COMPUTATIONAL CONTENT OF PROOFS

HELMUT SCHWICHTENBERG

Hilbert-Bernays Summer School, Göttingen, July 2016

Proofs have two aspects: (i) they ensure the truth of what is claimed, and (ii) they may have computational content. In this short course we deal with the latter (mainly via examples), in sections

- Constructive logic
- List reversal
- Maximal scoring segment

The main points are how to uncover the (possibly hidden) computational content, and how one can use "decorations" to optimize it.

Introduction

In this introduction we deal with the basics of formalizing proofs and, via the Curry-Howard correspondence, analysing their computational structure.

Minimal logic is a system of rules for deriving logical formulas based just on the two symbols $\to$ (implication) and $\forall$ (for all). Each symbol has two rules: an introduction rule ($\to^+$, $\forall^+$) and an elimination rule ($\to^-$, $\forall^-$). The rules for implication are

$$\frac{\begin{array}{c}[A]\\ \mid M\\ B\end{array}}{A \to B}\,{\to^+} \qquad \frac{\begin{array}{cc}\mid M & \mid N\\ A \to B & A\end{array}}{B}\,{\to^-}$$

and the rules for universal quantification are

$$\frac{\begin{array}{c}\mid M\\ A\end{array}}{\forall_x A}\,{\forall^+ x} \qquad \frac{\begin{array}{cc}\mid M & \\ \forall_x A(x) & r\end{array}}{A(r)}\,{\forall^-}$$

These are Gentzen's (1935) Natural Deduction rules. Gentzen's idea was that natural deduction rules do indeed reflect the ways in which we construct logical arguments.

Notice that subderivations of the premises of rules are labelled M, N. In order to avoid obvious invalid derivations (for example $Px \to \forall_x Px$), the rule $\forall^+ x$ with conclusion $\forall_x A$ is subject to the following *(eigen-)variable*

condition: the derivation M of the premise A should not contain any open (undischarged) assumptions having x as a free variable.

It is clear that derivations need to be started off somewhere, so in addition we need to introduce assumptions and – as in the implication introduction – allow some assumptions to be closed or discharged in the course of a derivation. The notation for a discharged assumption A is $[A]$.

Here is a simple example. Assume A, B are formulas and $x \notin \mathrm{FV}(A)$, the set of variables free in A.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall_x(A \to B)]}{A \to B}\,\forall^- \qquad \dfrac{x}{A}}{B}\,\to^-}{\forall_x B}\,\forall^+ x}{A \to \forall_x B}\,\to^+}{\forall_x(A \to B) \to A \to \forall_x B}\,\to^+$$

Note that the variable condition is satisfied: x is not free in A (and also not free in $\forall_x(A \to B)$).

It is possible that a derivation makes an unnecessary "detour" – an elimination immediately following an introduction – and we may want to remove it. This can be done for implication via

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \mid M\\ B\end{array}}{A \to B}\,\to^+ \qquad \begin{array}{c}\mid N\\ A\end{array}}{B}\,\to^- \qquad \text{reduces to} \qquad \begin{array}{c}\mid N\\ A\\ \mid M\\ B\end{array}$$

and for universal quantification via

$$\dfrac{\dfrac{\begin{array}{c}\mid M(x)\\ A(x)\end{array}}{\forall_x A(x)}\,\forall^+ x \qquad r}{A(r)}\,\forall^- \qquad \text{reduces to} \qquad \begin{array}{c}\mid M(r)\\ A(r)\end{array}$$

Clearly the tree structure of logical derivations of any complexity at all becomes quite cumbersome, and the availability of some alternative representation therefore becomes increasingly important, especially when we wish to operate on derivations. The Curry-Howard correspondence provides a neat, computationally inspired alternative. The underlying idea is that if we have a derivation $M(x)$ of $A(x)$ then any means of (universally) binding the x should then represent a derivation of $\forall_x A(x)$. The notation chosen for binding the x is $\lambda_x M(x)$, denoting the function $x \mapsto M(x)$. On the side of $\to$ a derivation M of B from some assumptions A, each of which must now in addition have a label u, is then represented as $\lambda_u M(u)$, denoting the

function $u \mapsto M(u)$. This requires the labelling of assumptions so that all assumptions discharged by an application of $\to^+$ must have the same label. For example the unnecessary detour via $\to$ will thus be represented as

$$(\lambda_u M(u))N \quad \text{reduces to} \quad M(N).$$

Similarly the unnecessary detour via $\forall$ will be represented as

$$(\lambda_x M(x))r \quad \text{reduces to} \quad M(r).$$

These reductions are instances of what, in lambda calculus terms, is known as *beta reduction* and we write

$$\begin{aligned} (\lambda_u M(u))N &\quad \mapsto_\beta \quad M(N), \\ (\lambda_x M(x))r &\quad \mapsto_\beta \quad M(r). \end{aligned}$$

The lambda calculus provides an abstract setting for representing and computing functions and beta reduction is the main computational mechanism. Now there comes one further detail: we want to be able to move back and forth from derivations to lambda representations of them and back again from lambda terms to derivations. For this reason it is necessary to assign to each lambda term a "type", which will be the formula whose proof it represents. The formula type will be written as a superscript. In detail then, the first beta reduction example above now becomes

$$(\lambda_u M(u^A)^B)^{A\to B} N^A \quad \mapsto_\beta \quad M(N^A)^B.$$

In summary, the Curry-Howard correspondence is completely described by the Table 1 below.

