Π^0_2-conservativity of classical Σ^0_1-arithmetic over primitive recursive arithmetic.

2. The theorems of this section can be extended to the case that among the assumptions there are formulas that are not definite. For such assumptions realizers of their Kripke translations are needed in the witnessing term r. An example is the negative translation of the axiom of countable choice. It is not hard to check that the realizers of other interpretations of this axiom studied [3] and [5] are also valid for our Kripke translation.

If we give up controlling the levels of inductions in proofs and the levels of recursions in extracted programs we could replace our Kripke translation by the simpler 'Friedman style' translation mapping formula A to $A[\exists y\, G/\bot]$. This is similar to the translation in [6] and does not introduce second-order logic. We have then an analogue to lemma 6.1 and therefore, as an analogue to theorem 6.3, that $\Gamma \vdash_{\mathrm{m}} \neg\neg G$ implies $\mathsf{LEM}(\mathsf{P}), \Gamma \vdash_1 G$ for definite Γ and Harrop G. Note that we neither restrict nor control induction. From this we obtain the following analogues to Theorems 7.1 and 7.3.

THEOREM 7.5. *Assume* $\Gamma \vdash_{\mathrm{m}} M : \neg\neg\exists x\, G$ *where* Γ *consists of* P*-definite formulas and* G *is a* P*-goal formula. Then* $\mathsf{DEC}(\mathsf{P}), \Gamma \vdash_1 G[r/x]$ *for some term* r *extracted from* M*.*

THEOREM 7.6. *Assume* $\Gamma \vdash_{\mathrm{m}_0} M : \neg\neg\exists x\, B$ *where* Γ *consists of quasi-negative formulas and* B *is a quasi-positive formula. Then* $\mathsf{DEC}(\mathsf{Crit}(\Gamma, \neg B)), \Gamma \vdash_1 B[r/x]$ *for some term* r *extracted from* M*.*

Remark. As pointed out by Ulrich Kohlenbach a result similar to theorem 7.6 can be obtained via Gödel's Dialectica Interpretation [11] (see also [21]). Indeed it is easily seen that the Dialectica Interpretation of a quasi-negative formula A is (up to provable equivalence) of the form $\forall x\, A_0$ with $\vdash_1^{\emptyset} A \to \forall x\, A_0$. Similarly a quasi-positive formula B has a Dialectica Interpretation $\exists x\, B_0$ with B_0 quantifier free and $\exists x\, B_0 \to B$. From this and the soundness theorem for the Dialectica Interpretation theorem 7.6 readily follows, however without control on the use of decidability of predicates in terms of Γ and B alone (in the Dialectica Interpretation the use of decidability depends on the derivation M).

§8. Examples.

We now discuss some simple examples illustrating our methods of extracting programs form proofs. To begin with we consider the following specification of the reverse of a list (we use Haskell notation for lists and operations on lists).

$$\mathsf{REV}([],[]) \tag{1}$$
$$\mathsf{REV}(v,w) \to \mathsf{REV}(v +\!\!+ [x], x : w) \tag{2}$$

Suppose we want to extract a program reversing lists from a proof of

$$\forall v\, \exists w\, \mathsf{REV}(v,w). \tag{3}$$

One way to prove (3) constructively is to prove first that any non-empty list is of the form $v +\!\!+ [x]$ and then use induction on the length of v to prove (3). Another (slightly better) solution is to add an axiom stating that REV is symmetric (from (1) and (2) symmetry does not follow) and prove $\forall v\, \exists w\, \mathsf{REV}(w,v)$ by a straightforward list induction on v. Clearly, the programs extracted from either proofs yield quadratic time algorithm for reversing lists. For example, the latter yields

```
reverse1 [] = []
reverse1 (x : v) = (reverse v) ++ [x]
```

Now let us see what we get if we prove (3) classically. We assume (some fixed) v_0 were not reversible, that is

$$\neg\exists w\, \mathsf{REV}(v_0,w) \tag{4}$$

and try to derive (in minimal arithmetic) a contradiction. Informally, from (2) it follows immediately that if $v +\!\!+ [x]$ is not reversible then v is not either. Hence, inductively, it follows from (4) that all initial segments of v_0 are non-reversible, and in particular the empty list is not reversible contradicting (1). More formally we prove

$$v +\!\!+ u = v_0 \to \neg\exists w\, \mathsf{REV}(v,w) \tag{5}$$

by induction on u, using our (false) assumption (4) to prove the base case and (2) and associativity of $+\!\!+$ to prove the step. Instantiating (5) with $v = []$, $u = v_0$ and $w = []$ yields a contradiction to (1).

