Minimal Logic for Computable Functionals

Helmut Schwichtenberg

Mathematisches Institut der Universität München
April 1, 2022

Contents

CHAPTER 1

Minimal Logic

One of our goals is to study program extraction from proofs. We shall cover the theoretical foundations of the subject, but also try to gain some practical experience. For the latter we will make use of the system Minlog.

Minlog is intended to reason about computable functionals, using minimal logic. It is an interactive prover with the following features.

- Proofs are treated as first class objects: they can be normalized and then used for reading off an instance if the proven formula is existential, or changed for program development by proof transformation.
- To keep control over the complexity of extracted programs, we follow Kreisel's proposal and aim at a theory with a strong language and weak existence axioms. It should be conservative over (a fragment of) arithmetic.
- Minlog is based on minimal rather than classical or intuitionistic logic. This more general setting makes it possible to implement program extraction from classical proofs, via a refined A-translation (cf. [**8**]).
- Constants are intended to denote computable functionals. Since their (mathematically correct) domains are the Scott-Ershov partial continuous functionals, this is the intended range of the quantifiers.
- Variables carry (simple) types, with free algebras as base types. The latter need not be finitary (so we allow e.g. countably branching trees), and can be simultaneously generated. Type parameters (ML style) are allowed, but we keep the theory predicative and disallow type quantification. Also predicate variables are allowed, as placeholders for formulas (or more precisely, comprehension terms).
- To simplify equational reasoning, the system identifies terms with the same normal form. A rich collection of rewrite rules is provided, which can be extended by the user. Decidable predicates are implemented via boolean valued functions, hence the rewrite mechanism applies to them as well.

Acknowledgements. The Minlog system has been under development since around 1990. My sincere thanks go to the many contributors: Holger Benl (Dijkstra algorithm, inductive data types), Ulrich Berger (very many contributions), Michael Bopp (program development by proof transformation), Wilfried Buchholz (translation of classical proof into intuitionistic ones), Laura Crosilla (tutorial), Matthias Eberl (normalization by evaluation), Dan Hernest (functional interpretation), Felix Joachimski (many

contributions, in particular translation of classical proofs into intuitionistic ones, producing Tex output, documentation), Ralph Matthes (documentation), Karl-Heinz Niggl (program development by proof transformation), Jaco van de Pol (experiments concerning monotone functionals), Martin Ruckert (many contributions, in particular the MPC tool), Robert Stärk (alpha equivalence), Monika Seisenberger (many contributions, including inductive definitions and translation of classical proofs into intuitionistic ones), Klaus Weich (proof search, the Fibonacci numbers example), Wolfgang Zuber (documentation).

1.1. Natural Deduction

The system we pick for the representation of proofs is Gentzen's natural deduction, from [**19**]. Our reasons for this choice are twofold. First, as the name says this is a *natural* notion of formal proof, which means that the way proofs are represented corresponds very much to the way a careful mathematician writing out all details of an argument would go anyway. Second, formal proofs in natural deduction are closely related (via the *Curry-Howard correspondence*) to terms in typed λ-calculus. This provides us not only with a compact notation for logical derivations (which otherwise tend to become somewhat unmanagable tree-like structures), but also opens up a route to applying the computational techniques which underpin λ-calculus.

1.1.1. First Order Languages. For first order languages we use the standard language containing $\to$, $\wedge$, $\forall$, $\exists$ as primitive logical operators. We assume countably infinite supplies of individual variables, n-place predicate (or relation) symbols (constants or variables) for all $n \in \mathbb{N}$, symbols (again constants or variables) for n-ary functions for all $n \in \mathbb{N}$. 0-place predicate symbols are also called propositional symbols. 0-argument function symbols are also called (individual) constants. The language will *not*, unless stated otherwise, contain $=$ as a primitive.

Atomic formulas are formulas of the form $Rt_1 \dots t_n$, R a predicate symbol, $t_1, \dots, t_n$ individiual terms. For formulas which are either atomic or $\bot$ we use the term *prime* formula.

We use certain categories of symbols, possibly with sub- or superscripts or primed, as metavariables for certain syntactical categories (locally different conventions may be introduced):

- x, y, z, u, v, w for individual variables;
- f, g, h for function symbols;
- c, d for individual constants;
- t, s, r for terms;
- P, Q for atomic formulas;
- R for predicate symbols;
- A, B, C, D, E, F for arbitrary formulas in the language.

We introduce abbreviations:

$$\begin{aligned} \neg A &:= A \to \bot, \\ A \leftrightarrow B &:= (A \to B) \wedge (B \to A), \\ \exists^{\mathsf{cl}} x A &:= \neg\forall x \neg A \quad \text{(the classical existential quantifier)}. \end{aligned}$$

In writing formulas we save on parentheses by assuming that $\forall, \exists, \neg$ bind more strongly than $\wedge$, and that in turn $\wedge$ binds more strongly than $\to, \leftrightarrow$. Outermost parentheses are also usually dropped. Thus $A \wedge \neg B \to C$ is read as $((A \wedge (\neg B)) \to C)$. In the case of iterated implications we sometimes use the short notation

$$A_1 \to A_2 \to \dots A_{n-1} \to A_n \quad \text{for} \quad A_1 \to (A_2 \to \dots (A_{n-1} \to A_n) \dots).$$

To save parentheses in quantified formulas, we use a mild form of the *dot notation*: a dot immediately after $\forall x$ or $\exists x$ makes the scope of that quantifier as large as possible, given the parentheses around. So $\forall x.A \to B$ means $\forall x(A \to B)$, not $(\forall x A) \to B$.