Suppose that we have extended minimal logic with axioms introducing the existential quantifier:

$$\exists^+ \colon \forall_x(A \to \exists_x A), \qquad \exists^- \colon \exists_x A \to \forall_x(A \to B) \to B \quad (x \text{ not free in } B).$$

These now allow us to make computationally meaningful derivations. The underlying principle is this: suppose one has a derivation of a closed formula $\exists_x A(x)$ resulting from an existential introduction axiom $\exists^+$, i.e., the derivation (written as a Curry-Howard term) is of the form $\exists^+ r u^{A(r)}$. Then r (the computational content) is a witness for the existential quantifier, and it may be read off immediately. Of course the derivation may not end with an existential introduction. However, the process of normalization will beta-reduce the derivation term into one in which $\exists^+$ *is* the final operator to be applied. In general normalization is the process of computing out a lambda term, until no further beta reductions can be made. In other words, normalization reduces away all unnecessary detours.

Now suppose that the formula $\exists_x A(x)$ is not closed, say it has one free variable z. By instantiating z we obtain again a closed formula depending

Derivation	Term
$u\colon A$	u^A
$\dfrac{\begin{array}{c}[u\colon A]\\ \mid M\\ B\end{array}}{A \to B}\,{\to}^+u$	$(\lambda_{u^A} M^B)^{A\to B}$
$\dfrac{\begin{array}{cc}\mid M & \mid N\\ A \to B & A\end{array}}{B}\,{\to}^-$	$(M^{A\to B} N^A)^B$
$\dfrac{\begin{array}{c}\mid M\\ A\end{array}}{\forall_x A}\,\forall^+x$ (with var.cond.)	$(\lambda_x M^A)^{\forall_x A}$ (with var.cond.)
$\dfrac{\begin{array}{cc}\mid M & \\ \forall_x A(x) & r\end{array}}{A(r)}\,\forall^-$	$(M^{\forall_x A(x)} r)^{A(r)}$

TABLE 1. Derivation terms for $\to$ and $\forall$

on the instantiated value. Extracting a witnessing term from a normalized derivation term, as above, then provides a witness depending on the instantiated value. However, to bring out the uniformity involved in this process requires a new method, *realizability*.

In the course we will study such computational aspects of proofs, with an emphasis on optimizing of their computational content via "decoration". This will include some cases studies, done with our proof assistant Minlog[1].

[1] `www.minlog-system.de`

1. Constructive logic

We bring out the constructive content of logic, particularly in regard to the relationship between minimal and classical logic. It seems that the latter is most appropriately viewed as a subsystem of the former.

1.1. **Basics.** Negation and the classical (or weak) existential quantifier are defined by

$$\neg A := (A \to \bot),$$
$$\tilde{\exists}_x A := \neg\forall_x \neg A.$$

Here $\bot$ is just a propositional variable; we do not require any properties of it. If we leave the realm of pure logic and have say the natural numbers available, then alternatively we can use the "arithmetical falsity" $\mathbf{F}$ defined by $0 = 1$.

The following can easily be derived in (minimal) logic:

$$A \to \neg\neg A,$$
$$\neg\neg\neg A \to \neg A.$$

However, $\neg\neg A \to A$ is in general *not* derivable. Derivations for the following formulas are left as exercises.

$$(A \to B) \to \neg B \to \neg A,$$
$$\neg(A \to B) \to \neg B,$$

$$\neg\neg(A \to B) \to \neg\neg A \to \neg\neg B,$$
$$(\bot \to B) \to (\neg\neg A \to \neg\neg B) \to \neg\neg(A \to B),$$

$$\neg\neg\forall_x A \to \forall_x \neg\neg A.$$

Recall that the (constructive, or strong) existential quantifier is provided by means of the axioms

$$\exists^+ \colon \forall_x(A \to \exists_x A), \qquad \exists^- \colon \exists_x A \to \forall_x(A \to B) \to B \quad (x \text{ not free in } B),$$

and that these allow us to make computationally meaningful derivations. According to Kolmogorov (1932) a formula can be seen as a problem, and its proof as providing a solution to this problem, in the following sense:

(i) p proves $\exists_{x\in D} A(x)$ if and only if p is a pair (d, q) with $d \in D$ and q a proof of $A(d)$;
(ii) p proves $A \to B$ if and only if p is a construction transforming any proof q of A into a proof $p(q)$ of B;
(iii) p proves $\forall_{x\in D} A(x)$ if and only if p is a construction such that for all $d \in D$, $p(d)$ proves $A(d)$.

We call a formula *computationally relevant* (c.r.) if it has a strictly positive occurrence of an existential quantifier; "strictly positive" means never on the left hand side on an implication. Other formulas are called non-computational (n.c.)

To be able to express dependence on and independence of such parameters we split each of our logical connectives $\to, \forall$ into two variants, a "computational" one $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$ and a "non-computational" one $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$. This distinction (for the universal quantifier) is due to Berger (1993, 2005). Similar (but somewhat less flexible) concepts in the literature are

- the Set / Prop distinction in Coq, see Bertot and Castéran (2004, Ch.3);
- irrelevant type theory (dot notation in Agda), see Abel and Scherer (2012);
- propositional truncation in homotopy type theory, see Univalent Foundations Program (2013, Ch.3), and
- bracket types, see Awodey and Bauer (2004).

