Decorating proofs[1]

Diana Ratiu and Helmut Schwichtenberg

ABSTRACT. The programs synthesized from proofs are guaranteed to be correct, however at the cost of sometimes introducing irrelevant computations, as a consequence of the fact that the extracted code faithfully reflects the proof. In this paper we extend the work of Ulrich Berger [2], which introduces the concept of "non-computational universal quantifiers", and propose an algorithm by which we identify at the proof level the components - quantified variables, as well as premises of implications - that are computationally irrelevant and mark them as such. We illustrate the benefits of this (optimal) decorating algorithm in some case studies and present the results obtained with the proof assistant Minlog.

We consider proofs in minimal logic, written in natural deduction style. The only rules are introduction and elimination for implication and the universal quantifier. The logical connectives $\exists, \wedge$ are seen as special cases of inductively defined predicates, and hence are defined by the introduction and elimination schemes

$$\forall_x(A \to \exists_x A), \qquad A \to B \to A \wedge B,$$
$$\exists_x A \to \forall_x(A \to B) \to B \ (x \text{ not free in } B), \qquad A \wedge B \to (A \to B \to C) \to C.$$

Disjunction can be defined by $A \vee B := \exists_p((p \to A) \wedge (\neg p \to B))$ with p a boolean variable. When the computational content of a proof is of interest, it is appropriate to distinguish between computational and non-computational variants of $\to, \forall$, written $\to^{\mathrm{c}}, \forall^{\mathrm{c}}$ and $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$, respectively; for $\forall^{\mathrm{nc}}$ this was first done by Berger [2]. The introduction rules for $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$ then need an additional restriction: the abstracted (assumption or object) variable is not allowed to be "computational", which can be defined to mean "not free in the extracted term of the premise proof". The insertion of such marks is called a "decoration" of the proof.

In the present paper we are interested in "fine-tuning" the computational content of proofs, by inserting decorations. After adapting in section 1 the standard theory of proof interpretation by (modified) realizability, in section 2 we define what a computational strengthening of a decorated formula is, and construct a derivation of $A_1 \to^{\mathrm{c}} A_2$ for A_1 a computational strengthening of A_2. Here is an example (due to Robert Constable) of why this is of interest. Suppose that in a proof M of a formula C we have made use of a case distinction based on an auxiliary lemma stating a disjunction, say $L\colon A \vee B$. Then the extract $[\![M]\!]$ will contain the extract $[\![L]\!]$ of the proof of the auxiliary lemma,

[1] Dedicated to Grigori Mints on occasion of his 70th birthday

which may be large. Now suppose further that in the proof M of C, the only computationally relevant use of the lemma was which one of the two alternatives holds true, A or B. We can express this fact by using a weakened form of the lemma instead: $L' \colon A \vee^{\mathrm{u}} B$. Since the extract $[\![L']\!]$ is a boolean, the extract of the modified proof has been "purified" in the sense that the (possibly large) extract $[\![L]\!]$ has disappeared.

In section 3 we consider the question of "optimal" decorations of proofs: suppose we are given an undecorated proof, and a decoration of its end formula. The task then is to find a decoration of the whole proof (including a further decoration of its end formula) in such a way that any other decoration "extends" this one. Here "extends" just means that some connectives have been changed into their more informative versions, disregarding polarities. We show that such an optimal decoration exists, and give an algorithm to construct it.

The final section 4 first takes up the example of list reversal used by Berger [3] to demonstrate that usage of $\forall^{\mathrm{nc}}$ rather than $\forall^{\mathrm{c}}$ can significantly reduce the complexity of extracted programs, in this case from quadratic to linear. Our implementation (cf. `http://www.minlog-system.de`) of the decoration algorithm from section 3 automatically finds the optimal decoration. A similar application of decoration occurs when one derives double induction (recurring to two predecessors) in continuation passing style, i.e., not directly, but using as an intermediate assertion (proved by induction)

$$\forall^{\mathrm{c}}_{n,m}((Qn \to^{\mathrm{c}} Q(\mathrm{S}n) \to^{\mathrm{c}} Q(n+m)) \to^{\mathrm{c}} Q0 \to^{\mathrm{c}} Q1 \to^{\mathrm{c}} Q(n+m)).$$

After decoration, the formula becomes

$$\forall^{\mathrm{c}}_{n}\forall^{\mathrm{nc}}_{m}((Qn \to^{\mathrm{c}} Q(\mathrm{S}n) \to^{\mathrm{c}} Q(n+m)) \to^{\mathrm{c}} Q0 \to^{\mathrm{c}} Q1 \to^{\mathrm{c}} Q(n+m)).$$

This is applied (as in [5]) to obtain a continuation based tail recursive definition of the Fibonacci function, from a proof of its totality. We finally give two more applications of a slightly extended version of the decoration algorithm involving a data base of proven theorems and proof transformations. The first one concerns Constable's example above, and the second one the maximal segment problem studied in [1].

1 Computational content of proofs

Computational content in proofs arises when the formula proved contains a strictly positive occurrence of an existential quantifier, as in $\forall_x \exists_y A(x,y)$. If $A(x,y)$ is quantifier-free, the computational content of the proof consists of a function assigning to every number n a number m such that $A(n,m)$ holds. One can view $\exists_y A(x,y)$ as a (somewhat degenerated) inductively defined predicate, whose single clause is $\forall_{x,y}(A(x,y) \to \exists_y A(x,y))$. More generally, the computational content of a proof of $I\vec{r}$ for an inductively defined predicate I is a "generation tree", witnessing how the arguments $\vec{r}$ were put into I. For example, consider the clauses $\mathrm{Even}(0)$ and $\forall_n(\mathrm{Even}(n) \to \mathrm{Even}(\mathrm{S}(\mathrm{S}n)))$. A generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

It is a tempting idea that one should be able in such cases to switch on or off the computational influence of a universally quantified variable (as in [2])

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.)
$\dfrac{\begin{array}{cc}\mid M & \\ \forall_x A(x) & r\end{array}}{A(r)}\,\forall^{-}$	$(M^{\forall_x A(x)} r)^{A(r)}$

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

or of the premise of an implication. For instance, in the clause $\forall_n(\mathrm{Even}(n) \to \mathrm{Even}(\mathrm{S}(\mathrm{S}n)))$ only the premise $\mathrm{Even}(n)$ should be computationally relevant, not the quantifier $\forall_n$. We therefore "decorate" $\to$ and write $\to^{\mathrm{c}}$, but leave the quantifier $\forall_n$ undecorated, i.e., consider it as "non-computational" and write $\forall^{\mathrm{nc}}$. In the clause $\forall_x(A \to \exists_x A)$ for an existential quantifier $\exists_x A$ one may decorate the quantifier $\forall_x$ and the implication $\to$ independently, which gives four (only) computationally different variants $\exists^{\mathrm{d}}, \exists^{\mathrm{l}}, \exists^{\mathrm{r}}, \exists^{\mathrm{u}}$ (with d, l, r, u for "double", "left", "right", "uniform") of the existential quantifier, defined below.

Incorporating these two computationally different variants of implication and universal quantification into Gentzen's calculus of natural deduction is rather easy. Recall the introduction and elimination rules for $\to$ and $\forall$, written in the standard tree notation as well as in a (more condensed) linear term notation, in figure 1. For the universal quantifier $\forall$ there is an introduction rule $\forall^{+}x$ and an elimination rule $\forall^{-}$, whose right premise is the term r to be substituted. The rule $\forall^{+}x$ is subject to an *(Eigen-) variable condition*: The derivation term

M of the premise A should not contain any open assumption with x as a free variable.

For $\to^{\mathrm{c}}$ and $\forall^{\mathrm{c}}$ no changes are necessary, and for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$ only the introduction rules need to be adapted: in "computational relevant" parts of the derivation the abstracted (assumption or object) variable needs to be "non-computational". This will be defined later, simultaneously with the notion of an "extracted term" of a derivation.

