The Warshall algorithm and Dickson's lemma: Two examples of realistic program extraction

Ulrich Berger
Department of Computer Science, University of Wales Swansea

Helmut Schwichtenberg and Monika Seisenberger
Mathematisches Institut der Ludwig-Maximilians-Universität München

Abstract. By means of two well-known examples it is demonstrated that the method of extracting programs from proofs is manageable in practice and may yield efficient programs. The Warshall algorithm computing the transitive closure of a relation is extracted from a constructive proof that repetitions in a path can always be avoided. Secondly, we extract a program from a classical (i.e. non constructive) proof of a special case of Dickson's Lemma, by transforming the classical proof into a constructive one. These techniques (as well as the examples) are implemented in the interactive theorem prover MINLOG developed at the University of Munich.

1. Introduction

The objective of this paper is to show that the method of extracting programs from proofs is not only a powerful metamathematical tool, but is also of considerable practical interest: used in a refined way it is applicable to rather involved proofs and yields concise and efficient programs which moreover are provably correct. We present two examples highlighting different aspects of the method. The first is a constructive proof dominated by equational reasoning. It is dramatically simplified by an extensive use of term rewriting techniques. The second is a short and elegant non-constructive proof. We translate it into a constructive proof and extract an efficient higher type program.

In the following we discuss the examples in some detail. The first starts from a constructive proof of the following

PROPOSITION [WARSHALL]. *For every two points j, k in the finite field of a given relation R we can either find a repetition-free R-path leading from j to k or else find that there is no such R-path.*

Given this proof, we can extract a very short and readable program that corresponds to the Warshall algorithm. This is remarkable, because the naive test for reachability via the relation R in a set of n elements has complexity $O(n^4)$, whereas the Warshall algorithm does the same with complexity $O(n^3)$. The reason for the fact that our proof yields the better algorithm is of course the phrase 'repetition-free' in

the statement. But at the 'mathematical' level this is a rather obvious requirement, and since the program can be obtained automatically once the formal proof is known we have here an instance of the (Bates and Constable, 1985) goal of 'very high level programming'.

The second example, a special case of Dickson's Lemma (Dickson, 1913), has been proposed in (Veldman and Bezem, 1993).

PROPOSITION [DICKSON]. *Let $f(0), f(1), \ldots$ and $g(0), g(1), \ldots$ be two infinite sequences of natural numbers. Then there are indices $i, j \in \mathbb{N}$, $i < j$, such that $f(i) \leq f(j)$ and $g(i) \leq g(j)$.*

Proof. Let $i_0 \in \mathbb{N}$ be such that $f(i_0) = \min\{ f(i) \mid i \in \mathbb{N} \}$ and, inductively, $i_{n+1} > i_n$ such that $f(i_{n+1}) = \min\{ f(i) \mid i > i_n \}$. Clearly we have $i_0 < i_1 < \ldots$ and $f(i_0) \leq f(i_1) \leq \ldots$. By the wellfoundedness of $\mathbb{N}$, the sequence $g(i_0), g(i_1), \ldots$ cannot always decrease strictly. Hence there is a $k \in \mathbb{N}$ such that $g(i_k) \leq g(i_{k+1})$. Now the numbers $i := i_k$ and $j := i_{k+1}$ are as required.

This proof can be viewed as a simplification of Nash–Williams' minimal–bad–sequence argument for Higman's lemma (Nash-Williams, 1963). Clearly it is not constructive since it requires determining the minimum of an infinite set. However, by applying a refined version of the A-translation, we get a constructive proof that contains a (higher type) primitive recursive program computing from arbitrary sequences $f(0), f(1), \ldots$ and $g(0), g(1), \ldots$ indices $i < j \in \mathbb{N}$ such that $f(i) \leq f(j)$ and $g(i) \leq g(j)$.

The paper is organized as follows. Section 2 contains a brief description of the formal system we are working with. In section 3 we sketch program extraction via Kreisel's modified realizability and apply this in section 4 to the Warshall example. In section 5 we develop a refined version of the A-translation which will enable us in section 6 to transform the classical proof of our Dickson proposition into a constructive proof and to obtain a program from the translated proof.

Both examples are implemented in the interactive theorem prover MINLOG developed at the university of Munich. For a brief introduction to the system see (Benl et al., 1998). The MINLOG system as well as the examples and the MINLOG manual are available at http://www.mathematik.uni-muenchen.de/~logik/

2. Intuitionistic arithmetic for functionals of finite type

The system we consider is essentially Heyting's intuitionistic arithmetic in finite types $\mathcal{HA}^\omega$ as described e.g. in (Troelstra, 1973). Classical arithmetic is obtained by restriction to formulas without the constructive existential quantifier $\exists^*$ but using the classical definition $\exists = \neg\forall\neg$ instead. Equations are treated on the meta level by identifying terms with the same normal form.

2.1. Gödel's system T

We use Gödel's system T of primitive recursive functionals of finite type formulated as a simply typed lambda calculus with higher type arithmetic constants.

Types. $\mathsf{boole} \mid \mathsf{nat} \mid \rho \times \sigma \mid \rho \to \sigma$.

Constants. $\mathsf{true}^{\mathsf{boole}} \mid \mathsf{false}^{\mathsf{boole}} \mid 0^{\mathsf{nat}} \mid \mathsf{S}^{\mathsf{nat}\to\mathsf{nat}} \mid \mathsf{R}_{\mathsf{boole},\rho} \mid \mathsf{R}_{\mathsf{nat},\rho}$.

$\mathsf{R}_{\mathsf{nat},\rho}$ is the recursor of type $\rho \to (\mathsf{nat} \to \rho \to \rho) \to \mathsf{nat} \to \rho$ (where $\to$ associates to the right). $\mathsf{R}_{\mathsf{boole},\rho}$ is of type $\rho \to \rho \to \mathsf{boole} \to \rho$ and represents boolean induction, i.e. definition by cases. Instead of $\mathsf{R}_{\mathsf{boole},\rho} r s t$ we will often write $\mathsf{if}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s$.

Depending on applications we may add further ground types and constants.

Terms. $x^\rho \mid c^\rho$ (c^ρ a constant) $\mid \langle r, s\rangle \mid \pi_0(r) \mid \pi_1(r) \mid \lambda x^\rho r \mid rs$

with the usual typing rules. We will use the convention that application associates to the left, i.e. rst means $(rs)t$, and the "dot notation" which makes the scope of a binder as large as possible, i.e. $\lambda x\,.\,rs$ means $\lambda x\,(rs)$.