One can view this "decoration" of $\to, \forall$ as turning our (minimal) logic into a "computational logic", which is able to express dependence on and independence of parameters. The rules for $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$ are similar to the ones for $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$: we only need to restrict the introduction rules $(\to^{\mathrm{nc}})^+, (\forall^{\mathrm{nc}})^+$ to situations where the (assumption or object) variable bound by this rule is not "used computationally". It can be defined by requiring that the bound variable has no trace in the "extracted term" of the derivation (see below). For readability we usually write $\to, \forall$ for $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$.

We refine the distinction between computationally relevant (c.r.) and non-computational (n.c.) formulas by providing a type. To indicate that there is no computational content we introduce a "nulltype" symbol $\circ$ and extend the use of $\rho \to \sigma$ and $\rho \times \sigma$ by

$$(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ,$$
$$(\rho \times \circ) := \rho, \quad (\circ \times \sigma) := \sigma, \quad (\circ \times \circ) := \circ.$$

The type $\tau(A)$ of a formula A is defined by

$$\tau(\exists_{x^\rho} A) := \rho \times \tau(A),$$
$$\tau(A \to B) := (\tau(A) \to \tau(B)), \quad \tau(A \to^{\mathrm{nc}} B) := \tau(B),$$
$$\tau(\forall_{x^\rho} A) := (\rho \to \tau(A)), \quad \tau(\forall^{\mathrm{nc}}_{x^\rho} A) := \tau(A).$$

For every c.r. formula A we define an n.c. formula $z\ \mathbf{r}\ A$ with z a variable of type $\tau(A)$:

$$(d, z)\ \mathbf{r}\ \exists_x A(x) := z\ \mathbf{r}\ A(d),$$

$$
\begin{aligned}
z\ \mathbf{r}\ (A \to B) &:= \begin{cases} \forall_w(w\ \mathbf{r}\ A \to zw\ \mathbf{r}\ B) & \text{if } A \text{ is c.r.} \\ A \to z\ \mathbf{r}\ B & \text{if } A \text{ is n.c.} \end{cases} \\
z\ \mathbf{r}\ (A \to^{\mathrm{nc}} B) &:= A \to z\ \mathbf{r}\ B \\
z\ \mathbf{r}\ \forall_x A &:= \forall_x(zx\ \mathbf{r}\ A) \\
z\ \mathbf{r}\ \forall_x^{\mathrm{nc}} A &:= \forall_x(z\ \mathbf{r}\ A).
\end{aligned}
$$

Finally, for a derivation M of a c.r. formula A we define its *extracted term* $\mathrm{et}(M)$, of type $\tau(A)$. It will be a term in our underlying term language. This definition is relative to a fixed assignment of object variables to assumption variables: to every assumption variable u^A for a c.r. formula A we assign an object variable z_u of type $\tau(A)$. For derivations M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Otherwise

$$
\begin{aligned}
\mathrm{et}(u^A) &:= z_u^{\tau(A)} \quad (z_u^{\tau(A)} \text{ uniquely associated to } u^A), \\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \begin{cases} \lambda_{z_u}^{\tau(A)}\mathrm{et}(M) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.} \end{cases} \\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \begin{cases} \mathrm{et}(M)\mathrm{et}(N) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.} \end{cases} \\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) &:= \lambda_x^\rho \mathrm{et}(M), \\
\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) &:= \mathrm{et}(M) r, \\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}} B}) &:= \mathrm{et}(M), \\
\mathrm{et}((M^{A\to^{\mathrm{nc}} B} N^A)^B) &:= \mathrm{et}(M), \\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x^{\mathrm{nc}} A}) &:= \mathrm{et}(M), \\
\mathrm{et}((M^{\forall_x^{\mathrm{nc}} A(x)} r)^{A(r)}) &:= \mathrm{et}(M).
\end{aligned}
$$

It remains to define extracted terms for the axioms. If for instance our theory refers to natural numbers or lists of natural numbers, the extracted term of the induction axiom is the recursion operator $\mathcal{R}$ for this data type. For this to work it is necessary to the universally quantify the parameters of the induction axiom by $\forall^{\mathrm{nc}}$.

Theorem (Soundness)**.** *Let M be a derivation of a c.r. formula A from assumptions $u_i\colon C_i$ $(i < n)$. Then we can derive* $\mathrm{et}(M)\ \mathbf{r}\ A$ *from assumptions $z_{u_i}\ \mathbf{r}\ C_i$ in case C_i is c.r. and C_i otherwise.*

The proof is by induction on M.

2. List reversal

This is an example of "program development by proof transformation". The standard proof of list reversal has as its computational content a quadratic algorithm. A certain "decoration algorithm" (see Schwichtenberg and Wainer (2012, Sec. 7.5)) applied to the formalization of this proof transforms a universal quantifier $\forall_{v_1}$ into a "non-computational" one $\forall^{\mathrm{nc}}_{v_1}$. The new proof has as its content the well-known linear algorithm for list reversal, using an accumulator.

2.1. **Existence proof.** We first give an informal weak existence proof for list reversal. Write vw for the result $v * w$ of appending the list w to the list v, vx for the result $v * x$: of appending the one element list x: to the list v, and xv for the result $x :: v$ of constructing a list by writing an element x in front of a list v, and omit the parentheses in $R(v, w)$ for (typographically) simple arguments. Assume

$$\begin{aligned}&\mathrm{InitR}\colon R([],[]),\\ &\mathrm{GenR}\colon \forall_{v,w,x}(Rvw \to R(vx, xw)).\end{aligned}$$

We view R as a predicate variable without computational content. The reader should not be confused: of course these formulas involving R do express how a computation of the reverted list should proceed. But the predicate R itself only represents the graph of the list reversal function.