For $\exists, \wedge$ we have a whole zoo of (only) computationally different variants: $\exists^{\mathrm{d}}, \exists^{\mathrm{l}}, \exists^{\mathrm{r}}, \exists^{\mathrm{u}}, \wedge^{\mathrm{d}}, \wedge^{\mathrm{l}}, \wedge^{\mathrm{r}}, \wedge^{\mathrm{u}}$ ($\exists^{\mathrm{r}}$ already appeared in [3]). They are defined by their introduction and elimination axioms, which involve both $\to^{\mathrm{c}}$, $\forall^{\mathrm{c}}$ and $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$. For the first four these are

$$\begin{array}{ll}
\forall^{\mathrm{c}}_x(A \to^{\mathrm{c}} \exists^{\mathrm{d}}_x A), & \exists^{\mathrm{d}}_x A \to^{\mathrm{c}} \forall^{\mathrm{c}}_x(A \to^{\mathrm{c}} B) \to^{\mathrm{c}} B,\\
\forall^{\mathrm{c}}_x(A \to^{\mathrm{nc}} \exists^{\mathrm{l}}_x A), & \exists^{\mathrm{l}}_x A \to^{\mathrm{c}} \forall^{\mathrm{c}}_x(A \to^{\mathrm{nc}} B) \to^{\mathrm{c}} B,\\
\forall^{\mathrm{nc}}_x(A \to^{\mathrm{c}} \exists^{\mathrm{r}}_x A), & \exists^{\mathrm{r}}_x A \to^{\mathrm{c}} \forall^{\mathrm{nc}}_x(A \to^{\mathrm{c}} B) \to^{\mathrm{c}} B,\\
\forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} \exists^{\mathrm{u}}_x A), & \exists^{\mathrm{u}}_x A \to^{\mathrm{c}} \forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} B) \to^{\mathrm{c}} B
\end{array}$$

where x is not free in B, and similar for $\wedge$:

$$\begin{array}{ll}
A \to^{\mathrm{c}} B \to^{\mathrm{c}} A \wedge^{\mathrm{d}} B, & A \wedge^{\mathrm{d}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} B \to^{\mathrm{c}} C) \to^{\mathrm{c}} C,\\
A \to^{\mathrm{c}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{l}} B, & A \wedge^{\mathrm{l}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} B \to^{\mathrm{nc}} C) \to^{\mathrm{c}} C,\\
A \to^{\mathrm{nc}} B \to^{\mathrm{c}} A \wedge^{\mathrm{r}} B, & A \wedge^{\mathrm{r}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} B \to^{\mathrm{c}} C) \to^{\mathrm{c}} C,\\
A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{u}} B, & A \wedge^{\mathrm{u}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} C) \to^{\mathrm{c}} C.
\end{array}$$

Types ρ, σ, τ are generated from ground types like the types $\mathbf{N}$ of natural numbers and $\mathbf{B}$ of booleans by forming list types $\mathbf{L}(\rho)$, product types $\rho \times \sigma$ and function types $\rho \to \sigma$. The types $\mathbf{N}$, $\mathbf{B}$, $\mathbf{L}(\rho)$ and $\rho \times \sigma$ can be viewed as special cases of the formation of free algebras (with parameters); however, for simplicity we only consider these cases here.

Terms r, s, t of a given type are built from (typed) variables and (typed) constants by (type-correct) abstraction and application. The *projections* of a term r of product type $\rho \times \sigma$ are written as $r0$ and $r1$.

Kolmogorov [9] proposed to view a formula A as a "computational problem", of type $\tau(A)$, the type of a potential solution or "realizer" of A. More precisely, $\tau(A)$ should be the type of the term (or "program") to be extracted from a proof of A. Formally, we assign to every formula A an object $\tau(A)$ (a type or the "nulltype" symbol $\circ$). In case $\tau(A) = \circ$ proofs of A have no computational content; such formulas A are called *computationally irrelevant* (c.i.) (or *Harrop* formulas); the other ones are called *computationally relevant* (c.r.). The definition can be conveniently written if we extend the use of $\rho \to \sigma$ and $\rho \times \sigma$ to the nulltype symbol $\circ$:

$$\begin{array}{l}
(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ,\\
(\rho \times \circ) := \rho, \quad (\circ \times \sigma) := \sigma, \quad (\circ \times \circ) := \circ.
\end{array}$$

With this understanding of $\rho \to \sigma$ and $\rho \times \sigma$ we can simply write

$\tau(P) := \circ$ for P a decidable prime formula (given by a boolean term),

$\tau(A \to^{\mathrm{c}} B) := \tau(A) \to \tau(B), \quad \tau(A \to^{\mathrm{nc}} B) := \tau(B),$
$\tau(\forall^{\mathrm{c}}_{x^\rho} A) := \rho \to \tau(A), \quad \tau(\forall^{\mathrm{nc}}_{x^\rho} A) := \tau(A),$
$\tau(\exists^{\mathrm{d}}_{x^\rho} A) := \rho \times \tau(A), \quad \tau(\exists^{\mathrm{l}}_{x^\rho} A) := \rho, \quad \tau(\exists^{\mathrm{r}}_{x^\rho} A) := \tau(A), \quad \tau(\exists^{\mathrm{u}}_{x^\rho} A) := \circ,$
$\tau(A \wedge^{\mathrm{d}} B) := \tau(A) \times \tau(B), \quad \tau(A \wedge^{\mathrm{l}} B) := \tau(A), \quad \tau(A \wedge^{\mathrm{r}} B) := \tau(B),$
$\tau(A \wedge^{\mathrm{u}} B) := \circ.$

We now define *realizability*. It will be convenient to introduce a special "nullterm" symbol ε to be used as a "realizer" for c.i. formulas. We extend term application to the nullterm symbol by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

DEFINITION 1 (t realizes A). Let A be a formula and t either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε if for c.i. A. We use P for decidable prime formulas (given by a boolean term).

$$\begin{aligned}
&\varepsilon\ \mathbf{r}\ P && := P\\
&t\ \mathbf{r}\ (A \to^{\mathrm{c}} B) && := \forall^{\mathrm{nc}}_x(x\ \mathbf{r}\ A\ \to^{\mathrm{nc}}\ tx\ \mathbf{r}\ B),\\
&t\ \mathbf{r}\ (A \to^{\mathrm{nc}} B) && := \forall^{\mathrm{nc}}_x(x\ \mathbf{r}\ A\ \to^{\mathrm{nc}}\ t\ \mathbf{r}\ B),\\
&t\ \mathbf{r}\ (\forall^{\mathrm{c}}_x A) && := \forall^{\mathrm{nc}}_x(tx\ \mathbf{r}\ A),\\
&t\ \mathbf{r}\ (\forall^{\mathrm{nc}}_x A) && := \forall^{\mathrm{nc}}_x(t\ \mathbf{r}\ A).
\end{aligned}$$