A this point uniform quantifiers come into play, since we observe that we can prove (5) in fact uniformly in v, that is

$$\forall u\, \{\forall v\}\, .\, v +\!\!+ u = v_0 \to \neg\exists w\, \mathsf{REV}(v,w) \tag{6}$$

It is instructive to examine the step in the inductive proof of (6) in detail. Assume $v +\!\!+ (x : u) = v_0$ and $\mathsf{REV}(v,w)$. We have to prove $\bot$. From the first assumption it follows $(v +\!\!+ [x]) +\!\!+ u = v_0$, by associativity of $+\!\!+$, and from the second assumption we get $\mathsf{REV}(v +\!\!+ [x], x : w)$, by (2). Instantiating the induction hypothesis, which is $\{\forall v'\}\, .\, v' +\!\!+ u = v_0 \to \neg\exists w'\, \mathsf{REV}(v',w')$, with $v' = v +\!\!+ [x]$ and $w' = x : w$ we obtain $\bot$. In order for this to be a correct derivation in minimal (uniform) arithmetic ($\vdash_{\mathrm{m}}$) it is crucial that v has not become computationally relevant: The first instantiation does not count because it is an application of the rule $\{\forall\}^-$ and in the second instantiation – which does count because it is an application of the rule $\exists^+$ – the variable v does not occur.

To this derivation theorem 7.1 (or theorem 7.3) can be applied with Γ consisting of the (universal closures of) (1) and (2) (and, for convenience, associativity of $+\!\!+$) and $B = \mathsf{REV}(v_0,w)$. Then Γ is a set of goal formulas and B is definite with respect to $\mathsf{P} := \emptyset$. The extracted program is the well-known linear time algorithm for reversing lists:

```
reverse2 v0 = rev v0 []   where
    rev [] w = w
    rev (x : u) w = rev u (x : w)
```

If in (6) we had quantified the variable v non-uniformly ($\forall v$), then it would appear in the extracted program as a (superfluous) parameter of the subroutine:

```
reverse3 v0 = rev v0 [] []   where
    rev [] v w = w
    rev (x : u) v w = rev u (v ++ [x]) (x : w)
```

It should be noted that although obviously reverse3 can be transformed into reverse2, a formal proof of the correctness of this transformation requires induction (exercise: on which predicate?). On the other hand the correct application of the uniform rules is easily checked mechanically (without any proof calculus being involved).

In order to demonstrate the effect of the refinements (ii) and (iii) in theorem 7.1 we use the example of the Fibonacci numbers in [6]. The graph of the Fibonacci sequence is axiomatized by

$$\mathsf{FIB}(0,0) \wedge \mathsf{FIB}(1,1) \tag{7}$$

$$\mathsf{FIB}(n,k) \to \mathsf{FIB}(n+1,l) \to \mathsf{FIB}(n+2,k+l) \tag{8}$$

In [6] from a proof in minimal arithmetic of (essentially)

$$\forall n\, \neg\neg\exists k\, \mathsf{FIB}(n,k) \tag{9}$$

the following program was extracted (which is linear in the unary representation of natural numbers).

```
fibonacci1 n = fib n (\k -> \l -> k)  where
    fib n f = if n == 0
              then f 0 1
              else fib (n - 1) (\k -> \l -> f l (k + 1))
```

The type level 2 recursion is due to the fact that the proof contains an induction on the following generalization of (9)

$$\forall n\, \neg\neg\exists k,l\, (\mathsf{FIB}(n,k) \wedge \mathsf{FIB}(n+1,l)) \tag{10}$$