Let us now prove

$$\forall_v \tilde{\exists}_w Rvw \qquad (:= \forall_v(\forall_w(Rvw \to \bot) \to \bot)). \tag{1}$$

Fix R, v and assume `InitR`, `GenR` and the "false" assumption $u\colon \forall_w \neg Rvw$; we need to derive a contradiction. To this end we prove that all initial segments of v are non-revertible, which contradicts `InitR`. More precisely, from u and `GenR` we prove

$$\forall_{v_2} A(v_2) \quad \text{with } A(v_2) := \forall_{v_1}(v_1 v_2 = v \to \forall_w \neg Rv_1 w)$$

by induction on v_2. For $v_2 = []$ this follows from $u_0\colon v_1\ [] = v$ and our ("false") assumption u. For the step case, assume $u_1\colon v_1(xv_2) = v$, fix w and assume further $u_2\colon Rv_1 w$. We must derive a contradiction. We use the induction hypotheses with $v_1 x$ and xw to obtain the desidered contradiction. This requires us to prove (i) $(v_1 x)v_2 = v$ and (ii) $R(v_1 x, xw)$. But (i) follows from u_1 using properties of the append function, and (ii) follows from u_2 using `GenR`.

Our goal now is to machine extract computational content from this proof, which requires full formalization, and in particular to identify all axioms and lemmas used. Certainly induction and existence introduction $\exists^+$ will show up, but also dealing with equalities needs some attention. We have the

generally available (n.c.) Leibniz equality $=^{\mathrm{d}}$ with axioms $v =^{\mathrm{d}} v$ and the (c.r.) compatibility axiom

$$\mathrm{CompatRev}\colon \forall^{\mathrm{nc}}_{v,w}(v =^{\mathrm{d}} w \to Xw \to Xv).$$

In addition for "finitary" data types like natural numbers or lists of natural numbers we have the decidable equality as a binary boolean valued function defined by certain equations used as "computation rules". In case of the natural numers these are

$$\begin{aligned} &(0 =_{\mathbf{N}} 0) = \mathtt{tt}, && (Sn =_{\mathbf{N}} 0) = \mathtt{ff},\\ &(0 =_{\mathbf{N}} Sm) = \mathtt{ff}, && (Sn =_{\mathbf{N}} Sm) = (n =_{\mathbf{N}} m). \end{aligned}$$

One can prove easily that Leibniz equality implies decidable equality:

$$\mathrm{EqToEqD}\colon \forall_{v,w}(v = w \to v =^{\mathrm{d}} w).$$

This lemma will show up in our formalization, but since it is n.c. it will not influence the extracted term.

The proof term is displayed in Figure 1.

$$\begin{aligned} M :=& \lambda_{R,v}\lambda_{u_{\mathrm{InitR}}}\lambda_{u_{\mathrm{GenR}}}\lambda_u^{\forall_w\neg Rvw}(\\ &\mathrm{Ind}_{v_2,A(v_2)}vRvM_{\mathrm{Base}}M_{\mathrm{Step}}\ []\ \mathrm{Truth}^{[]\,v=v}\ []\ u_{\mathrm{InitR}})\\ &\text{with}\\ M_{\mathrm{Base}} :=& \lambda_{v_1}\lambda_{u_0}^{v_1[]=v}(\\ &\mathrm{CompatRev}\,\{\,v \mid \forall_w\neg Rvw\,\}\,R\,v\,v_1\,v\,(\mathrm{EqToEqD}\,v_1vu_0)u),\\ M_{\mathrm{Step}} :=& \lambda_{x,v_2}\lambda_{u_0}^{A(v_2)}\lambda_{v_1}\lambda_{u_1}^{v_1(xv_2)=v}\lambda_w\lambda_{u_2}^{Rv_1w}(\\ &u_0(v_1x)u_1(xw)(u_{\mathrm{GenR}}v_1wxu_2)). \end{aligned}$$

FIGURE 1. Proof term for list reversal

We now have a proof M of $\forall_v\tilde{\exists}_w Rvw$ from $\mathrm{InitR}\colon D_1$ and $\mathrm{GenR}\colon D_2$, with $D_1 := R([],[])$ and $D_2 := \forall_{v,w,x}(Rvw \to R(vx,xw))$. Replace $\bot$ throughout by $\exists_w Rvw$. The end formula $\tilde{\exists}_w Rvw := \neg\forall_w\neg Rvw := \forall_w(Rvw \to \bot) \to \bot$ is turned into $\forall_w(Rvw \to \exists_w Rvw) \to \exists_w Rvw$. Since its premise is an instance of existence introduction we obtain a derivation $M^{\exists}$ of $\exists_w Rvw$. The term `neterm` extracted in Minlog from a formalization of the proof above is

```
[R,v](Rec list nat=>list nat=>list nat=>list nat)v([v0,v1]v1)
 ([x,v0,g,v1,v2]g(v1++x:)(x::v2)) (Nil nat) (Nil nat)
```

with g a variable for binary functions on lists. In fact, the underlying algorithm defines an auxiliary function h by

$$h([], v_1, v_2) := v_2, \qquad h(xv, v_1, v_2) := h(v, v_1x, xv_2)$$

and gives the result by applying h to the original list and twice $[]$.

Notice that the second argument of h is not needed. However, its presence makes the algorithm quadratic rather than linear, because in each recursion step v_1x is computed, and the list append function is defined by recursion on its first argument.