In case A is c.i., $\forall^{\mathrm{nc}}_x(x\ \mathbf{r}\ A \to^{\mathrm{nc}} B(x))$ means $\varepsilon\ \mathbf{r}\ A \to^{\mathrm{nc}} B(\varepsilon)$. For the other connectives realizability is defined by

$$t\ \mathbf{r}\ \exists^{\mathrm{d}}_x A(x) := \begin{cases} t1\ \mathbf{r}\ A(t0) & \text{if } A \text{ is c.r.} \\ \varepsilon\ \mathbf{r}\ A(t) & \text{otherwise,} \end{cases} \quad t\ \mathbf{r}\ \exists^{\mathrm{l}}_x A(x) := \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ A(t)),$$
$$t\ \mathbf{r}\ \exists^{\mathrm{r}}_x A(x) := \exists^{\mathrm{u}}_x(t\ \mathbf{r}\ A(x)), \quad \varepsilon\ \mathbf{r}\ \exists^{\mathrm{u}}_x A(x) := \exists^{\mathrm{u}}_{x,y}(y\ \mathbf{r}\ A(x)),$$
$$t\ \mathbf{r}\ (A \wedge^{\mathrm{d}} B) := t0\ \mathbf{r}\ A \wedge t1\ \mathbf{r}\ B, \quad t\ \mathbf{r}\ (A \wedge^{\mathrm{l}} B) := t\ \mathbf{r}\ A \wedge \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ B),$$
$$t\ \mathbf{r}\ (A \wedge^{\mathrm{r}} B) := \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ A) \wedge t\ \mathbf{r}\ B,$$
$$\varepsilon\ \mathbf{r}\ (A \wedge^{\mathrm{u}} B) := \exists^{\mathrm{u}}_y(y\ \mathbf{r}\ A) \wedge \exists^{\mathrm{u}}_z(z\ \mathbf{r}\ B).$$

Call two formulas A and A' *computationally equivalent* if each of them computationally implies the other, and in addition the identity realizes each of the two derivations of $A' \to^{\mathrm{c}} A$ and of $A \to^{\mathrm{c}} A'$. It is an easy exercise to verify that for c.i. A, the formulas $A \to^{\mathrm{c}} B$ and $A \to^{\mathrm{nc}} B$ are computationally equivalent, and hence can be identified. In the sequel we shall simply write $A \to B$ for either of them. Similarly, for c.i. A the two formulas $\forall^{\mathrm{c}}_x A$ and $\forall^{\mathrm{nc}}_x A$ are c.i., and both $\varepsilon\ \mathbf{r}\ \forall^{\mathrm{c}}_x A$ and $\varepsilon\ \mathbf{r}\ \forall^{\mathrm{nc}}_x A$ are defined to be $\forall^{\mathrm{nc}}_x(\varepsilon\ \mathbf{r}\ A)$. Hence they can be identified as well, and we shall simply write $\forall_x A$ for either of them. Since the formula $t\ \mathbf{r}\ A$ is c.i., under this convention the $\to, \forall$-cases in the definition of realizability can be written

$$\begin{aligned}
&t\ \mathbf{r}\ (A \to^{\mathrm{c}} B) && := \forall_x(x\ \mathbf{r}\ A\ \to\ tx\ \mathbf{r}\ B),\\
&t\ \mathbf{r}\ (A \to^{\mathrm{nc}} B) && := \forall_x(x\ \mathbf{r}\ A\ \to\ t\ \mathbf{r}\ B),
\end{aligned}$$

$$\begin{aligned} t\ \mathbf{r}\ (\forall^{\mathrm{c}}_x A) &:= \forall_x(tx\ \mathbf{r}\ A),\\ t\ \mathbf{r}\ (\forall^{\mathrm{nc}}_x A) &:= \forall_x(t\ \mathbf{r}\ A). \end{aligned}$$

Notice that the formula $t\ \mathbf{r}\ A$ is "invariant" in the sense that $\varepsilon\ \mathbf{r}\ (t\ \mathbf{r}\ A)$ and $t\ \mathbf{r}\ A$ are identical formulas.

We simultaneously define (1) *derivations* M, and (2) the *extracted term* $[\![M]\!]$ of type $\tau(A)$ of a derivation M with endformula A. As already mentioned, this simultaneous definition is needed because the introduction rules for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$ must be restricted: in "computational relevant" parts of the derivation the abstracted (assumption or object) variable needs to be "non-computational".

For derivations M^A with A c.i. let $[\![M^A]\!] := \varepsilon$; there is no condition on the usage of introduction rules for $\to^{\mathrm{nc}}$ and $\forall^{\mathrm{nc}}$ in such derivations. Now assume that M derives a c.r. formula. Then

$$\begin{aligned} [\![u^A]\!] &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated with } u^A),\\ [\![(\lambda_{u^A} M^B)^{A\to^{\mathrm{c}}B}]\!] &:= \lambda_{x_u^{\tau(A)}}[\![M]\!],\\ [\![(M^{A\to^{\mathrm{c}}B}N^A)^B]\!] &:= [\![M]\!][\![N]\!],\\ [\![(\lambda_{x^\rho} M^A)^{\forall^{\mathrm{c}}_x A}]\!] &:= \lambda_{x^\rho}[\![M]\!],\\ [\![(M^{\forall^{\mathrm{c}}_x A(x)}r)^{A(r)}]\!] &:= [\![M]\!]r,\\ [\![(\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}]\!] &:= [\![(M^{A\to^{\mathrm{nc}}B}N^A)^B]\!] := [\![(\lambda_{x^\rho} M^A)^{\forall^{\mathrm{nc}}_x A}]\!]\\ &:= [\![(M^{\forall^{\mathrm{nc}}_x A(x)}r)^{A(r)}]\!] := [\![M]\!]. \end{aligned}$$

Here $\lambda_{x_u^{\tau(A)}}[\![M]\!]$ means just $[\![M]\!]$ if A is c.i.

In all these cases the correctness of the derivation is preserved, except possibly in those involving λ: $(\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}$ is correct if M is and in case A is c.r., $x_u \notin \mathrm{FV}([\![M]\!])$. $(\lambda_{x^\rho} M^A)^{\forall^{\mathrm{nc}}_x A}$ is correct if M is and – in addition to the usual variable condition – $x \notin \mathrm{FV}([\![M]\!])$.

It remains to define extracted terms for the axioms. Consider for example the induction schema for the type $\mathbf{L}(\rho)$ of lists of type ρ objects. It is written here in the form

$$\forall^{\mathrm{c}}_v(A(\mathrm{nil}) \to^{\mathrm{c}} \forall^{\mathrm{c}}_{x,v}(A(v) \to^{\mathrm{c}} A(xv)) \to^{\mathrm{c}} A(v)),$$

with x, v variables of type $\rho, \mathbf{L}(\rho)$ and xv denoting $\mathrm{cons}(x, v)$; clearly the decoration is appropriate given the computational meaning of induction. The extracted term is the corresponding recursion operator $\mathcal{R}^{\tau}_{\mathbf{L}(\rho)}$ (in the sense of Gödel [8]), of type

$$\mathbf{L}(\rho) \to \tau \to (\rho \to \mathbf{L}(\rho) \to \tau \to \tau) \to \tau.$$

Notice that a different decoration of induction is possible:

$$\forall^{\mathrm{c}}_v(A(\mathrm{nil}) \to^{\mathrm{c}} \forall^{\mathrm{nc}}_{x,v}(A(v) \to^{\mathrm{nc}} A(xv)) \to^{\mathrm{c}} A(v)).$$

The computational meaning then is case distinction (whether a list is empty or not) rather than recursion. This arises naturally if induction is introduced

as elimination scheme for an inductive definition of totality, from a different decoration of the clauses.

For the introduction and elimination axioms for $\exists^{\mathrm{d}}$, $\exists^{\mathrm{l}}$, $\exists^{\mathrm{r}}, \exists^{\mathrm{u}}$, $\wedge^{\mathrm{d}}$, $\wedge^{\mathrm{l}}$, $\wedge^{\mathrm{r}}$, $\wedge^{\mathrm{u}}$ the extracted terms are rather obvious and not spelled out here.

THEOREM 2 (Soundness). *Let M be a derivation of A from assumptions $u_i\colon C_i$ $(i < n)$. Then we can derive $[\![M]\!]$ $\mathbf{r}$ A from assumptions x_{u_i} $\mathbf{r}$ C_i (with $x_{u_i} := \varepsilon$ in case C_i is c.i.).*

2 Computational strengthening

Formulas can be decorated in many different ways, and it is a natural question to ask when one such decoration A' is "stronger" than another one A, in the sense that the former computationally implies the latter, i.e., $\vdash A' \to^{\mathrm{c}} A$. We give a partial answer to this question in the proposition below.