Conversions. (Writing $t+1$ for $\mathsf{S}t$.)

$$
\begin{aligned}
\pi_0\langle r, s\rangle &\mapsto r\\
\pi_1\langle r, s\rangle &\mapsto s\\
(\lambda x\, r)s &\mapsto r[s/x]\\
\mathsf{R}_{\mathsf{boole},\rho} rs\,\mathsf{true} &\mapsto r\\
\mathsf{R}_{\mathsf{boole},\rho} rs\,\mathsf{false} &\mapsto s\\
\mathsf{R}_{\mathsf{nat},\rho} rs0 &\mapsto r\\
\mathsf{R}_{\mathsf{nat},\rho} rs(t+1) &\mapsto st(\mathsf{R}_{\mathsf{nat},\rho} rst).
\end{aligned}
$$

We let $=_{\beta\mathsf{R}}$ denote the congruence generated by these conversions and identify terms r, s with $r =_{\beta\mathsf{R}} s$. By the normalization theorem due to (Tait, 1967) and (Troelstra, 1973) every term in Gödel's T has a unique normal form w.r.t. the rewrite relation generated by $\mapsto$. Hence we can decide whether $r =_{\beta\mathsf{R}} s$ holds by normalizing both r and s. Treating the calculus modulo $=_{\beta\mathsf{R}}$ makes proofs much shorter and more perspicuous because we don't have to handle equations at the logical level.

2.2. Arithmetic

Predicate symbols and atomic formulas. Each predicate symbol P has a fixed arity that is a tuple of types $(\rho_1, \ldots, \rho_k)$. Atomic formulas are of the form $P(r_1^{\rho_1}, \ldots, r_k^{\rho_k})$. If P is of arity $()$ then the atomic formula $P()$ will be written simply P.

The choice of the predicate symbols depends on the particular problem in consideration. In most cases there will be the predicate symbols $\perp$ (falsity) of arity $()$ and atom of arity (boole). The intended interpretation of atom is the set $\{\mathsf{true}\}$. Hence '$\mathsf{atom}(t)$' means '$t = \mathsf{true}$'. If confusion is unlikely, we will write t instead of $\mathsf{atom}(t)$.

Formulas. $P_t(r_1^{\rho_1}, \ldots, r_k^{\rho_k}) \mid A \wedge B \mid A \to B \mid \forall x^\rho A \mid \exists^* x^\rho A$.
A formula not containing $\exists^*$ is called *negative* or *classical*. *Negation* and the *classical existential quantifier* are defined by

$$
\begin{aligned}
\neg A &:= A \to \perp\\
\exists x\, A &:= \neg\forall x\, \neg A.
\end{aligned}
$$

Axioms for boolean and natural induction.

$$
\begin{aligned}
\mathsf{Ind}_{\forall p\, A(p)}&: \quad A(\mathsf{true}) \to A(\mathsf{false}) \to \forall p A(p)\\
\mathsf{Ind}_{\forall n\, A(n)}&: \quad A(0) \to \forall n (A(n) \to A(n+1)) \to \forall n\, A(n).
\end{aligned}
$$

Axioms for the constructive $\exists^*$.

$$
\begin{aligned}
\exists^{*+}_{x,A(x)}&: \quad \forall x\,.\,A(x) \to \exists^* x\, A(x)\\
\exists^{*-}_{x,A(x),B}&: \quad \exists^* x^\rho A(x) \to \forall x^\rho (A(x) \to B) \to B
\end{aligned}
$$

with the usual proviso that x is not free in B.

Predicate-axioms. Usually one will postulate axioms for describing the intended interpretation of the predicate symbols. For example

$$
\begin{aligned}
\mathsf{Ax}_\top&: \quad \mathsf{atom}(\mathsf{true}) \qquad \text{(truth axiom)}\\
\mathsf{Ax}_\perp&: \quad \perp \leftrightarrow \mathsf{atom}(\mathsf{false}) \qquad \text{(falsity axiom)}
\end{aligned}
$$

In most cases the intended interpretation of a predicate symbol P is a decidable relation with a decision procedure represented by a closed term $t\colon \rho_1 \to \ldots \to \rho_k \to \mathsf{boole}$, where $(\rho_1, \ldots, \rho_k)$ is the arity of P. In that case we include the axiom

$$
\mathsf{Ax}_P: \quad \forall \vec{x}\,.\,P(\vec{x}) \leftrightarrow \mathsf{atom}(t\vec{x}) \qquad \text{(characteristic axiom)}
$$

and we call P *decidable*.

For reasons to be explained later we require that the predicate axioms are negative formulas, i.e. do not contain $\exists^*$. Otherwise there is no restriction, except, of course, that they should be true in the intended model.

Derivations are formulated in a natural deduction calculus written as Curry-Howard typed lambda-terms.

$$\begin{array}{l} u^B \quad \text{(assumptions)} \mid \text{axioms} \mid \\ (\lambda u^A\, d^B)^{A\to B} \mid (d^{A\to B} e^A)^B \mid \\ \langle d^A, e^B\rangle^{A\wedge B} \mid \pi_i(d^{A_0\wedge A_1})^{A_i} \mid \\ (\lambda x^\rho d^A)^{\forall x^\rho A} \mid (d^{\forall x^\rho A} t^\rho)^{A[t^\rho/x^\rho]} \end{array}$$

where in the $\forall$-introduction $\lambda x d^A$ x must not be free in any B with u^B a free assumption variable in d.

A derivation without free assumptions is called a *proof.* More generally, a derivation d of a formula A using only assumptions from a set of formulas Γ is called a *proof of A from Γ*, written $\Gamma \vdash d : A$.

It is easy to see that stability, $\neg\neg A \to A$, is provable for all negative formulas A, by using boolean induction and the axioms $\mathsf{Ax}_\top$ and $\mathsf{Ax}_\bot$. Hence, restricting ourselves to negative formulas (which is not a restriction from a classical point of view) we have full classical logic.

3. From intuitionistic proofs to programs

We now describe how from a proof of a constructive $\forall\exists^*$-statement

$$\forall x \exists^* y\, B(x,y),$$

where $B(x,y)$ is negative, a program t can be extracted such that

$$\forall x\, B(x, tx)$$

is provable. We will do this via Kreisel's *modified realizability interpretation* (Kreisel, 1959) which is treated extensively in the literature (Troelstra, 1973), (Troelstra and van Dalen, 1988), (see also Berger, 1993, Berger and Schwichtenberg, 1995).