One may see that the source of this problem is the use of $\forall_{v_1}$ rather than $\forall^{\mathrm{nc}}_{v_1}$ in the proof (by induction on v_2) of the formula $A(v_2) := \forall_{v_1}(v_1v_2 = v \to \forall_w \neg Rv_1w)$. In fact, v_1 is not used computationally in this proof.

It will turn out that a certain decoration algorithm is able to automatically detect and repair this point. It returns a decorated version of the proof, where $\forall^{\mathrm{nc}}_{v_1}$ occurs. This makes the corresponding algorithm linear.

2.2. **Decoration algorithm.** The *sequent* $\mathrm{Seq}(M)$ of a proof M consists of its *context* and *end formula*. The *proof pattern* $\mathrm{P}(M)$ of a proof M is the result of marking in c.r. parts of M (i.e., not above a n.c. formula) all occurrences of implications and universal quantifiers as non-computational, except the "uninstantiated" formulas of axioms and theorems. For instance, the induction axiom for $\mathbf{N}$ consists of the uninstantiated formula $\forall_n(X0 \to \forall_n(Xn \to X(Sn)) \to Xn^{\mathbf{N}})$ with a predicate variable X and a predicate substitution $X \mapsto \{\, x \mid A(x) \,\}$. Notice that a proof pattern in most cases is not a correct proof, because at axioms formulas may not fit.

We say that a formula D *extends* C if D is obtained from C by changing some (possibly zero) of its occurrences of non-computational implications and universal quantifiers into their computational variants $\to$ and $\forall$.

A proof N *extends* M if (i) N and M are the same up to variants of implications and universal quantifiers in their formulas, and (ii) every formula in c.r. parts of M is extended by the corresponding one in N. Every proof M whose proof pattern $\mathrm{P}(M)$ is U is called a *decoration* of U.

In the sequel we assume that every axiom has the property that for every extension of its formula we can find a further extension which is an instance of an axiom, and which is the least one under all further extensions that are instances of axioms. This property clearly holds for axioms whose uninstantiated formula only has $\to$ and $\forall$, for instance induction. However, in $\forall_n(A(0) \to \forall_n(A(n) \to A(Sn)) \to A(n^{\mathbf{N}}))$ the given extension of the four A's might be different. One needs to pick their "least upper bound" as further extension.

$$\dfrac{\dfrac{\dfrac{}{v_1 =^{\mathrm{d}} v \to \forall_w \neg^{\exists} Rvw \to \forall_w \neg^{\exists} Rv_1 w}\,\mathrm{CompatRev}\quad R\quad v\quad v_1\quad v \qquad \begin{array}{c}[u_1 \colon v_1\,[] = v]\\ \mid N_1\end{array}}{\forall_w \neg^{\exists} Rvw \to \forall_w \neg^{\exists} Rv_1 w}\qquad \dfrac{}{\exists^{+}\quad R\quad v}}{\dfrac{\dfrac{\forall_w \neg^{\exists} Rv_1 w}{v_1\,[] = v \to \forall_w \neg^{\exists} Rv_1 w}\to^{+} u_1}{\forall_{v_1}(v_1\,[] = v \to \forall_w \neg^{\exists} Rv_1 w)\quad (= A([]))}}$$

FIGURE 2. Base derivation M_B

Theorem (Dedoration algorithm)**.** *Under the assumption above, for every proof pattern U and every extension of its sequent* $\mathrm{Seq}(U)$ *we can find a decoration M_∞ of U such that*

(a) $\mathrm{Seq}(M_\infty)$ *extends the given extension of* $\mathrm{Seq}(U)$*, and*
(b) M_∞ *is* optimal *in the sense that any other decoration M of U whose sequent* $\mathrm{Seq}(M)$ *extends the given extension of* $\mathrm{Seq}(U)$ *has the property that M also extends M_∞.*

The proof is by induction on derivations.

2.3. **Decoration of the list reversal proof.** We present our proof in more detail, particularly by writing proof trees with formulas. Recall that we essentially use list induction. The full derivation M is obtained from

$$\dfrac{\dfrac{\dfrac{}{A([]) \to \forall_{x,v_2}(A(v_2) \to A(xv_2)) \to A(v)}\,\mathrm{Ind}\quad v\quad R\quad v \qquad \begin{array}{c}\mid M_B\\ A([])\end{array}}{\forall_{x,v_2}(A(v_2) \to A(xv_2)) \to A(v)} \qquad \begin{array}{c}\mid M_S\\ \forall_{x,v_2}(A(v_2) \to A(xv_2))\end{array}}{\forall_{v_1}(v_1 v = v \to \forall_w \neg^{\exists} Rv_1 w)\quad (= A(v))}$$

by applying it to [], Truth, [] and InitR and finally introducing $\forall_v$. Here

$$\begin{aligned}
&\mathrm{Ind}\colon && \forall^{\mathrm{nc}}_{v,R}\forall_w(A([]) \to \forall_{x,v_2}(A(v_2) \to A(xv_2)) \to A(w)),\\
&A(v_2) := && \forall_{v_1}(v_1 v_2 = v \to \forall_w \neg^{\exists} Rv_1 w),\\
&\neg^{\exists} B := && B \to \exists_w Rvw.
\end{aligned}$$