We define a relation $A' \sqsupseteq A$ (A' is a *computational strengthening* of A) between c.r. formulas A', A inductively. It is reflexive, transitive and satisfies

$$
\begin{aligned}
&(A \to^{\mathrm{nc}} B) \sqsupseteq (A \to^{\mathrm{c}} B),\\
&(A \to^{\mathrm{c}} B) \sqsupseteq (A \to^{\mathrm{nc}} B) \quad \text{if } A \text{ is c.i.},\\
&(A \to B') \sqsupseteq (A \to B) \quad \text{if } B' \sqsupseteq B, \text{ with } \to \in \{\to^{\mathrm{c}}, \to^{\mathrm{nc}}\},\\
&(A \to B) \sqsupseteq (A' \to B) \quad \text{if } A' \sqsupseteq A, \text{ with } \to \in \{\to^{\mathrm{c}}, \to^{\mathrm{nc}}\},\\
&\forall^{\mathrm{nc}}_x A \sqsupseteq \forall^{\mathrm{c}}_x A,\\
&\forall_x A' \sqsupseteq \forall_x A \quad \text{if } A' \sqsupseteq A, \text{ with } \forall \in \{\forall^{\mathrm{c}}, \forall^{\mathrm{nc}}\}.
\end{aligned}
$$

Moreover, for the defined $\exists^{\mathrm{d}}, \exists^{\mathrm{l}}, \exists^{\mathrm{r}}, \wedge^{\mathrm{d}}, \wedge^{\mathrm{l}}, \wedge^{\mathrm{r}}$ it satisfies

$$
\begin{aligned}
&\exists^{\mathrm{d}}_x A \sqsupseteq \exists^{\mathrm{l}}_x A, \exists^{\mathrm{r}}_x A,\\
&\exists_x A' \sqsupseteq \exists_x A \quad \text{if } A' \sqsupseteq A, \text{ with } \exists \in \{\exists^{\mathrm{d}}, \exists^{\mathrm{l}}, \exists^{\mathrm{r}}\},\\
&(A \wedge^{\mathrm{d}} B) \sqsupseteq (A \wedge^{\mathrm{l}} B), (A \wedge^{\mathrm{r}} B),\\
&(A' \wedge B) \sqsupseteq (A \wedge B) \quad \text{if } A' \sqsupseteq A, \text{ with } \wedge \in \{\wedge^{\mathrm{d}}, \wedge^{\mathrm{l}}, \wedge^{\mathrm{r}}\},\\
&(A \wedge B') \sqsupseteq (A \wedge B) \quad \text{if } B' \sqsupseteq B, \text{ with } \wedge \in \{\wedge^{\mathrm{d}}, \wedge^{\mathrm{l}}, \wedge^{\mathrm{r}}\}.
\end{aligned}
$$

PROPOSITION 3. *If $A' \sqsupseteq A$, then $\vdash A' \to^{\mathrm{c}} A$.*

Proof. We show that the relation "$\vdash A' \to^{\mathrm{c}} A$" has the same closure properties as "$A' \sqsupseteq A$". For reflexivity and transitivity this is clear. For the rest we give some sample derivations.

$$
\dfrac{\dfrac{A \to^{\mathrm{nc}} B \qquad u\colon A}{B}}{A \to^{\mathrm{c}} B}\,(\to^{\mathrm{c}})^+, u
\qquad\qquad
\dfrac{\dfrac{\begin{array}{c}\mid \text{assumed}\\ B' \to^{\mathrm{c}} B\end{array} \qquad \dfrac{A \to^{\mathrm{nc}} B' \qquad u\colon A}{B'}}{B}}{A \to^{\mathrm{nc}} B}\,(\to^{\mathrm{nc}})^+, u
$$

where in the last derivation the final $(\to^{\mathrm{nc}})^+$-application is correct since u is

not a computational assumption variable in the premise derivation of B.

$$\dfrac{\dfrac{A \to^{\mathrm{nc}} B \qquad \dfrac{\mid \text{assumed}}{A' \to^{\mathrm{c}} A} \qquad u\colon A'}{A}}{\dfrac{B}{A' \to^{\mathrm{nc}} B}\,(\to^{\mathrm{nc}})^+, u}$$

where for the same reason the final $(\to^{\mathrm{nc}})^+$-application is correct. ■

3 Optimal decorations

We denote the *sequent* of a proof M by $\mathrm{Seq}(M)$; it 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. formulas of M (i.e., those not above a c.i. 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 $\forall^{\mathrm{c}}_n(P0 \to^{\mathrm{c}} \forall^{\mathrm{c}}_n(Pn \to^{\mathrm{c}} P(\mathrm{S}n)) \to^{\mathrm{c}} Pn^{\mathbf{N}})$ with a unary predicate variable P and a predicate substitution $P \mapsto \{\, x \mid A(x) \,\}$. 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 $\to^{\mathrm{c}}$ and $\forall^{\mathrm{c}}$.

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 c.r. formula 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.

REMARK 4. Notice that if a proof N extends another one M, then $\mathrm{FV}(\llbracket N \rrbracket)$ is essentially (that is, up to extensions of assumption formulas) a superset of $\mathrm{FV}(\llbracket M \rrbracket)$. This can be proven by induction on N.

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 the decorated $\to^{\mathrm{c}}$ and $\forall^{\mathrm{c}}$, for instance induction. However, in $\forall^{\mathrm{c}}_n(A(0) \to^{\mathrm{c}} \forall^{\mathrm{c}}_n(A(n) \to^{\mathrm{c}} A(\mathrm{S}n)) \to^{\mathrm{c}} A(n^{\mathbf{N}}))$ the given extension of the four A's might be different. One needs to pick their "least upper bound" as further extension. To make this assumption true for the other axioms listed above (introduction and elimination axioms for $\exists^{\mathrm{d}}, \exists^{\mathrm{l}}, \exists^{\mathrm{r}}, \exists^{\mathrm{u}}, \wedge^{\mathrm{d}}, \wedge^{\mathrm{l}}, \wedge^{\mathrm{r}}, \wedge^{\mathrm{u}}$) we also add all their extensions as axioms.

We will define a *decoration algorithm*, assigning to every proof pattern U and every extension of its sequent an "optimal" decoration M_∞ of U, which further extends the given extension of its sequent.

THEOREM 5. *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_∞*.*

Proof. By induction on derivations. It suffices to consider derivations with a c.r. endformula. For axioms the validity of the claim was assumed, and for assumption variables it is clear.

Case $(\to^{\mathrm{nc}})^+$. Consider the proof pattern

$$\dfrac{\begin{array}{c}\Gamma, u\colon A\\ \mid U\\ B\end{array}}{A \to^{\mathrm{nc}} B}\,(\to^{\mathrm{nc}})^+, u$$

with a given extension $\Delta \Rightarrow C \to^{\mathrm{nc}} D$ or $\Delta \Rightarrow C \to^{\mathrm{c}} D$ of its sequent $\Gamma \Rightarrow A \to^{\mathrm{nc}} B$. Applying the induction hypothesis for U with sequent $\Delta, C \Rightarrow D$, one obtains a decoration M_∞ of U whose sequent $\Delta_1, C_1 \Rightarrow D_1$ extends $\Delta, C \Rightarrow D$. Now apply $(\to^{\mathrm{nc}})^+$ in case the given extension is $\Delta \Rightarrow C \to^{\mathrm{nc}} D$ and $x_u \notin \mathrm{FV}([\![M_\infty]\!])$, and $(\to^{\mathrm{c}})^+$ otherwise.