3.1. Program extraction

The extraction operation maps every derivation d of a formula A to a term $[\![d]\!]$ 'realizing A'. The term $[\![d]\!]$ is the program extracted from the

derivation d. In particular, if A is the formula $\forall x \exists^* y\, B(x,y)$, $B(x,y)$ negative, then '$[\![d]\!]$ realizes A' will mean $\forall x\, B(x, [\![d]\!]x)$ i.e. $[\![d]\!]$ correctly meets the specification A.

We assign to every formula A an object $\tau(A)$ that is either a type or the symbol $*$. Then $\tau(A)$ will be the type of the program extracted from a proof of A (provided $\tau(A) \neq *$).

$$\begin{aligned}
\tau(P(\vec{s})) &:= * \\
\tau(\exists^* x^\rho A) &:= \begin{cases} \rho & \text{if } \tau(A) = *, \\ \rho \times \tau(A) & \text{otherwise.} \end{cases} \\
\tau(\forall x^\rho A) &:= \begin{cases} * & \text{if } \tau(A) = *, \\ \rho \to \tau(A) & \text{otherwise.} \end{cases} \\
\tau(A_0 \wedge A_1) &:= \begin{cases} \tau(A_i) & \text{if } \tau(A_{1-i}) = *, \\ \tau(A_0) \times \tau(A_1) & \text{otherwise.} \end{cases} \\
\tau(A \to B) &:= \begin{cases} \tau(B) & \text{if } \tau(A) = *, \\ * & \text{if } \tau(B) = *, \\ \tau(A) \to \tau(B) & \text{otherwise.} \end{cases}
\end{aligned}$$

A formula is called *computationally meaningful* if $\tau(A) \neq *$. Otherwise it is called a *Harrop-formula.* Note that negative formulas are Harrop-formulas.

Next for every derivation d of a computationally meaningful formula A we define the *extracted program* $[\![d]\!]\colon \tau(A)$. For assumption variables u^A we set $[\![u^A]\!] := x_u^{\tau(A)}$, where $x_u^{\tau(A)}$ is some object variable associated with the assumption variable u^A in a one to one way, and inductively

$$\begin{aligned}
[\![\lambda u^A d]\!] &:= \begin{cases} [\![d]\!] & \text{if } \tau(A) = *, \\ \lambda x_u^{\tau(A)} [\![d]\!] & \text{otherwise.} \end{cases} \\
[\![d^{A\to B} e]\!] &:= \begin{cases} [\![d]\!] & \text{if } \tau(A) = *, \\ [\![d]\!][\![e]\!] & \text{otherwise.} \end{cases} \\
[\![\langle d_0^{A_0}, d_1^{A_1}\rangle]\!] &:= \begin{cases} [\![d_i]\!] & \text{if } \tau(A_{1-i}) = * \\ \langle [\![d_0]\!], [\![d_1]\!]\rangle & \text{otherwise.} \end{cases} \\
[\![\pi_i(d^{A_0\wedge A_1})]\!] &:= \begin{cases} [\![d]\!] & \text{if } \tau(A_{1-i}) = * \\ \pi_i[\![d]\!] & \text{otherwise.} \end{cases} \\
[\![\lambda x^\rho d]\!] &:= \lambda x^\rho [\![d]\!], \\
[\![dt]\!] &:= [\![d]\!]t.
\end{aligned}$$

The axioms are realized by suitable closed terms. For induction axioms

$$\mathsf{Ind}_{\forall n\, A(n)} \colon A(0) \to (\forall n \,.\, A(n) \to A(n+1)) \to \forall n\, A(n)$$

the realizing program $[\![\mathsf{Ind}_{\forall n\, A(n)}]\!]$ is the recursor

$$\mathsf{R}_{\mathsf{nat},\rho} \colon \rho \to (\mathsf{nat} \to \rho \to \rho) \to \mathsf{nat} \to \rho,$$

where $\rho = \tau(A(n))$. Similarly boolean induction is realized by $\mathsf{R}_{\mathsf{boole},\rho}$. For the $\exists^*$-axioms we set

$$[\![\exists^{*+}_{x^\rho, A}]\!] := \begin{cases} \lambda x^\rho x & \text{if } \tau(A) = *, \\ \lambda x^\rho \lambda y^{\tau(A)} \langle x, y \rangle & \text{otherwise.} \end{cases}$$

$$[\![\exists^{*-}_{x^\rho, A(x), B}]\!] := \begin{cases} \lambda x^\rho \lambda f^{\rho \to \tau(B)} f x & \text{if } \tau(A) = *, \\ \lambda z^{\rho \times \tau(A)} \lambda f^{\rho \to \tau(A) \to \tau(B)} f \pi_0(z) \pi_1(z) & \text{otherwise.} \end{cases}$$

It is convenient to extend the definition of $[\![d]\!]$ to derivations d^A where A is a Harrop-formula, by setting in this case $[\![d]\!] := \epsilon$, where ϵ is some new symbol. This applies in particular if A is negative. Hence the extracted program of a predicate axiom is ϵ.

Example. It is easy to see that for every quantifier free formula C containing only decidable predicate symbols, there is a term t_C such that $C \leftrightarrow t_C$ is provable. Therefore for such C we can easily derive a scheme of proof by cases

$$\mathrm{Cases}_C \colon (C \to A) \to (\neg C \to A) \to A.$$

The proof term is

$$\begin{aligned} \lambda u_1^{C \to A} \lambda u_2^{\neg C \to A}.\quad & \mathsf{Ind}_{\forall p\,.\,(p \to C) \to (C \to p) \to A} \\ & (\lambda u_3^{\mathsf{true} \to C} \lambda u_4^{C \to \mathsf{true}}.\, u_1(u_3 \mathsf{Ax}_\top)) \\ & (\lambda u_5^{\mathsf{false} \to C} \lambda u_6^{C \to \mathsf{false}}.\, u_2 \lambda u_7^C \,.\, \mathsf{Ax}_\bot(u_6 u_7)) \\ & t_C \\ & d_1^{t_C \to C} \\ & d_2^{C \to t_C} \end{aligned}$$

with extracted program

$$\lambda x_1^\rho \lambda x_2^\rho .\, \mathsf{R}_{\mathsf{boole},\rho} x_1 x_2 t_C = \lambda x_1 \lambda x_2 \,.\, \mathsf{if}\ t_C\ \mathsf{then}\ x_1\ \mathsf{else}\ x_2$$

of type $\rho \to \rho \to \rho$, where $\rho := \tau(A)$.