The end formula then is $\forall_v \exists_w Rvw$. We have used the base derivation M_B in Figure 2 with N_1 involving EqToEqD: $\forall_{v,w}(v = w \to v =^{\mathrm{d}} w)$, and

$$\begin{aligned}
&\mathrm{CompatRev}\colon && \forall^{\mathrm{nc}}_{R,v,v_1,v_2}(v_1 =^{\mathrm{d}} v_2 \to \forall_w \neg^{\exists} Rv_2 w \to \forall_w \neg^{\exists} Rv_1 w)\\
&\exists^{+}\colon && \forall^{\mathrm{nc}}_{R,v}\forall_w \neg^{\exists} Rvw.
\end{aligned}$$

We have also used the step derivation M_S in Figure 3 with N_2 involving the

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[u_0\colon A(v_2)]\quad v_1x}{(v_1x)v_2{=}v \to \forall_w\neg^\exists R(v_1x,w)}\quad [u_1\colon v_1(xv_2){=}v]}{\forall_w\neg^\exists R(v_1x,w)}\quad xw}{\neg^\exists R(v_1x,xw)}\qquad \dfrac{[u_2\colon Rv_1w]}{\mid N_2}}{R(v_1x,xw)}}{\exists_w Rvw}\to^+u_2}{\neg^\exists Rv_1w}}{\forall_w\neg^\exists Rv_1w}}{v_1(xv_2)=v\to\forall_w\neg^\exists Rv_1w}\to^+u_1}{\forall_{v_1}(v_1(xv_2){=}v\to\forall_w\neg^\exists Rv_1w)\ (=A(xv_2))}}{A(v_2)\to A(xv_2)}}{\forall_{x,v_2}(A(v_2)\to A(xv_2))}\to^+u_0$$

FIGURE 3. Step derivation M_S

assumption GenR: $\forall_{v,w,x}(Rvw \to R(vx,xw))$.

We now apply the decoration algorithm. Notice that the sequent or our derivation consists of the context

$$\text{InitR}\colon R([],[]) \qquad \text{GenR}\colon \forall_{v,w,x}(Rvw \to R(vx,xw))$$

and the end formula $\forall_v\exists_w Rvw$. Among the axioms used, the only ones in c.r. parts are list induction, CompatRev and $\exists^+$. If we now form the proof pattern as defined above, we obtain a clash at the list induction axiom. Recall that it is given by its uninstantiated formula

$$\forall_v(X([]) \to \forall_{x,v_2}(X(v_2)\to X(xv_2)) \to X(v))$$

and the predicate substitution $X \mapsto \{\, v \mid A(v)\,\}$. When forming the proof pattern, $A(v_2)$ is changed into $\hat{\mathrm{A}}(v_2) := \forall^{\mathrm{nc}}_{v_1}(v_1v_2 = v \to \forall^{\mathrm{nc}}_w\neg^\exists Rv_1w)$, but the uninstantiated formula is not touched. The clash then consists in the fact that the conclusion of the decorated induction axiom

$$\forall^{\mathrm{nc}}_{v,R}\forall_w(\hat{\mathrm{A}}([]) \to \forall_{x,v_2}(\hat{\mathrm{A}}(v_2)\to\hat{\mathrm{A}}(xv_2)) \to \hat{\mathrm{A}}(w)),$$

is a proper extension of what is in the proof pattern:

$$\forall^{\mathrm{nc}}_{v,R,w}(\hat{\mathrm{A}}([]) \to^{\mathrm{nc}} \forall^{\mathrm{nc}}_{x,v_2}(\hat{\mathrm{A}}(v_2)\to^{\mathrm{nc}}\hat{\mathrm{A}}(xv_2)) \to^{\mathrm{nc}} \hat{\mathrm{A}}(w)).$$

The decoration algorithm now replaces the latter by the former. Similarly in M_B the conclusions of the decorated axioms CompatRev and $\exists^+$

$$\forall^{\mathrm{nc}}_{R,v,v_1,v_2}(v_1 =^{\mathrm{d}} v_2 \to \forall^{\mathrm{nc}}_w\neg^\exists Rv_2w \to \forall^{\mathrm{nc}}_w\neg^\exists Rv_1w)$$
$$\forall^{\mathrm{nc}}_{R,v}\forall_w(Rvw \to \exists_w Rvw)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{v_1 =^{\mathrm{d}} v \to \forall_w \neg^{\exists} Rvw \to \forall_w \neg^{\exists} Rv_1 w}\,\mathrm{CompatRev}\quad R\quad v\quad v_1\quad v \qquad \dfrac{[u_1 : v_1\,[] = v]}{\vdots\, N_1}}{v_1\,[] =^{\mathrm{d}} v}}{\forall_w \neg^{\exists} Rvw \to \forall_w \neg^{\exists} Rv_1 w} \qquad \dfrac{}{\forall_w \neg^{\exists} Rvw}\,\exists^{+}\quad R\quad v}{\forall_w \neg^{\exists} Rv_1 w}}{v_1\,[] = v \to \forall_w \neg^{\exists} Rv_1 w}\to^{+} u_1}{\forall^{\mathrm{nc}}_{v_1}(v_1\,[] = v \to \forall_w \neg^{\exists} Rv_1 w) \quad (= A'([]))}$$