We also save on parentheses by writing e.g. $Rxyz$, $Rt_0t_1t_2$ instead of $R(x, y, z)$, $R(t_0, t_1, t_2)$, where R is some predicate symbol. Similarly for a unary function symbol with a (typographically) simple argument, so fx for $f(x)$, etc. In this case no confusion will arise. But readability requires that we write in full $R(fx, gy, hz)$, instead of $Rfxgyhz$.

1.1.2. Natural Deduction. We give an inductive definition of derivation terms in Table 1.1.2, where for clarity we have written the corresponding derivations to the left.

Notice that we have left out the standard connectives $\exists$ and $\vee$, although they could easily be included, with the rules given below. The reason for this omission is that for simplicity we want our derivation terms to be pure lambda terms formed just by lambda abstraction, application, pairing and projections. This would be violated by the rules for $\exists$ and $\vee$, which require additional constructs.

In spite of this omission we can use $\exists$ and $\vee$ in our logic, if we allow appropriate axioms as constant derivation terms, e.g. for $\exists$

$$\exists^+_{x,A} \colon \forall x.A \to \exists x A$$
$$\exists^-_{x,A,B} \colon \exists x A \to (\forall x.A \to B) \to B$$

with the usual proviso $x \notin \mathsf{FV}(B)$. For $\vee$ we could introduce similar axioms; however, we do not do so here, since in the presence of e.g. the booleans we can define $\vee$ from $\exists$ via

$$A \vee B :\equiv \exists p.(p = \mathsf{tt} \to A) \wedge (p = \mathsf{ff} \to B).$$

Notice that there is one price to pay in this approach: derivations in normal form are not as normal as they could be. In particular, in the presence of the constants $\exists^+_{x,A}$ and $\exists^-_{x,A,B}$ the subformula property clearly is weaker than than it would be with the $\exists, \vee$-rules and *permutative conversion*: permute an elimination immediately following an $\exists, \vee$-rule over this rule to the minor premise.

1.2. Embedding Classical and Intuitionistic Logic

In minimal logic all propositional symbols play the same role. We now distinguish a special propositional symbol: $\bot$, to be read "falsum". We then obtain classical and intuitionistic logic by allowing appropriate additional assumptions.

derivation	term
$u\colon A$	u^A
$\dfrac{\begin{array}{cc}\mid M & \mid N\\ A & B\end{array}}{A\wedge B}\wedge^+$	$\langle M^A, N^B\rangle^{A\wedge B}$
$\dfrac{\mid M}{A\wedge B}\wedge_0^-$ $\quad$ $\dfrac{\mid M}{A\wedge B}\wedge_1^-$	$(M^{A\wedge B}0)^A$ $\quad$ $(M^{A\wedge B}1)^B$
$\dfrac{\begin{array}{c}[u\colon A]\\ \mid M\\ B\end{array}}{A\to B}\to^+ u$	$(\lambda u^A M^B)^{A\to B}$
$\dfrac{\begin{array}{cc}\mid M & \mid N\\ A\to B & A\end{array}}{B}\to^-$	$(M^{A\to B}N^A)^B$
$\dfrac{\mid M}{A}\forall^+ x$ $\quad$ (with var.cond.)	$(\lambda x M^A)^{\forall xA}$ (with var.cond.)
$\dfrac{\begin{array}{cc}\mid M & \\ \forall xA & t\end{array}}{A[x:=t]}\forall^-$	$(M^{\forall xA}t)^{A[x:=t]}$

TABLE 1.1.1. Derivation terms for $\wedge$, $\to$ and $\forall$

1.2.1. Ex-Falso-Quodlibet and Stability. To obtain *intuitionistic logic* we use as additional assumptions the *ex-falso-quodlibet* formulas Efq_R for every predicate symbol R different from $\bot$:

$$\forall\vec{x}.\bot\to R\vec{x}. \tag{Efq_R}$$

Similarly one obtains *classical logic* by allowing for every predicate symbol R different from $\bot$ the *principle of indirect proof* as an additional assumption, i.e. the formula