3.2. Modified realizability

Correctness of the extracted programs is guaranteed by the well-known soundness theorem (see below) stating that the extracted program *(modified) realizes* the formula derived.

We define formulas $r \,\mathbf{mr}\, A$, where A is a formula and r is either a term of type $\tau(A)$ if the latter is a type, or the symbol ϵ if $\tau(A) = *$.

$$\epsilon \,\mathbf{mr}\, P_t(\vec{s}) := P_t(\vec{s}),$$

$$r \,\mathbf{mr}\, \exists x^* A := \begin{cases} \epsilon \,\mathbf{mr}\, A[r/x] & \text{if } \tau(A) = *, \\ \pi_1(r) \,\mathbf{mr}\, A[\pi_0(r)/x] & \text{otherwise.} \end{cases}$$

$$r \,\mathbf{mr}\, \forall x A := \begin{cases} \forall x . \epsilon \,\mathbf{mr}\, A & \text{if } \tau(A) = *, \\ \forall x . r x \,\mathbf{mr}\, A, & \text{otherwise.} \end{cases}$$

$$r \,\mathbf{mr}\, (A_0 \wedge A_1) := \begin{cases} r \,\mathbf{mr}\, A_{1-i} \wedge\ \epsilon \,\mathbf{mr}\, A_i & \text{if } \tau(A_i) = *, \\ \pi_0(r) \,\mathbf{mr}\, A_0 \wedge \pi_1(r) \,\mathbf{mr}\, A_1 & \text{otherwise.} \end{cases}$$

$$r \,\mathbf{mr}\, (A \to B) := \begin{cases} \epsilon \,\mathbf{mr}\, A \to r \,\mathbf{mr}\, B & \text{if } \tau(A) = *, \\ \forall x . x \,\mathbf{mr}\, A \to \epsilon \,\mathbf{mr}\, B & \text{if } \tau(A) \neq * = \tau(B), \\ \forall x . x \,\mathbf{mr}\, A \to r x \,\mathbf{mr}\, B & \text{otherwise.} \end{cases}$$

Soundness Theorem. If d is a derivation of a formula B, then we can derive $[\![d]\!] \,\mathbf{mr}\, B$ from the assumptions $\{ x_u \,\mathbf{mr}\, C \mid u^C \in \mathrm{FA}(d) \}$ where $x_u := \epsilon$ if u^C is an assumption variable with a Harrop-formula C.

Proof. Induction on the structure of d. The proof can be found in any of the cited literature on realizability; see for instance (Troelstra and van Dalen, 1988).

Clearly if B is negative then $\epsilon \,\mathbf{mr}\, B = B$. (Note this need not be the case for arbitrary B with $\tau(B) = *$.) Hence for formulas of the form $\forall x^\rho \exists^* y^\sigma B(x, y)$ with negative $B(x, y)$ we have $\tau(\forall x \exists y^* A(x, y)) = \rho \to \sigma$ and

$$t \,\mathbf{mr}\, \forall x \exists^* y A(x, y) = \forall x A(x, t x).$$

Therefore we obtain as a corollary to the soundness theorem the following.

Extraction Theorem. From a proof d of $\forall x^\rho \exists y^\sigma B(x, y)$, B negative, from negative assumptions Γ one can extract a closed term $[\![d]\!]^{\rho \to \sigma}$ such that the formula $\forall x B(x, [\![d]\!] x)$ is provable from Γ.

Usually Γ consists of lemmas, i.e. true formulas that we could but do not prove in order to keep the derivation d short. The theorem says that this abbreviation does not affect program extraction.

4. The Warshall algorithm

The Warshall algorithm computes the transitive closure of a relation; its complexity is $O(n^3)$, whereas the naive algorithm needs $O(n^4)$ steps. Using the extraction method described in the preceeding section we will obtain this 'clever' algorithm from a proof that uses only the basic idea of Warshall's algorithm, i.e. to avoid repetitions in paths.

Besides nat we will use an additional ground type nats for lists of natural numbers representing paths with natural numbers as elements. It will be useful to have an error object $\perp^{\mathsf{nats}}$, since a natural function for concatenation of two paths will be only partially defined.

Lists $\hat{x}, \hat{y}, \hat{z}$ are generated from the empty list ε and the error object $\perp^{\mathsf{nats}}$ by means of $^\frown$, which constructs a list $k^\frown\hat{x}$ from a number k and a list $\hat{x}$. x, y, z (without hat $\hat{}$) denote lists $\neq \perp^{\mathsf{nats}}$, i, j, k, l, m, n natural numbers.

Our language consists of the following relation and function symbols with their intended meaning. We deal with a binary relation R on $\{0, 1, \ldots, n-1\}$, whose transitive closure is to be determined.

$$\begin{array}{rl}
k \mathrel{\mathsf{in}} \hat{x}: & k \text{ occurs in the path } \hat{x}, \\
\mathsf{rf}(\hat{x}): & \hat{x} \text{ is a repetition free path,} \\
P_i(\hat{x}, j, k): & \hat{x} \text{ is an } R\text{-path from } j \text{ to } k \\
& \text{whose inner elements are} < i, \\
\hat{x} \cdot \hat{y}: & \text{concatenation,}
\end{array}$$

where the concatenation function $\hat{x} \cdot \hat{y}$ is defined as follows. If $\hat{x}$ and $\hat{y}$ are lists with the same end and initial point respectively, then $\hat{x} \cdot \hat{y}$ is the list obtained by concatenating the two, where the same end and initial point is used only once. If end and initial points are different, the result is the error object $\perp^{\mathsf{nats}}$.

In the formal proofs we will make extensive use of equational reasoning, in the sense that a built-in term rewriting system is used to automatically decide or simplify atomic formulas in this language.

Now, we can formulate the goal that for our relation R (notationally suppressed) and given i, j, k there exists a path which either is empty (then there is no R-path from j to k with inner elements $< i$) or connects j and k with inner nodes $< i$ and is repetition free. Note that in both cases the path is total.