FIGURE 4. Base derivation M'_B

are proper extensions of what is in the proof pattern

$$\forall^{\mathrm{nc}}_{R,v,v_1,v_2}(v_1 =^{\mathrm{d}} v_2 \to \forall^{\mathrm{nc}}_w \neg^{\exists} Rv_2 w \to^{\mathrm{nc}} \forall^{\mathrm{nc}}_w \neg^{\exists} Rv_1 w)$$
$$\forall^{\mathrm{nc}}_{R,v,w}(Rvw \to \exists_w Rvw)$$

and the decoration algorithm replaces the latter by the former. But now the end formula $\forall_w \neg^{\exists} Rvw$ of the $\exists^{+}$-derivation is a proper extension of the premise of the conclusion $\forall^{\mathrm{nc}}_w \neg^{\exists} Rv_2 w \to \forall^{\mathrm{nc}}_w \neg^{\exists} Rv_1 w)$ of the CompatRev-derivation. This requires us to go back into the CompatRev-derivation and change the predicate substitution $X \mapsto \{\, v \mid \hat{\mathrm{A}}(v)\,\}$ to $X \mapsto \{\, v \mid A'(v)\,\}$ with $A'(v_2) := \forall^{\mathrm{nc}}_{v_1}(v_1 v_2 = v \to \forall_w \neg^{\exists} Rv_1 w)$. Thus we obtain the derivation in Figure 4.

But now we have a clash where M'_B is used:

$$\dfrac{\dfrac{}{\hat{\mathrm{A}}([]) \to \forall_{x,v_2}(\hat{\mathrm{A}}(v_2) \to \hat{\mathrm{A}}(xv_2)) \to \hat{\mathrm{A}}(v)}\,\mathrm{Ind}\quad v\quad R\quad v \qquad \begin{matrix}|M'_B\\ A'([])\end{matrix}}{\forall_{x,v_2}(\hat{\mathrm{A}}(v_2) \to^{\mathrm{nc}} \hat{\mathrm{A}}(xv_2)) \to^{\mathrm{nc}} \hat{\mathrm{A}}(v)}$$

Thus we have to go again into the left hand derivation and change the predicate substitution $X \mapsto \{\, v \mid \hat{\mathrm{A}}(v)\,\}$ used in the induction axiom into $X \mapsto \{\, v \mid A'(v)\,\}$. This gives us $\forall_{x,v_2}(A'(v_2) \to A'(xv_2)) \to A'(v)$.

Now the next clash appears where we used M_S: we have to change $\mathrm{P}(M_S)$ with end formula $\forall^{\mathrm{nc}}_{x,v_2}(\hat{\mathrm{A}}(v_2) \to^{\mathrm{nc}} \hat{\mathrm{A}}(xv_2))$ into a derivation M'_S of $\forall_{x,v_2}(A'(v_2) \to A'(xv_2))$. But this is easy, since no c.r. axioms are involved: just change $\forall^{\mathrm{nc}}_x$, $\forall^{\mathrm{nc}}_w$ everywhere into $\forall_x$, $\forall_w$.

Finally we obtain

$$\dfrac{\dfrac{\dfrac{\text{Ind}\quad v\quad R\quad v}{A'([]) \to \forall_{x,v_2}(A'(v_2) \to A'(xv_2)) \to A'(v)} \quad A'([])}{\forall_{x,v_2}(A'(v_2) \to A'(xv_2)) \to A'(v)} \quad \forall_{x,v_2}(A'(v_2) \to A'(xv_2))}{\forall^{\mathrm{nc}}_{v_1}(v_1 v = v \to \forall_w \neg^{\exists} R v_1 w) \quad (= A'(v))} \qquad \begin{array}{l} \mid M'_B \\ \\ \mid M'_S \end{array}$$

Applying this it to [], Truth, [] and InitR and finally introducing $\forall_v$ gives us a decorated derivation of formula $\forall_v \exists_w Rvw$. The difference is that induction is now used w.r.t. the formula $A'(v_2) := \forall^{\mathrm{nc}}_{v_1}(v_1 v_2 = v \to \forall_w \neg^{\exists} R v_1 w)$ with $\forall^{\mathrm{nc}}_{v_1}$ rather than $\forall_{v_1}$.

The extracted term `neterm` then is

```
[R,v](Rec list nat=>list nat=>list nat)v([v0]v0)
  ([x,v0,f,v1]f(x::v1))(Nil nat)
```

with `f` a variable for unary functions on lists. To run this algorithm one has to normalize the term obtained by applying `neterm` to a list:

```
(pp (nt (mk-term-in-app-form neterm (pt "1::2::3::4:"))))
```

The returned value is the reverted list `4::3::2::1:`. This time, the underlying algorithm defines an auxiliary function g by

$$g([], w) := w, \qquad g(x :: v, w) := g(v, x :: w)$$

and gives the result by applying g to the original list and []. In conclusion, we have obtained (by machine extraction from an automated decoration of a weak existence proof) the standard linear algorithm for list reversal, with its use of an accumulator.

3. Maximal scoring segment

The final example is due to Bates and Constable (1985), and deals with the "maximal scoring segment" (MSS) problem. Let X be a set with a linear ordering $\leq$, and consider an infinite sequence $f\colon \mathbf{N} \to X$ of elements of X. Assume further that we have a function $M\colon (\mathbf{N} \to X) \to \mathbf{N} \to \mathbf{N} \to X$ such that $M(f, i, k)$ "measures" the segment $f(i), \ldots, f(k)$. The task is to find a segment determined by $i \leq k \leq n$ such that its measure is maximal. To simplify the formalization let us consider M and f fixed and define $\mathrm{seg}(i, k) := M(f, i, k)$.