Using our theorem 7.1 the recursion can be lowered to level 0.

```
fibonacci2 n = (let (_,(k,_),_) = fibaux n in k)  where
      fibaux n = if n == 0
                 then (True,(0,1),0)
                 else let (p,(k,l),m) = fibaux (n-1)
                      in  if p == True
                          then (True,(l,k+l),0)
                          else (p,(k,l),m)
```

This program has approximately the same time complexity as `fibonacci1`. The triples `(p,(k,l),m)` come from lemma 3.4 which is used in theorem 7.1. They are clearly unnecessary because `p` will always have the value `True` (on the other hand they do not slow down the program very much either). The triples are avoided when we follow the advice in the remark after theorem 7.1 and derive the program directly from theorem 6.4 (avoiding lemma 3.4). This is possible, since the only induction in the proof does meet the required constraints.

```
fibonacci3 n = fst (fibpair n)  where
      fibpair n = if n == 0
                  then (0,1)
                  else let (k,l) = fibpair (n-1)
                       in (l,k+l)
```

Finally, if we use the method in [9] to extract a program we obtain the following.

```
fibonacci4 n = fst (fibaux n)  where
      fibaux n = if n == 0
                 then if cfib (0,0) && cfib (1,1)
                      then (0,1)
                      else (0,0)
                 else let (k,l) = fibaux (n-1)
                      in if cfib (n,l) && cfib (n+1,k+l)
                         then (l,k+l)
                         else (0,0)
```

where `cfib` is supposed to compute the characteristic function of the graph of the Fibonacci sequence.

§9. Conclusion.

In this paper we presented a system HA^{u} of uniform Heyting arithmetic and developed a refined method of extracting programs from classical proofs that in several respects extends and improves methods known so far. The main results are summarized in theorems 7.1, 7.3 and 7.4, where we give conditions under which from a minimal HA^{u} proof of $\neg\neg\exists y\, B$ from assumptions Γ a witnessing term for the existential quantifier $\exists y$ can be extracted. In addition we give bounds for the levels of recursions and decision procedures used in the extracted term. It seems fair to say that the ideas in the papers [4], [9] and [6], which formed the starting point of our work, fit together very well and support each other. By joining them new results have been obtained.

One might object that the system HA^{u} is rather complex and not easy to use, in particular in view of applications in program development. This might be partially true, but on the other hand if a proof is constructed with an extracted program in mind, the use of uniform and non-uniform quantifiers is quite natural since it makes the computational role of variables explicit. The system HA^{u} can also be used as method for optimizing ordinary proofs in HA^{ω} with respect to program extraction. Given an HA^{ω} proof M of a formula A from assumptions Γ one can tag certain quantifiers in A and Γ as uniform and then test whether in the derivation M certain occurrences of rules can be modified into their uniform version such that the result is a correct HA^{u} derivation (with an optimized extracted program). Clearly it is decidable whether such a modification of rules is possible.

This and previous work on program extraction from constructive and classical proofs has been strongly inspired by Helmut Schwichtenberg's MINLOG system [2]. MINLOG is an implementation of Heyting Arithmetic in finite types customized by a flexible rewrite system and ML-like polymorphism. MINLOG supports program extraction from constructive and classical proofs via modified realizability and A-translation. Our system HA^{u} together with optimized program extraction has been partly implemented in MINLOG.

REFERENCES

[1] F. BARBANERA and S. BERARDI, *Extracting constructive content from classical logic via control–like reductions*, ***Typed lambda calculi and applications*** (M. Bezem and J.F. Groote, editors), LNCS Vol. 664, 1993, pp. 45–59.

[2] H. BENL, U. BERGER, H. SCHWICHTENBERG, M. SEISENBERGER, and W. ZUBER, *Proof theory at work: Program development in the Minlog system*, ***Automated deduction – a basis for applications*** (W. Bibel and P.H. Schmitt, editors), Applied Logic Series, vol. II: Systems and Implementation Techniques, Kluwer Academic Publishers, Dordrecht, 1998, pp. 41–71.