For (b) consider a decoration $\lambda_u M$ of $\lambda_u U$ whose sequent extends the given extended sequent $\Delta \Rightarrow C \to^{\mathrm{nc}} D$ or $\Delta \Rightarrow C \to^{\mathrm{c}} D$. Clearly the sequent $\mathrm{Seq}(M)$ of its premise extends $\Delta, C \Rightarrow D$. Then M extends M_∞ by induction hypothesis for U. If $\lambda_u M$ derives a non-computational implication then the given extended sequent must be of the form $\Delta \Rightarrow C \to^{\mathrm{nc}} D$ and $x_u \notin \mathrm{FV}([\![M]\!])$, hence $x_u \notin \mathrm{FV}([\![M_\infty]\!])$. But then by construction we have applied $(\to^{\mathrm{nc}})^+$ to obtain $\lambda_u M_\infty$. Hence $\lambda_u M$ extends $\lambda_u M_\infty$. If $\lambda_u M$ does not derive a non-computational implication, the claim follows immediately.

Case $(\to^{\mathrm{nc}})^-$. Consider a proof pattern

$$\dfrac{\begin{array}{c}\Phi, \Gamma\\ \mid U\\ A \to^{\mathrm{nc}} B\end{array}\qquad\begin{array}{c}\Gamma, \Psi\\ \mid V\\ A\end{array}}{B}\,(\to^{\mathrm{nc}})^-$$

We are given an extension $\Pi, \Delta, \Sigma \Rightarrow D$ of $\Phi, \Gamma, \Psi \Rightarrow B$. Then we proceed in alternating steps, applying the induction hypothesis to U and V.

(1) The induction hypothesis for U for the extension $\Pi, \Delta \Rightarrow A \to^{\mathrm{nc}} D$ of its sequent gives a decoration M_1 of U whose sequent $\Pi_1, \Delta_1 \Rightarrow C_1 \to D_1$ extends $\Pi, \Delta \Rightarrow A \to^{\mathrm{nc}} D$, where $\to$ means $\to^{\mathrm{nc}}$ or $\to^{\mathrm{c}}$. This already suffices if A is c.i., since then the extension $\Delta_1, \Sigma \Rightarrow C_1$ of V is a correct proof (recall that in c.i. parts of a proof decorations of implications and universal quantifiers can be ignored). If A is c.r.:

(2) The induction hypothesis for V for the extension $\Delta_1, \Sigma \Rightarrow C_1$ of its sequent gives a decoration N_2 of V whose sequent $\Delta_2, \Sigma_2 \Rightarrow C_2$ extends $\Delta_1, \Sigma \Rightarrow C_1$.

(3) The induction hypothesis for U for the extension $\Pi_1, \Delta_2 \Rightarrow C_2 \to D_1$ of its sequent gives a decoration M_3 of U whose sequent $\Pi_3, \Delta_3 \Rightarrow C_3 \to D_3$ extends $\Pi_1, \Delta_2 \Rightarrow C_2 \to D_1$.

(4) The induction hypothesis for V for the extension $\Delta_3, \Sigma_2 \Rightarrow C_3$ of its sequent gives a decoration N_4 of V whose sequent $\Delta_4, \Sigma_4 \Rightarrow C_4$ extends $\Delta_3, \Sigma_2 \Rightarrow C_3$. This process is repeated until in V no further proper extension of Δ_3 and C_3 is returned. Such a situation will always be reached since there is a maximal extension, where all connectives are maximally decorated. But then we easily obtain (a): Assume that in (4) we have $\Delta_4 = \Delta_3$ and $C_4 = C_3$. Then the decoration

$$\dfrac{\begin{array}{cc}\Pi_3, \Delta_3 & \Delta_4, \Sigma_4 \\ \mid M_3 & \mid N_4 \\ C_3 \to D_3 & C_4\end{array}}{D_3}\,{\to}^-$$

of UV derives a sequent $\Pi_3, \Delta_3, \Sigma_4 \Rightarrow D_3$ extending $\Pi, \Delta, \Sigma \Rightarrow D$.

For (b) we need to consider a decoration MN of UV whose sequent $\mathrm{Seq}(MN)$ extends the given extension $\Pi, \Delta, \Sigma \Rightarrow D$ of $\Phi, \Gamma, \Psi \Rightarrow B$. We must show that MN extends M_3N_4. To this end we go through the alternating steps again.

(1) Since the sequent $\mathrm{Seq}(M)$ extends $\Pi, \Delta \Rightarrow A \to^{\mathrm{nc}} D$, the induction hypothesis for U for the extension $\Delta \Rightarrow A \to^{\mathrm{nc}} D$ of its sequent ensures that M extends M_1.

(2) Since then the sequent $\mathrm{Seq}(N)$ extends $\Delta_1, \Sigma \Rightarrow C_1$, the induction hypothesis for V for the extension $\Delta_1, \Sigma \Rightarrow C_1$ of its sequent ensures that N extends N_2.

(3) Therefore $\mathrm{Seq}(M)$ extends the sequent $\Pi_1, \Delta_2 \Rightarrow C_2 \to D_1$, and the induction hypothesis for U for the extension $\Pi_1, \Delta_2 \Rightarrow C_2 \to D_1$ of U's sequent ensures that M extends M_3.

(4) Therefore $\mathrm{Seq}(N)$ extends $\Delta_3, \Sigma_2 \Rightarrow C_3$, and induction hypothesis for V for the extension $\Delta_3, \Sigma_2 \Rightarrow C_3$ of V's sequent ensures that N also extends N_4.

But since $\Delta_4 = \Delta_3$ and $C_4 = C_3$ by assumption, MN extends the decoration M_3N_4 of UV constructed above.

Case $(\forall^{\mathrm{nc}})^+$. Consider a proof pattern

$$\dfrac{\begin{array}{c}\Gamma \\ \mid U \\ A\end{array}}{\forall^{\mathrm{nc}}_x A}\,(\forall^{\mathrm{nc}})^+$$

with a given extension $\Delta \Rightarrow \forall^{\mathrm{nc}}_x C$ or $\Delta \Rightarrow \forall^{\mathrm{c}}_x C$ of its sequent. Applying the induction hypothesis for U with sequent $\Delta \Rightarrow C$, one obtains a decoration M_∞ of U whose sequent $\Delta_1 \Rightarrow C_1$ extends $\Delta \Rightarrow C$. Now apply $(\forall^{\mathrm{nc}})^+$ in case the given extension is $\Delta \Rightarrow \forall^{\mathrm{nc}}_x C$ and $x \notin \mathrm{FV}([\![M_\infty]\!])$, and $(\forall^{\mathrm{c}})^+$ otherwise.

For (b) consider a decoration $\lambda_x M$ of $\lambda_x U$ whose sequent extends the given extended sequent $\Delta \Rightarrow \forall^{\mathrm{nc}}_x C$ or $\Delta \Rightarrow \forall^{\mathrm{c}}_x C$. Clearly the sequent $\mathrm{Seq}(M)$ of its premise extends $\Delta \Rightarrow C$. Then M extends M_∞ by induction hypothesis for U. If $\lambda_x M$ derives a non-computational generalization, then the given

extended sequent must be of the form $\Delta \Rightarrow \forall^{\mathrm{nc}}_x C$ and $x \notin \mathrm{FV}([\![M]\!])$, hence $x \notin \mathrm{FV}([\![M_\infty]\!])$ (by the remark above). But then by construction we have applied $(\forall^{\mathrm{nc}})^+$ to obtain $\lambda_x M_\infty$. Hence $\lambda_x M$ extends $\lambda_x M_\infty$. If $\lambda_x M$ does not derive a non-computational generalization, the claim follows immediately.