Such a problem appears e.g. in computational biology, when one wants to compute regions with high G, C content in DNA. Let

$$X := \{G, C, A, T\},$$
$$g\colon \mathbf{N} \to X \quad \text{(gene)},$$

$$f\colon \mathbf{N} \to \mathbf{Z}, \quad f(i) := \begin{cases} 1 & \text{if } g(i) \in \{G, C\}, \\ -1 & \text{if } g(i) \in \{A, T\}, \end{cases}$$
$$\mathrm{seg}(i,k) = f(i) + \cdots + f(k).$$

Of course we can simply solve this problem by trying all possibilities; these are $O(n^2)$ many. The first proof to be given below corresponds to this general claim. Then we will show that for a more concrete problem with the sum $x_i + \cdots + x_k$ as measure the proof can be simplified, using monotonicity of the sum at an appropriate place. From this simplified proof one can extract a better algorithm, which is linear rather than quadratic. Our goal is to achieve this effect by decoration.

We provide two lemmata proving the existence of a maximal end segment for $n+1$. The first one is

$$\mathtt{L}\colon \forall_n \exists_{j \le n+1} \forall_{j' \le n+1} (\mathrm{seg}(j', n+1) \le \mathrm{seg}(j, n+1)).$$

Its proof introduces an auxiliary variable m and proceeds by induction on m, with n a parameter:

$$\forall_n^{\mathrm{nc}} \forall_{m \le n+1} \exists_{j \le n+1} \forall_{j' \le m} (\mathrm{seg}(j', n+1) \le \mathrm{seg}(j, n+1)).$$

The second one is

$$\mathtt{LMon}\colon \forall_n^{\mathrm{nc}} (\mathtt{ES}_n \to \mathtt{Mon} \to \exists_{j \le n+1} \forall_{j' \le n+1} (\mathrm{seg}(j', n+1) \le \mathrm{seg}(j, n+1))).$$

It has as additional assumptions the existence $\mathtt{ES}_n$ of a maximal end segment for n

$$\mathtt{ES}_n\colon \exists_{j \le n} \forall_{j' \le n} (\mathrm{seg}(j', n) \le \mathrm{seg}(j, n))$$

and the assumption $\mathtt{Mon}$ of monotonicity of seg

$$\mathtt{Mon}\colon \mathrm{seg}(i,k) \le \mathrm{seg}(j,k) \to \mathrm{seg}(i, k+1) \le \mathrm{seg}(j, k+1).$$

The proof proceeds by cases on $\mathrm{seg}(j, n+1) \le \mathrm{seg}(n+1, n+1)$. If $\le$ holds, take $n+1$, else the previous j.

We now prove the existence of a maximal segment by induction on n, simultaneously with the existence of a maximal end segment.

$$\mathtt{MaxSegMon}\colon \forall_n (\exists_{i \le k \le n} \forall_{i' \le k' \le n} (\mathrm{seg}(i', k') \le \mathrm{seg}(i, k)) \wedge^{\mathrm{d}} \exists_{j \le n} \forall_{j' \le n} (\mathrm{seg}(j', n) \le \mathrm{seg}(j, n)))$$

In the step, we compare the maximal segment i, k for n with the maximal end segment $j, n+1$ provided separately. If $\le$ holds, take the new i, k to be $j, n+1$. Else take the old i, k.

Depending on how the existence of a maximal end segment was proved, we obtain a quadratic or a linear algorithm. If we consider the first proof involving induction on the auxiliary variable m, we obtain a quadratic algorithm. The reason is that the computational content of $\mathtt{L}$ involves an

additional recursion, since L was proved by induction on m. The two nested recursions then give a quadratic algorithm.

Now how could the better proof be found by decoration? We have

$\mathtt{L}\colon \forall_n \exists_{j\le n+1} \forall_{j'\le n+1}(\mathrm{seg}(j', n+1) \le \mathrm{seg}(j, n+1))$,

$\mathtt{LMon}\colon \forall_n^{\mathrm{nc}}(\mathtt{ES}_n \to \mathtt{Mon} \to \exists_{j\le n+1} \forall_{j'\le n+1}(\mathrm{seg}(j', n+1) \le \mathrm{seg}(j, n+1)))$.

The decoration algorithm arrives at L with

$$\exists_{j\le n+1} \forall_{j'\le n+1}(\mathrm{seg}(j', n+1) \le \mathrm{seg}(j, n+1)).$$

LMon fits as well, its assumptions $\mathtt{ES}_n$ and Mon are in the context, and it has the less extended $\forall_n^{\mathrm{nc}}$ rather than $\forall_n$, hence is preferred.

References

Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science*, 8(1): 1–36, 2012.

Steven Awodey and Andrej Bauer. Propositions as [types]. *J. Log. and Comput.*, 14(4):447–471, 2004.

Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, January 1985.

Ulrich Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 91–106. Springer Verlag, Berlin, Heidelberg, New York, 1993.

Ulrich Berger. Uniform Heyting arithmetic. *Annals of Pure and Applied Logic*, 133:125–148, 2005.

Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer Verlag, Berlin, Heidelberg, New York, 2004.

Gerhard Gentzen. Untersuchungen über das logische Schließen I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

Andrey N. Kolmogorov. Zur Deutung der intuitionistischen Logik. *Math. Zeitschr.*, 35:58–65, 1932.

Helmut Schwichtenberg and Stanley S. Wainer. *Proofs and Computations*. Perspectives in Logic. Association for Symbolic Logic and Cambridge University Press, 2012.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

Mathematisches Institut der LMU, München, Germany