[3] S. BERARDI, M. BEZEM, and T. COQUAND, *On the computational content of the axiom of choice*, ***The Journal of Symbolic Logic***, vol. 63 (1998), no. 2, pp. 600–622.

[4] U. BERGER, *Program extraction from normalization proofs*, ***Typed lambda calculi and applications (TLCA'93)*** (M. Bezem, editor), Lecture Notes in Computer Science, vol. 664, Springer Verlag, 1993, pp. 91–106.

[5] U. BERGER and P. OLIVA, *Modified bar recursion and classical dependent choice*, ***Logic Colloquium 2001***, Springer, to appear.

[6] U. BERGER, H. SCHWICHTENBERG, and W. BUCHHOLZ, *Refined program extraction from classical proofs*, ***Annals of Pure and Applied Logic***, (to appear).

[7] R. CONSTABLE and C. MURTHY, *Finding computational content in classical proofs*, ***Logical frameworks*** (G. Huet and G. Plotkin, editors), Cambridge University Press, 1991, pp. 341–362.

[8] T. COQUAND, *Computational content of classical logic*, ***Semantics and logics of computation*** (A. Pitts and P. Dybjer, editors), Cambridge University Press, 1997, pp. 33–78.

[9] T. COQUAND and M. HOFMANN, *A new method for establishing conservativity of classical systems over their intuitionistic version*, ***Math. Struct. in Comp. Science***, vol. 9 (1999), pp. 323–333.

[10] J-Y. GIRARD, *Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types*, ***Proceedings of the second scandinavian logic symposium*** (J.E. Fenstad, editor), North–Holland, Amsterdam, 1971, pp. 63–92.

[11] K. GÖDEL, *Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes*, ***Dialectica***, vol. 12 (1958), pp. 280–287.

[12] U. KOHLENBACH, *Effective bounds from ineffective proofs in analysis: an application of functional interpretation and majorization*, ***The Journal of Symbolic Logic***, vol. 57 (1992), pp. 1239–1273.

[13] G. KREISEL, *Interpretation of analysis by means of constructive functionals of finite types*, ***Constructivity in Mathematics***, (1959), pp. 101–128.

[14] J-L. KRIVINE, *Classical logic, storage operators and second-order lambda-calculus*, ***Annals of Pure and Applied Logic***, vol. 68 (1994), pp. 53–78.

[15] M. PARIGOT, *$\lambda\mu$–calculus: an algorithmic interpretation of classical natural deduction*, ***Proceedings of logic programming and automatic reasoning, St. Petersburg***, LNCS, vol. 624, Springer Verlag, Berlin, Heidelberg, New York, 1992, pp. 190–201.

[16] C. PARSONS, *On a number–theoretic choice schema and its relation to induction*, ***Intuitionism and proof theory, proceedings of the summer conference at Buffalo N.Y. 1968*** (J. Myhill A. Kino and R.E. Vesley, editors), Studies in logic and the foundations of mathematics, North–Holland, Amsterdam, 1970, pp. 459–473.

[17] C. PAULIN-MOHRING and B. WERNER, *Synthesis of ml programs in the system coq*, ***Journal of Symbolic Computation***, vol. 11 (1993), pp. 1–34.

[18] S.G. SIMPSON, ***Subsystems of second order arithmetic***, Perspectives in Mathematical Logic, Springer-Verlag, 1999.

[19] A. S. TROELSTRA, *Realizability*, ***Handbook of proof theory*** (S.R. Buss, editor), vol. 137, North Holland, Amsterdam, 1998, pp. 408–473.

[20] A. S. Troelstra and D. van Dalen, ***Constructivism in mathematics. An introduction***, vol. 121, 123, North–Holland, Amsterdam, 1988.

[21] A.S. Troelstra, ***Metamathematical investigation of intuitionistic Arithmetic and Analysis***, Lecture Notes in Mathematics, vol. 344, Springer–Verlag, 1973.

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WALES SWANSEA
SINGLETON PARK
SWANSEA
SA2 8PP, UNITED KINGDOM
E-mail: u.berger@swan.ac.uk