Proposition.

$$\forall i, j, k \exists^* x \{[x = \varepsilon \to \forall y \neg P_i(y, j, k)] \wedge [x \neq \varepsilon \to P_i(x, j, k) \wedge \mathsf{rf}(x)]\}.$$

We use the following assumptions:

$$P_i(x, j, k) \to P_{i+1}(x, j, k) \tag{1}$$
$$P_0(x, j, k) \wedge j \neq k \to R(j, k) \tag{2}$$
$$P_i(x, j, i) \wedge P_i(y, i, k) \to P_{i+1}(x \cdot y, j, k) \tag{3}$$
$$P_i(x, j, i) \wedge \mathsf{rf}(x) \wedge P_i(y, i, k) \wedge \mathsf{rf}(y) \wedge \forall z \neg P_i(z, j, k) \to \mathsf{rf}(x \cdot y) \tag{4}$$
$$P_{i+1}(x, j, k) \wedge \neg P_i(x, j, k) \to \tilde{\exists} y P_i(y, j, i) \wedge \tilde{\exists} z P_i(z, i, k) \tag{5}$$

The last assumption (where we used the classical $\tilde{\exists}$) contains the main idea of the proof: if there is a path from j to k whose inner elements are smaller than $i+1$, but not smaller than i, then there exist paths from j to i and from i to k with inner elements smaller than i. Assumption (4) can be seen as follows: under the given hypothesises $x \cdot y$ could only contain a repetition if x and y have a common inner element. But this would contradict $\forall z \neg P_i(z, j, k)$.

Proof. Induction on i. *Basis*. Given j, k. *Case* $j = k$. Let $x := j^\frown\varepsilon$. Then we have $P_0(j^\frown\varepsilon, j, k)$ since $j = k$ and $\mathsf{rf}(j^\frown\varepsilon)$ by definition. *Case* $j \neq k$. *Subcase* $R(j, k)$. Let $x := j^\frown k^\frown\varepsilon$. Then we have $P_0(j^\frown k^\frown\varepsilon, j, k)$ since $R(j, k)$ and $\mathsf{rf}(j^\frown k^\frown\varepsilon)$ since $j \neq k$. *Subcase* $\neg R(j, k)$. Let $x := \varepsilon$. We obtain $\forall y \neg P_i(y, j, k)$ by using (2).

Step $i + 1$. Given j, k. By IH we have x_i. We must find x_{i+1}.

Case $x_i = \varepsilon$. Then we have $\forall y \neg P_i(y, j, k)$. By IH, applied to j, i, (i, k respectively) we have $x_{i,j,i}$ and $x_{i,i,k}$, i.e. paths from j to i (i to k respectively) with inner elements smaller than i.

Subcase $x_{i,j,i} = \varepsilon$. Then we have $\forall y \neg P_i(y, j, i)$. Let $x_{i+1} = \varepsilon$. Given y. Assumption: $P_{i+1}(y, j, k)$. By (5) (since $\neg P_i(y, j, k)$) we have $\tilde{\exists} y P_i(y, j, i)$; contradiction.

Subcase $x_{i,j,i} \neq \varepsilon$.

Subsubcase $x_{i,i,k} = \varepsilon$. Then we have $\forall y \neg P_i(y, i, k)$. Let $x_{i+1} = \varepsilon$. Given y. Assumption: $P_{i+1}(y, j, k)$. By (5) (since $\neg P_i(y, j, k)$) we have $\tilde{\exists} z P_i(z, i, k)$; contradiction.

Subsubcase $x_{i,i,k} \neq \varepsilon$. Let $x_{i+1} := x_{i,j,i} \cdot x_{i,i,k}$. Then, by (3) we have $P_{i+1}(x_{i+1}, j, k)$ and by (4) $\mathsf{rf}(x_{i+1})$, since $\forall y \neg P_i(y, j, k)$ follows from $x_i = \varepsilon$.

Case $x_i \neq \varepsilon$. Let $x_{i+1} := x_i$. Then we have $\mathsf{rf}(x_{i+1})$ and $P_i(x_{i+1}, j, k)$, hence $P_{i+1}(x_{i+1}, j, k)$ by (1). This concludes the proof.

From a formal proof of this proposition we have extracted in Minlog the following program; this is the original output, with only the names of the variables adjusted and indentation added.

```
[R]nat-rec
([j,k][if j=k then con j eps else
      [if R j k then con j(con k eps) else eps]])
([i,g,j,k][if (g j k)=eps then
              [if (g j i)=eps then
                  eps else
                  [if (g i k)=eps then
                      eps else
                      dot(g j i)(g i k)]] else
               g j k])
```

To make this program more readable, we give the (primitive) recursion equations for the function f defined by it, returning for given i, j, k either a path from j to k with inner elements smaller than i or the empty path ε. We again suppress the parameter for the given relation:

$$
\begin{array}{rcllll}
f(0,j,k) & := & \textsf{if} & j = k & & \\
 & & \textsf{then} & j^\frown\varepsilon & & \\
 & & \textsf{else} & \textsf{if} & Rjk & \\
 & & & \textsf{then} & j^\frown k^\frown\varepsilon & \\
 & & & \textsf{else} & \varepsilon &
\end{array}
$$

$$
\begin{array}{rclllll}
f(i+1,j,k) & := & \textsf{if} & f(i,j,k) = \varepsilon & & & \\
 & & \textsf{then} & \textsf{if} & f(i,j,i) = \varepsilon & & \\
 & & & \textsf{then} & \varepsilon & & \\
 & & & \textsf{else} & \textsf{if} & f(i,i,k) = \varepsilon & \\
 & & & & \textsf{then} & \varepsilon & \\
 & & & & \textsf{else} & f(i,j,i)\cdot f(i,i,k) & \\
 & & \textsf{else} & f(i,j,k) & & &
\end{array}
$$

Remarks. 1. In order to get a truly $O(n^3)$ algorithm f has to be understood as a function computing for every i an $n \times n$ matrix of paths (and not as a function of three arguments).

2. The Warshall algorithm was also considered by (Broy and Pepper, 1981) and (Pfenning, 1990) in the context of program optimization. Pfenning informally reads off the algorithm from a similar proof and claims that this could be done automatically.

3. The Dijkstra algorithm has been treated similarly in (Benl and Schwichtenberg, 1999). However, the proof is somewhat more complex; in particular one has to be careful to select the right language. The

program extracted from the proof is as readable as the one above and not much longer.