Case $(\forall^{\mathrm{nc}})^-$. Consider a proof pattern

$$\dfrac{\begin{array}{c}\Gamma\\ |\, U\\ \forall^{\mathrm{nc}}_x A(x)\end{array} \qquad r}{A(r)}\,(\forall^{\mathrm{nc}})^-$$

and let $\Delta \Rightarrow C(r)$ be any extension of its sequent $\Gamma \Rightarrow A(r)$. The induction hypothesis for U for the extension $\Delta \Rightarrow \forall^{\mathrm{nc}}_x C(x)$ produces a decoration M_∞ of U whose sequent extends $\Delta \Rightarrow \forall^{\mathrm{nc}}_x C(x)$. Then apply $(\forall^{\mathrm{nc}})^-$ or $(\forall^{\mathrm{c}})^-$, whichever is appropriate, to obtain the required $M_\infty r$.

For (b) consider a decoration Mr of Ur whose sequent $\mathrm{Seq}(Mr)$ extends the given extension $\Delta \Rightarrow C(r)$ of $\Gamma \Rightarrow A(r)$. Then M extends M_∞ by induction hypothesis for U, and hence Mr extends $M_\infty r$. ■

4 Applications

4.1 Decoration of implication and conjunction

We illustrate the effects of decoration on a simple example involving implications. Consider $A \to B \to A$ with the trivial proof $M := \lambda^A_{u_1}\lambda^B_{u_2} u_1$. Clearly, one does not need the decoration of the second implication, so we apply the decoration algorithm and specify as extension of $\mathrm{Seq}(\mathrm{P}(M))$ the formula $A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} A$. The algorithm detects that the first implication needs to be decorated, since the abstracted assumption variable is computational. Since the second implication can be left undecorated, a proof of $A \to^{\mathrm{c}} B \to^{\mathrm{nc}} A$ is constructed from M.

A similar phenomenon occurs for $A \wedge^{\mathrm{d}} B \to B$. Let M be its proof and $U := \mathrm{P}(M)$ its proof pattern. When given the extension $A \wedge^{\mathrm{u}} B \to^{\mathrm{nc}} B$ for $\mathrm{Seq}(U)$, the decoration algorithm constructs a correct proof of $A \wedge^{\mathrm{r}} B \to^{\mathrm{c}} B$.

4.2 List reversal

We work in Heyting Arithmetic HA^ω for a language based on Gödel's T [8], which is finitely typed (cf. Troelstra [10] for general background). We call the formulas built from terms r of type $\mathbf{B}$ by means of a special operator $\mathrm{atom}(r^{\mathbf{B}})$ *decidable* prime formulas. They include for instance equations between terms of type $\mathbf{N}$, since the boolean-valued binary equality function $=_{\mathbf{N}} : \mathbf{N} \to \mathbf{N} \to \mathbf{B}$ can be defined by

$$\begin{aligned}&(0 =_{\mathbf{N}} 0) := \mathtt{tt}, \quad (\mathrm{S}n =_{\mathbf{N}} 0) := \mathtt{ff},\\ &(0 =_{\mathbf{N}} \mathrm{S}m) := \mathtt{ff}, \quad (\mathrm{S}n =_{\mathbf{N}} \mathrm{S}m) := (n =_{\mathbf{N}} m).\end{aligned}$$

For falsity we can take the atomic formula $\mathbf{F} := \mathrm{atom}(\mathtt{ff})$ – called *arithmetical falsity* – built from the boolean constant $\mathtt{ff}$. Since in the A-translation below we need to substitute a formula for falsity, we alternatively use a special predicate

variable $\bot$ to mark such occurrences of falsity. The *formulas* of HA^ω are built from prime formulas by the connectives $\to$ and $\forall$. We define *negation* $\neg A$ by $A \to \mathbf{F}$ or $A \to \bot$ (depending on the context), and the *weak* (or "classical") existential quantifier by

$$\tilde{\exists}_x A := \neg\forall_x \neg A.$$

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. Assuming

$$\mathrm{InitRev}\colon R(\mathrm{nil}, \mathrm{nil}), \tag{1}$$
$$\mathrm{GenRev}\colon \forall_{v,w,x}(Rvw \to R(vx, xw)) \tag{2}$$

we prove

$$(3)\quad \forall_v \tilde{\exists}_w Rvw \qquad (:= \forall_v(\forall_w(Rvw \to \bot) \to \bot)).$$

Fix v and assume $u\colon \forall_w \neg Rvw$; we need to derive a contradiction. To this end we prove that all initial segments of v are non-revertible, which contradicts (1). More precisely, from u and (2) 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 $v_2 = \mathrm{nil}$ this follows from our initial assumption u. For the step case, assume $v_1(xv_2) = v$, fix w and assume further $Rv_1 w$. We must derive a contradiction. By (2) we conclude that $R(v_1 x, xw)$. On the other hand, properties of the append function imply that $(v_1 x)v_2 = v$. The induction for $v_1 x$ gives $\forall_w \neg R(v_1 x, w)$. Taking xw for w leads to the desired contradiction.

We formalize this proof, to prepare it for decoration. The following lemmata will be used.

$$\begin{aligned}
&\mathrm{Compat}\colon \forall_P \forall_{v_1,v_2}(v_1 = v_2 \to Pv_1 \to Pv_2),\\
&\mathrm{Symm}\colon \quad \forall_{v_1,v_2}(v_1 = v_2 \to v_2 = v_1),\\
&\mathrm{Trans}\colon \quad \forall_{v_1,v_2,v_3}(v_1 = v_2 \to v_2 = v_3 \to v_1 = v_3),\\
&L_1\colon \qquad \forall_v(v = v\,\mathrm{nil}),\\
&L_2\colon \qquad \forall_{v_1,x,v_2}((v_1 x)v_2 = v_1(xv_2)),
\end{aligned}$$

The proof term is

$$M := \lambda_v \lambda_u^{\forall_w \neg Rvw}(\mathrm{Ind}_{v_2, A(v_2)} vv M_{\mathrm{Base}} M_{\mathrm{Step}}\ \mathrm{nil}\ \mathrm{T}^{\mathrm{nil}\, v = v}\ \mathrm{nil}\ \mathrm{InitRev})$$

with

$$\begin{aligned}
M_{\mathrm{Base}} &:= \lambda_{v_1} \lambda_{u_1}^{v_1 \mathrm{nil} = v}(\mathrm{Compat}\,\{\, v \mid \forall_w \neg Rvw \,\}\, vv_1\\
&\qquad (\mathrm{Symm}\, v_1 v(\mathrm{Trans}\, v_1(v_1\,\mathrm{nil})v(L_1 v_1)u_1))u),\\
M_{\mathrm{Step}} &:= \lambda_{x,v_2} \lambda_{u_0}^{A(v_2)} \lambda_{v_1} \lambda_{u_1}^{v_1(xv_2) = v} \lambda_w \lambda_{u_2}^{Rv_1 w}(\\
&\qquad u_0(v_1 x)(\mathrm{Trans}\,((v_1 x)v_2)(v_1(xv_2))v(L_2 v_1 x v_2)u_1)\\
&\qquad (xw)(\mathrm{GenRev}\, v_1 w x u_2)).
\end{aligned}$$

We now have a proof M of $\forall_v \tilde{\exists}_w Rvw$ from the clauses InitRev: D_1 and GenRev: D_2, with $D_1 := R(\mathrm{nil}, \mathrm{nil})$ and $D_2 := \forall_{v,w,x}(Rvw \to R(vx, xw))$. Using the Dragalin/Friedman [6, 7] A-translation (in its refined form of [4]) we can replace $\bot$ throughout by $\exists_w Rvw$. The end formula $\forall_v \tilde{\exists}_w Rvw := \forall_v \neg\forall_w \neg Rvw := \forall_v(\forall_w(Rvw \to \bot) \to \bot)$ is turned into $\forall_v(\forall_w(Rvw \to \exists_w Rvw) \to \exists_w Rvw)$. Since its premise is an instance of existence introduction we obtain a derivation $M^{\exists}$ of $\forall_v \exists_w Rvw$. Moreover, in this case neither the D_i nor any of the axioms used involves $\bot$ in its uninstantiated formulas, and hence the correctness of the proof is not affected by the substitution. The term `neterm` extracted in Minlog from a formalization of the proof above is (after "animating" `Compat`)

```
[v0]
 (Rec list nat=>list nat=>list nat=>list nat)v0([v1,v2]v2)
 ([x1,v2,g3,v4,v5]g3(v4:+:x1:)(x1::v5))
 (Nil nat)
 (Nil nat)
```

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

$$h(\mathrm{nil}, v_2, v_3) := v_3, \qquad h(xv_1, v_2, v_3) := h(v_1, v_2 x, xv_3)$$

and gives the result by applying h to the original list and twice nil.