5. From classical to intuitionistic proofs

In our second example we will study a non-constructive proof of a special case of Dickson's Lemma. To extract a program we need to transform this proof into a constructive one. In the following we describe in general how and under which conditions a classical existence proof, i.e. a proof of $\exists y B$, can be transformed into an intuitionistic existence proof, i.e. a proof of $\exists^* y\, B$. This method is mostly referred to as H. Friedman's A-translation (Friedman, 1978). It was independently investigated in (Dragalin, 1979), and variants were considered in (Leivant, 1985) and (Troelstra and van Dalen, 1988). In (Murthy, 1990) this translation has been applied to a classical proof of Higman's Lemma. The results there however showed that a literal application leads to an explosion in size of the translated proof and in turn of the extracted program. For this reason we use a refinement of the A-translation, developed in (Berger and Schwichtenberg, 1995). First we allow assumptions (without computational content) and second we apply the translation only when necessary, i.e. to atomic formulas with so-called critical relation symbols (see below). Using this refinement the extracted programs will in general have a simpler control structure and thus will be more efficient and more readable.

The basic idea of the translation is as follows: by means of Gödel's well-known negative translation a classical proof of $\exists y\, B$ (say B atomic for simplicity) is transformed into a proof in *minimal logic* of the same formula (recall that $\exists y\, B = (\forall y. B \to \bot) \to \bot$). Since in minimal logic the propositional constant $\bot$ does not play a special role, it may be replaced by an arbitrary proposition, for example by

$$A := \exists^* y\, B.$$

Applying this to our minimal proof of $\exists y\, B$ yields a proof in minimal logic of $(\forall y. B \to A) \to A$. Since the premise, $\forall y. B \to A$, is a theorem of intuitionistic logic, we obtain an intuitionistic proof of A.

To describe the refinement we first set

$$\exists y_1, \ldots, y_k \bigwedge_{i=1}^{n} B_i := (\forall y_1, \ldots, y_k\, .\, B_1 \to \ldots \to B_n \to \bot) \to \bot$$

or more briefly $\exists \vec{y} \bigwedge_i B_i := (\forall \vec{y}\, .\, \vec{B} \to \bot) \to \bot$. Suppose we have a proof $d\colon \exists \vec{y} \bigwedge_i B_i$, where B_i is quantifier free, using a set of Π-assumptions

$\Gamma = \{\forall \vec{x}_1\, C_1, \ldots, \forall \vec{x}_n\, C_n\}$, where the C_i are quantifier free. We may assume that the B_i and C_j are purely implicational formulas (otherwise using the logical equivalences $[A \wedge B \to C] \leftrightarrow [A \to B \to C]$ and $[A \to C \wedge D] \leftrightarrow [A \to C] \wedge [A \to D]$ we can push $\wedge$ to the outside). To specify the atomic formulas affected by the translation we set

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

and define the set of *L-critical* relation symbols as the smallest set satisfying the following conditions:

(i) $\perp$ is L-critical.

(ii) If $(\vec{C}_1 \to P_1(\vec{s}_1)) \to \ldots \to (\vec{C}_m \to P_m(\vec{s}_m)) \to R(\vec{t})$ is a positive subformula of a formula in L, and if for some i P_i is L-critical, then R is L-critical.

We let $A := \exists^* \vec{y} \bigwedge\!\!\!\bigwedge_i B_i$ and define an A-translation relative to L:

$$\perp^A := A \quad \text{and for } R \neq \perp$$
$$R(\vec{t})^A := \begin{cases} (R(\vec{t}) \to A) \to A & \text{if } R \text{ is } L\text{-critical,} \\ R(\vec{t}) & \text{otherwise,} \end{cases}$$
$$(B \to C)^A := B^A \to C^A.$$

Now, a constructive proof of $\exists^* \vec{y} \bigwedge\!\!\!\bigwedge_i B_i$ can be generated with help of the following lemmas.

Lemma 1. *$C \to C^A$ is derivable for any $C \in \{\, C_1, \ldots, C_n\}$.*

Lemma 2. *There is a derivation of $\forall \vec{y}.\vec{B}^A \to A$.*

Following the ideas sketched in the raw translation the A-translation transforms the proof $d\colon \exists \vec{y} \bigwedge\!\!\!\bigwedge_i B_i$ into a derivation

$$d^A\colon (\forall \vec{y}\,.\, \vec{B}^A \to A) \to A$$

from the assumptions $\forall \vec{x}_1 C_1^A, \ldots, \forall \vec{x}_n C_n^A$. By lemma 1 these assumptions are provable (from the Π-assumptions they come from), and by lemma 2 the premise $\forall \vec{y}.\vec{B}^A \to A$ is provable.

To summarize, we end up with a proof of $A = \exists^* \vec{y} \bigwedge\!\!\!\bigwedge_i B_i$ from the assumptions $\forall \vec{x}_1 C_1, \ldots, \forall \vec{x}_n C_n$, which (compared with the original A-translation) uses significantly fewer case distinctions (cf. Berger and Schwichtenberg, 1995).

6. The computational content of Dickson's Lemma

In this section we study a non-constructive proof of a special case of Dickson's Lemma. The proof will be given informally, but in such a detail that it will be immediately clear how to formalize it. We sketch how this proof is transformed into a constructive one by applying the method above, and discuss the extracted program.

6.1. The classical proof and its translation

Recall Dickson's Lemma for pairs of sequences:

Proposition. $\forall f, g\, \exists i, j.\ i < j \ \wedge\ f(j) \not< f(i) \ \wedge\ g(j) \not< g(i)$.

Analogously to the informal proof given in the introduction, we will use a (classical) minimum principle saying that if there is an x with property $P(x)$, then there also exists a minimal one, w.r.t. a given measure function m:

$$\exists\, x\ P(x) \to \exists\, x\ (P(x) \ \wedge\ \forall y.\, m(y) < m(x) \to \neg P(y)).$$

Proof of the proposition. Let f and g be given. Consider the set

$$M := \{x \mid \forall y \geq x.\, f(y) \not< f(x)\}.$$

1. M is not empty because f has a global minimum (Use the minimum principle with $P(x) :\equiv \top$ and $m :\equiv f$).
2. There is an $i \in M$ minimal w.r.t. g, i.e. $\forall y\, (g(y) < g(i) \to y \notin M)$ (Apply the minimum principle at $P(x) :\equiv x \in M$ and $m :\equiv g$).
3. By a third application of the minimum principle with $P(x) :\equiv x > i$ and $m :\equiv f$ we obtain an element $j > i$, s.t. $\forall y > i.\, f(y) \not< f(j)$.

Now, i and j are as desired: $f(j) \not< f(i)$ because $i \in M$, and, using the minimality of i, for proving $g(j) \not< g(i)$ it suffices to show $j \in M$, i.e. $\forall y \geq j\, f(y) \not< f(j)$. But this holds by minimality of j and since $i < j$.

To transform this proof into a constructive one we have to look at the set of formulas L steering the translation. Here, it consists of the kernel of the wrong assumption

$$u\colon\ \forall i, j.\ i < j \to f(j) \not< f(i) \to g(j) \not< g(i) \to \perp.$$

and the kernels of the Π-assumptions, tacitly used in the proof above

$$\begin{aligned} C_1\colon &\quad x \leq y \to x < y \\ C_2\colon &\quad x < y \to y \leq z \to x < z. \end{aligned}$$

Furthermore in the proof of the minimum principle (omitted) one use the assumptions

$$\begin{aligned} C_3\colon &\quad x < y \to y < z+1 \to x < z \\ C_4\colon &\quad \neg x < 0 \end{aligned}$$

which also have to be taken into account. Clearly the only L-critical relation symbol is $\bot$. Therefore C_4: $\neg n < 0$ and the wrong assumption u are the only assumptions affected by the translation. We do not show the constructive proof, since despite its computational content it is of no interest here.

6.2. The extracted program

The program extracted from the constructive proof depends on the proof of the minimum principle which in turn can be proved by zero-successor induction. Since we have used the minimum principle three times, we obtain a program containing three nested subroutines called Φ, Ψ and Ξ. For better readability we informally communicate them via recursion equations. The program takes two functions f and g as inputs (notationally suppressed) and returns a pair $i < j$ such that $f(i) \leq f(j)$ and $g(i) \leq g(j)$.

extracted-solution $= \Phi(f(0)+1)(0)$, where

$\Phi\colon \mathsf{nat} \to (\mathsf{nat} \to \mathsf{nat} \times \mathsf{nat})$

$$\begin{aligned} \Phi(0)(i) &= \mathsf{dummy} \\ \Phi(k{+}1)(i) &= \Psi(g(i){+}1, i, \Phi(k)) \end{aligned}$$

Here $\mathsf{dummy}\colon \mathsf{nat} \times \mathsf{nat}$ is some arbitrary term extracted from an Efq-Axiom.

$\Psi\colon \mathsf{nat} \to \mathsf{nat} \to (\mathsf{nat} \to \mathsf{nat} \times \mathsf{nat}) \to \mathsf{nat} \times \mathsf{nat}$

$$\begin{aligned} \Psi(0,i,h) &= \mathsf{dummy} \\ \Psi(l{+}1,i,h) &= \Xi_{l,i,h}(f(i{+}1){+}1)(i{+}1) \end{aligned}$$

$\Xi\colon \mathsf{nat} \to \mathsf{nat} \to (\mathsf{nat} \to \mathsf{nat} \times \mathsf{nat}) \to \mathsf{nat} \to \mathsf{nat} \to \mathsf{nat} \times \mathsf{nat}$

$$\Xi_{l,i,h}(0)(j) = \mathsf{dummy}$$

$$\begin{aligned} \Xi_{l,i,h}(m{+}1)(j) = \ &\mathsf{if}\quad g(j) < g(i) \\ &\mathsf{then}\quad \Psi(l, j, \Xi_{l,i,h}(m)) \\ &\mathsf{else}\quad \mathsf{if}\quad f(j) < f(i) \\ &\qquad\quad \mathsf{then}\quad h(j) \\ &\qquad\quad \mathsf{else}\quad (i,j) \end{aligned}$$

Remarks. 1. The program above can be optimized by suppressing the recursion parameters k, l and m which serve only as counters. Furthermore –as we will argue below– the zero cases may be omitted. We get the

optimized-solution $= \phi(0)$, where

$$\begin{aligned} \phi(i) &= \psi(i, \phi) \\ \psi(i,h) &= \xi_{i,h}(i+1) \\ \xi_{i,h}(j) &= \mathsf{if}\quad g(j) < g(i) \\ &\quad\ \mathsf{then}\quad \psi(j, \xi_{i,h}) \\ &\quad\ \mathsf{else}\quad \mathsf{if}\quad f(j) < f(i) \\ &\qquad\qquad \mathsf{then}\quad h(j) \\ &\qquad\qquad \mathsf{else}\quad (i,j) \end{aligned}$$

This optimized solution can also be extracted directly from the proof as follows: Observe that the minimum principle immediately follows from the principle of wellfounded induction w.r.t. a measure m:

$$\forall x(\forall y(m(y) < m(x) \to P(y)) \to P(x)) \to \forall x P(x)$$

In the proof above this principle was reduced to ordinary zero-successor induction, and hence realized by a primitive recursive functional. However it is also possible to realize wellfounded induction directly by the recursive functional

$$\begin{aligned} &\mathcal{R}_\tau : (\mathsf{nat} \to \mathsf{nat} \to \tau) \to \mathsf{nat} \to \tau : \\ &\mathcal{R}hx = hx(\mathcal{R}h) \end{aligned}$$

Although $\mathcal{R}_\tau$ formally is not primitive recursive and even is not total (!) it can be easily seen to realize wellfounded induction. Moreover it can be shown that in any extracted program this realizer may be substituted for the primitive recursive realizer of wellfounded induction without changing the input-output behavior of the program (but possibly making it more efficient). In our example this replacement results precisely in the optimized solution shown above.

2. Because of its second-order sub-program ξ the extracted program ϕ is probably not easy to understand. However one can transform it into the following more transparent first-order iterative program.

modified-solution $= \mathsf{mod}(\varepsilon, 0, 1)$, where

$$
\begin{array}{lllllll}
\mathsf{mod}(s,i,j) & = & \mathsf{if} & g(j) < g(i) & & & \\
 & & \mathsf{then} & \mathsf{mod}(s*i, j, j+1) & & & \\
 & & \mathsf{else} & \mathsf{if} & f(j) < f(i) & & \\
 & & & \mathsf{then} & \mathsf{if} & s = \varepsilon & \\
 & & & & \mathsf{then} & \mathsf{mod}(\varepsilon, j, j+1) & \\
 & & & & \mathsf{else} & \mathsf{mod}(\mathsf{lead}(s), \mathsf{last}(s), j) & \\
 & & & \mathsf{else} & (i,j) & &
\end{array}
$$

Here s runs over finite sequences of natural numbers, $s * i$ denotes s with i attached at the end, and $\mathsf{last}(s)$ and $\mathsf{lead}(s)$ denote the last element of s and s without the last element respectively. We emphasize that neither for a correctness proof nor for efficiency reasons is this transformation necessary.