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 $v_2 x$ is computed, and the list append function is defined by recursion on its first argument. We will be able to get rid of this superfluous second argument by decorating the proof. It will turn out that in the proof (by induction on v_2) of the auxiliary formula $A(v_2) := \forall_{v_1}(v_1 v_2 = v \to \forall_w \neg Rv_1 w))$, the variable v_1 is not used computationally. Hence, in the decorated version of the proof, we can use $\forall^{\mathrm{nc}}_{v_1}$.

Let us now apply the general method of decorating proofs to the example of list reversal. To this end, we present our proof in more detail, particularly by writing proof trees with formulas. The decoration algorithm then is applied to its proof pattern with the sequent consisting of the context $R(\mathrm{nil}, \mathrm{nil})$ and $\forall^{\mathrm{nc}}_{v,w,x}(Rvw \to^{\mathrm{nc}} R(vx, xw))$ and the end formula $\forall^{\mathrm{nc}}_v \exists^{\mathrm{l}}_w Rvw$.

Rather than describing the algorithm step by step we only display the end result. Among the axioms used, the only ones in c.r. parts are Compat and list induction. They appear in the decorated proof in the form

$$\begin{aligned} &\text{Compat}\colon \forall_P \forall^{\mathrm{nc}}_{v_1,v_2}(v_1 = v_2 \to Pv_1 \to^{\mathrm{c}} Pv_2),\\ &\text{Ind}\colon \quad \forall^{\mathrm{c}}_{v_2}(A(\mathrm{nil}) \to^{\mathrm{c}} \forall^{\mathrm{c}}_{x,v_2}(A(v_2) \to^{\mathrm{c}} A(xv_2)) \to^{\mathrm{c}} A(v_2)) \end{aligned}$$

with $A(v_2) := \forall^{\mathrm{nc}}_{v_1}(v_1 v_2{=}v \to \forall^{\mathrm{c}}_w \neg^{\exists} Rv_1 w)$ and $\neg^{\exists} Rv_1 w := Rv_1 w \to \exists^{\mathrm{l}}_w Rvw$. $M^{\exists}_{\mathrm{Base}}$ is the derivation in Figure 2, where N is a derivation involving L_1 with a free assumption $u_1\colon v_1\,\mathrm{nil}{=}v$. $M^{\exists}_{\mathrm{Step}}$ is the derivation in Figure 3, where N_1 is a derivation involving L_2 with free assumption $u_1\colon v_1(xv_2){=}v$, and N_2 is one involving GenRev with the free assumption $u_2\colon Rv_1 w$.

The extracted term `neterm` then is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Compat}\quad \{\, v \mid \forall^{\mathrm{c}}_{w}\neg^{\exists}Rvw \,\}\quad v\quad v_1}{v{=}v_1 \to \forall^{\mathrm{c}}_{w}\neg^{\exists}Rvw \to^{\mathrm{c}} \forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w} \qquad \begin{matrix}[u_1\colon v_1\,\mathrm{nil}{=}v]\\ \mid N\\ v{=}v_1\end{matrix}}{\forall^{\mathrm{c}}_{w}\neg^{\exists}Rvw \to^{\mathrm{c}} \forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w} \qquad \exists^{+}\colon \forall^{\mathrm{c}}_{w}\neg^{\exists}Rvw}{\forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w}}{v_1\,\mathrm{nil} = v \to \forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w}\,(\to^{\mathrm{nc}})^{+}u_1}{\forall^{\mathrm{nc}}_{v_1}(v_1\,\mathrm{nil} = v \to \forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w) \quad (= A(\mathrm{nil}))}$$

Figure 2. The decorated base derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[u_0\colon A(v_2)]\quad v_1x}{(v_1x)v_2{=}v \to \forall^{\mathrm{c}}_{w}\neg^{\exists}R(v_1x,w)} \qquad \begin{matrix}[u_1\colon v_1(xv_2){=}v]\\ \mid N_1\\ (v_1x)v_2{=}v\end{matrix}}{\forall^{\mathrm{c}}_{w}\neg^{\exists}R(v_1x,w)} \qquad xw}{\neg^{\exists}R(v_1x,xw)} \qquad \begin{matrix}[u_2\colon Rv_1w]\\ \mid N_2\\ R(v_1x,xw)\end{matrix}}{\exists^{\mathrm{l}}_{w}Rvw}}{\neg^{\exists}Rv_1w}\,(\to^{\mathrm{nc}})^{+}u_2}{\forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w}}{v_1(xv_2){=}v \to \forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w}\,(\to^{\mathrm{nc}})^{+}u_1}{\forall^{\mathrm{nc}}_{v_1}(v_1(xv_2){=}v \to \forall^{\mathrm{c}}_{w}\neg^{\exists}Rv_1w)\ \ (=A(xv_2))}}{A(v_2) \to^{\mathrm{c}} A(xv_2)}\,(\to^{\mathrm{c}})^{+}u_0}{\forall^{\mathrm{c}}_{x,v_2}(A(v_2) \to^{\mathrm{c}} A(xv_2))}$$

Figure 3. The decorated step derivation

```
[v0]
 (Rec list nat=>list nat=>list nat)v0([v1]v1)
 ([x1,v2,f3,v4]f3(x1::v4))
 (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:"))))
; 4::3::2::1:
```

This time, the underlying algorithm defines an auxiliary function g by

$$g(\mathrm{nil}, w) := w, \qquad g(x :: v, w) := g(v, x :: w)$$

and gives the result by applying g to the original list and nil. 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.

4.3 Passing continuations

A similar application of decoration occurs when one derives double induction

$$\forall^{\mathrm{c}}_n(Qn \to^{\mathrm{c}} Q(\mathrm{S}n) \to^{\mathrm{c}} Q(\mathrm{S}(\mathrm{S}n))) \to^{\mathrm{c}} \forall^{\mathrm{c}}_n(Q0 \to^{\mathrm{c}} Q1 \to^{\mathrm{c}} Qn)$$

in continuation passing style, i.e., not directly, but using as an intermediate assertion (proved by induction)

$$\forall^{\mathrm{c}}_{n,m}((Qn \to^{\mathrm{c}} Q(\mathrm{S}n) \to^{\mathrm{c}} Q(n+m)) \to^{\mathrm{c}} Q0 \to^{\mathrm{c}} Q1 \to^{\mathrm{c}} Q(n+m)).$$

After decoration, the formula becomes

$$\forall^{\mathrm{c}}_n\forall^{\mathrm{nc}}_m((Qn \to^{\mathrm{c}} Q(\mathrm{S}n) \to^{\mathrm{c}} Q(n+m)) \to^{\mathrm{c}} Q0 \to^{\mathrm{c}} Q1 \to^{\mathrm{c}} Q(n+m)).$$