3. Note that, of course, already the *classical* proof tells us that the systematic search for a pair (i,j) such that $f(j) \not< f(i)$ and $g(j) \not< g(i)$ will be successful. However, such a brute force search program doesn't use any information given by the proof whereas the extracted program does and hence has a chance to be 'more efficient'. Experiments with random sequences have confirmed this, even though a precise analysis of the efficiency of the extracted program remains to be done. It is not even clear what notion of efficiency should be used for programs with infinite sequences as inputs.

4. In order to obtain a good program from our proof of Dickson's Lemma we had to be very careful in formulating the assertion: for example instead of $\leq$ we had to use $\not>$. We hope to improve our extraction procedure such that it will be robust w.r.t. such changes and allows the user to formulate the problem in a most natural way.

7. Conclusion

In recent years quite a number of formal systems for extracting programs from constructive and non-constructive proofs have been studied and implemented. However not so many interesting examples have been presented yet, which would show that this method might be of practical

interest in the future. With this paper we tried to do one step in this direction by showing that fairly complicated proofs can be treated without problems and yield good results. A more involved example, done recently in the MINLOG system, is the extraction of the normalization-by-evaluation algorithm from a Tait-style normalization proof (Berger, 1993). Other interesting examples of program extraction have been studied by the group around Coquand (see e.g. Coquand and Persson, 1999) in a type theoretic context and by Kohlenbach (Kohlenbach, 1996) using a Dialectica-interpretation.

Acknowledgements

We thank H. Benl and F. Joachimski for their contributions to the work reported in this paper and two anonymous referees for their helpful comments.

References

Bates, J. L. and R. L. Constable: 1985, 'Proofs as Programs'. *ACM Transactions on Programming Languages and Systems* **7**(1), 113–136.

Benl, H., U. Berger, H. Schwichtenberg, M. Seisenberger, and W. Zuber: 1998, 'Proof theory at work: Program development in the Minlog system'. In: W. Bibel and P. Schmitt (eds.): *Automated Deduction – A Basis for Applications*, Vol. II: Systems and Implementation Techniques of *Applied Logic Series.* Dordrecht: Kluwer Academic Publishers, pp. 41–71.

Benl, H. and H. Schwichtenberg: 1999, 'Formal correctness proofs of functional programs: Dijkstra's algorithm, a case study'. In: U. Berger and H. Schwichtenberg (eds.): *Computational Logic*, Vol. 165 of *Series F: Computer and Systems Sciences.* pp. 113–126, Springer Verlag, Berlin, Heidelberg, New York.

Berger, U.: 1993, 'Program extraction from normalization proofs'. In: M. Bezem and J. Groote (eds.): *Typed Lambda Calculi and Applications*, Vol. 664 of *Lecture Notes in Computer Science.* pp. 91–106, Springer Verlag, Berlin, Heidelberg, New York.

Berger, U. and H. Schwichtenberg: 1995, 'Program Extraction from Classical Proofs'. In: D. Leivant (ed.): *Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, IN, USA, October 1994*, Vol. 960 of *Lecture Notes in Computer Science.* pp. 77–97, Springer Verlag, Berlin, Heidelberg, New York.

Broy, M. and P. Pepper: 1981, 'Program Development as a Formal Activity'. *IEEE Trans. on Software Eng.* **7**(1), 14–22.

Coquand, T. and H. Persson: 1999, 'Gröbner Bases in Type Theory'. In: T. Altenkirch, W. Naraschewski, and B. Reus (eds.): *Types for Proofs and Programs*, Vol. 1657 of *Lecture Notes in Computer Science.* Springer Verlag, Berlin, Heidelberg, New York.

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

Dragalin, A.: 1979, 'New Kinds of Realizability'. In: *Abstracts of the 6th International Congress of Logic, Methodology and Philosophy of Sciences*. Hannover, Germany, pp. 20–24.

Friedman, H.: 1978, 'Classically and intuitionistically provably recursive functions'. In: D. Scott and G. Müller (eds.): *Higher Set Theory*, Vol. 669 of *Lecture Notes in Mathematics*. pp. 21–28, Springer Verlag, Berlin, Heidelberg, New York.

Kohlenbach, U.: 1996, 'Analysing Proofs in Analysis'. In: W. Hodges, M. Hyland, C. Steinhorn, and J. Truss (eds.): *Logic: from Foundations to Applications. European Logic Colloquium (Keele, 1993)*. pp. 225–260, Oxford University Press.

Kreisel, G.: 1959, 'Interpretation of analysis by means of constructive functionals of finite types'. In: A. Heyting (ed.): *Constructivity in Mathematics*. North–Holland, Amsterdam, pp. 101–128.

Leivant, D.: 1985, 'Syntactic Translations and Provably Recursive Functions'. *The Journal of Symbolic Logic* **50**(3), 682–688.

Murthy, C.: 1990, 'Extracting Constructive Content from Classical Proofs'. Technical Report 90–1151, Dep.of Comp.Science, Cornell Univ., Ithaca, New York. PhD thesis.

Nash-Williams, C. S. J. A.: 1963, 'On well–quasi–ordering finite trees'. *Proc. Cambridge Phil. Soc.* **59**, 833–835.

Pfenning, F.: 1990, 'Program Development Through Proof Transformation'. In: W. Sieg (ed.): *Logic and Computation*, Vol. 106 of *Contemporary Mathematics*. Providence, Rhode Island, pp. 251–262, AMS.

Tait, W. W.: 1967, 'Intensional Interpretations of Functionals of Finite Type I'. *The Journal of Symbolic Logic* **32**(2), 198–212.

Troelstra, A. S.: 1973, *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, Vol. 344 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York.

Troelstra, A. S. and D. van Dalen: 1988, *Constructivism in Mathematics. An Introduction*, Vol. 121, 123 of *Studies in Logic and the Foundations of Mathematics*. North–Holland, Amsterdam.

Veldman, W. and M. Bezem: 1993, 'Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics'. *Journal of the London Mathematical Society* **47**, 193–211.

Address for Offprints:
U. Berger
Department of Computer Science
University of Wales Swansea
Singleton Park
Swansea SA2 8PP, UK

H. Schwichtenberg, Monika Seisenberger
Mathematisches Institut
der Ludwig-Maximilians-Universität München
Theresienstraße 39
80333 München, Germany