This can be applied to obtain a continuation based tail recursive definition of the Fibonacci function, from a proof of its totality. Let G be the graph of the Fibonacci function, defined by the clauses

$$G(0,0), \quad G(1,1),$$
$$\forall^{\mathrm{nc}}_{n,v,w}(G(n,v) \to^{\mathrm{nc}} G(\mathrm{S}n, w) \to^{\mathrm{nc}} G(\mathrm{S}(\mathrm{S}n), v+w)).$$

¿From these assumptions one can easily derive

$$\forall^{\mathrm{c}}_n\exists_v G(n,v),$$

using double induction (proved in continuation passing style). The term extracted from this proof is

```
[n0]
 (Rec nat=>nat=>(nat=>nat=>nat)=>nat=>nat=>nat)
 n0([n1,k2]k2)
 ([n1,p2,n3,k4]p2(Succ n3)([n7,n8]k4 n8(n7+n8)))
```

applied to `0`, `([n1,n2]n1)`, `0` and `1`. An unclean aspect of this term is that the recursion operator has value type

```
nat=>(nat=>nat=>nat)=>nat=>nat=>nat
```

rather than `(nat=>nat=>nat)=>nat=>nat=>nat`, which would correspond to an iteration. However, we can repair this by decoration. After (automatic) decoration of the proof, the extracted term becomes

```
[n0]
 (Rec nat=>(nat=>nat=>nat)=>nat=>nat=>nat)
 n0([k1]k1)
 ([n1,p2,k3]p2([n6,n7]k3 n7(n6+n7)))
```

applied to `([n1,n2]n1)`, `0` and `1`. This indeed is iteration in continuation passing style.

4.4 Proof transformations

In the next two examples we allow the decoration algorithm to substitute an auxiliary lemma used in the proof by a lemma that we specify explicitly. The algorithm will verify if the lemma passed to it as an argument is fitting and if this is the case, it will replace the lemma used in the original proof by the specified one. If not, the initial lemma is kept. This will allow for a certain control over the computational content, as shown by the following examples.

Our first example is an elaboration of Constable's idea described in the introduction. Let Pn mean "n is prime". Consider

$$\begin{aligned} &\forall^{\mathrm{c}}_n(Pn \vee^{\mathrm{r}} \exists^{\mathrm{d}}_{m,k>1}(n = mk)) && \text{factorization,}\\ &\forall^{\mathrm{c}}_n(Pn \vee^{\mathrm{u}} \exists^{\mathrm{d}}_{m,k>1}(n = mk)) && \text{prime number test.} \end{aligned}$$

Euler's φ-function has the properties

$$\begin{cases} \varphi(n) = n - 1 & \text{if } Pn,\\ \varphi(n) < n - 1 & \text{if } n \text{ is composed.} \end{cases}$$

Suppose that somewhat foolishly we have used factorization and these properties to obtain a proof of

$$\forall^{\mathrm{c}}_n(\varphi(n) = n - 1 \vee^{\mathrm{u}} \varphi(n) < n - 1).$$

Our goal is to get rid of the expensive factorization algorithm in the computational content, via decoration.

The decoration algorithm arrives at the factorization theorem

$$\forall^{\mathrm{c}}_n(Pn \vee^{\mathrm{r}} \exists^{\mathrm{d}}_{m,k>1}(n = mk))$$

with the decorated formula

$$\forall^{\mathrm{c}}_n(Pn \vee^{\mathrm{u}} \exists^{\mathrm{d}}_{m,k>1}(n = mk)).$$

Since the prime number test can be considered instead of the factorization lemma, we can specify that the decoration algorithm should try to replace the former by the latter. In case this is possible, a new proof is constructed, using the the prime number test lemma. Should this fail, the factorization lemma is kept. As it turns out in this case, the replacement is possible.

In the Minlog implementation the difference is clearly visible. `cL` denotes the computational content of the factorization lemma `L` (i.e., the factorization algorithm), and `cLU` the computational content of the lemma `LU` expressing the prime number test. The extract from the original proof involves computing `(cL n0)`, i.e., factorizing the argument, whereas after decoration the prime number test `cLU` suffices.

```
(pp (nt (proof-to-extracted-term proof)))
; [n0][if (cL n0) cInlOrU ([algC1]cInrOrU)]

(pp (nt (proof-to-extracted-term (decorate proof))))
; cLU
```

The second example is due to Bates and Constable [1], and deals with the "maximal segment 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 $f(i),\dots,f(k)$. 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 $\mathrm{seg}(i,k) := M(f,i,k)$.

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.

Let us be more concrete. The original specification is to find a maximal segment $x_i,\dots,x_k$, i.e.,

$$\forall_n^{\mathrm{c}} \exists_{i\leq k\leq n}^{\mathrm{d}} \forall_{i'\leq k'\leq n} (\mathrm{seg}(i',k') \leq \mathrm{seg}(i,k)).$$

A special case is to find the maximal end segment

$$\forall_n^{\mathrm{c}} \exists_{j\leq n}^{\mathrm{l}} \forall_{j'\leq n} (\mathrm{seg}(j',n) \leq \mathrm{seg}(j,n)).$$

We first provide two proofs of the existence of a maximal end segment for $n+1$

$$\forall_n^{\mathrm{c}} \exists_{j\leq n+1}^{\mathrm{l}} \forall_{j'\leq n+1} (\mathrm{seg}(j',n+1) \leq \mathrm{seg}(j,n+1)).$$

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

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

The second proof uses as assumptions an "induction hypothesis"

$$\mathtt{IH}_n\colon \exists^{\mathrm{l}}_{j\le n}\forall_{j'\le n}(\mathrm{seg}(j',n)\le \mathrm{seg}(j,n))$$

and the additional assumption of monotonicity of seg

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

It 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.

$$\forall^{\mathrm{c}}_n(\exists^{\mathrm{d}}_{i\le k\le n}\forall_{i'\le k'\le n}(\mathrm{seg}(i',k')\le \mathrm{seg}(i,k))\wedge^{\mathrm{d}}\\ \exists^{\mathrm{l}}_{j\le n}\forall_{j'\le n}(\mathrm{seg}(j',n)\le \mathrm{seg}(j,n)))$$

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. For this reason, we can use the option of specifying which lemma the decoration algorithm should use. We have

$$\mathtt{L1}\colon \forall^{\mathrm{c}}_n\exists^{\mathrm{l}}_{j\le n+1}\forall_{j'\le n+1}(\mathrm{seg}(j',n+1)\le \mathrm{seg}(j,n+1)),$$
$$\mathtt{L2}\colon \forall^{\mathrm{nc}}_n(\mathtt{IH}_n\to^{\mathrm{c}}\mathtt{Mon}\to\exists^{\mathrm{l}}_{j\le n+1}\forall_{j'\le n+1}(\mathrm{seg}(j',n+1)\le \mathrm{seg}(j,n+1))),$$

so we can try to replace L1 by L2. Since this is possible, the decoration algorithm constructs the correct proof from L2 and produces the proof resulting in the linear algorithm.

BIBLIOGRAPHY

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

[2] 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.

[3] Ulrich Berger. Uniform Heyting Arithmetic. *Annals of Pure and Applied Logic*, 133:125–148, 2005.

[4] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114:3–25, 2002.

[5] Luca Chiarabini. *Program Development by Proof Transformation.* PhD thesis, Fakultät für Mathematik, Informatik und Statistik der LMU, München, 2009.

[6] Albert Dragalin. New kinds of realizability. In *Abstracts of the 6th International Congress of Logic, Methodology and Philosophy of Sciences*, pages 20–24, Hannover, Germany, 1979.

[7] Harvey Friedman. Classically and intuitionistically provably recursive functions. In D.S. Scott and G.H. Müller, editors, *Higher Set Theory*, volume 669 of *Lecture Notes in Mathematics*, pages 21–28. Springer Verlag, Berlin, Heidelberg, New York, 1978.

[8] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts. *Dialectica*, 12:280–287, 1958.

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

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