$$\forall\vec{x}.\neg\neg R\vec{x}\to R\vec{x}; \tag{Stab_R}$$

derivation	term
$\dfrac{t \qquad \begin{array}{c}\mid M\\ A[x{:=}t]\end{array}}{\exists x A}\,\exists^+$	$\big(\exists^+_{x,A} t M^{A[x:=t]}\big)^{\exists x A}$
$\dfrac{\begin{array}{cc}\mid M & [u\colon A]\\ \exists x A & \mid N\\ & B\end{array}}{B}\,\exists^-\,u$ (var.cond.)	$\big(M^{\exists x A}(u^A.N^B)\big)^B$ (var.cond.)
$\dfrac{\mid M}{A}\,\vee_0^+ \qquad \dfrac{\mid M}{B}\,\vee_1^+$	$\big(\vee\mathsf{I}_{0,B} M^A\big)^{A\vee B} \quad \big(\vee\mathsf{I}_{1,A} M^B\big)^{A\vee B}$
$\dfrac{\begin{array}{ccc}\mid M & [u\colon A] & [v\colon B]\\ A\vee B & \mid N & \mid K\\ & C & C\end{array}}{C}\,\vee^-\,u,v$	$\big(M^{A\vee B}(u^A.N^C, v^B.K^C)\big)^C$

TABLE 1.1.2. Derivation terms for $\exists$ and $\vee$

this formula is also called *stability* of R.

Notice that with $\bot$ for $R\vec{x}$ both formulas are trivially derivable; e.g. for stability we have $\neg\neg\bot \to \bot = ((\bot \to \bot) \to \bot) \to \bot$. The derivation is

$$\dfrac{v\colon (\bot \to \bot) \to \bot \qquad \dfrac{u\colon \bot}{\bot \to \bot}\,\to^+ u}{\bot}$$

Let

$$\mathsf{Efq} := \{\, \mathsf{Efq}_R \mid R \text{ predicate symbol} \neq \bot \,\},$$
$$\mathsf{Stab} := \{\, \mathsf{Stab}_R \mid R \text{ predicate symbol} \neq \bot \,\}.$$

We call the formula B *classically (intuitionistically) derivable* and write $\vdash_c B$ ($\vdash_i B$) if there is a derivation of B from stability assumptions Stab_A (ex-falso-quodlibet assumptions Efq_A). Similarly we define classical (intuitionistic) derivability from Γ and write $\Gamma \vdash_c B$ ($\Gamma \vdash_i B$), i.e.

$$\Gamma \vdash_i B \;:\Longleftrightarrow\; \Gamma \cup \mathsf{Efq} \vdash B,$$
$$\Gamma \vdash_c B \;:\Longleftrightarrow\; \Gamma \cup \mathsf{Stab} \vdash B.$$

LEMMA (Ex-falso-quodlibet). *$\vdash_i \bot \to A$ for every formula A.*

PROOF. By induction on A we construct for every formula A a derivation $\mathcal{D}_A$ of $\bot \to A$. *Case* A atomic formula. Use Efq_A. *Case* $A \wedge B$.

$$\dfrac{\dfrac{\mathcal{D}_A}{\bot \to A} \quad u\colon \bot}{A} \qquad \dfrac{\dfrac{\mathcal{D}_B}{\bot \to B} \quad u\colon \bot}{B}$$
$$\dfrac{A \wedge B}{\bot \to A \wedge B} \to^+ u$$

Case $A \to B$.

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_B}{\bot \to B} \quad u\colon \bot}{B}}{\dfrac{A \to B}{\bot \to A \to B}} \to^+ u$$

Case $\forall x A$.

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_A}{\bot \to A} \quad u\colon \bot}{A}}{\dfrac{\forall x A}{\bot \to \forall x A}} \to^+ u$$

Case $\exists x A$.

$$\dfrac{x \qquad \dfrac{\dfrac{\mathcal{D}_A}{\bot \to A} \quad u\colon \bot}{A}}{\dfrac{\exists x A}{\bot \to \exists x A}} \to^+ u$$

This concludes the proof. ∎

LEMMA (Stability). *For every formula* A *without* $\exists$, $\vdash_c \neg\neg A \to A$.

PROOF. Induction on A. In the constructed derivations we omit (for brevity) the introductions at the end. *Case* A atomic formula. Use Stab_A.

Case $A \wedge B$. Use $\vdash (\neg\neg A \to A) \to (\neg\neg B \to B) \to \neg\neg(A \wedge B) \to A \wedge B$, which can be derived easily from $\vdash \neg\neg(A \wedge B) \leftrightarrow (\neg\neg A \wedge \neg\neg B)$ (Exercise: derive the latter formula).

Case $A \to B$. Use $\vdash (\neg\neg B \to B) \to \neg\neg(A \to B) \to A \to B$. A derivation is

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

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

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

This concludes the proof. □

LEMMA. $\Gamma \vdash A \Rightarrow \Gamma \vdash_i A$ *and* $\Gamma \vdash_i A \Rightarrow \Gamma \vdash_c A$.

PROOF. It suffices to show $\vdash_c \mathsf{Efq}_A$. This can be seen as follows; for brevity assume R to be unary.

$$\dfrac{\dfrac{\dfrac{\forall x.\neg\neg Rx \to Rx \qquad x}{\neg\neg Rx \to Rx} \qquad \dfrac{u\colon \bot}{\neg\neg Rx}\to^+ v^{\neg Rx}}{Rx}}{\dfrac{\bot \to Rx}{\forall x.\bot \to Rx}\forall^+}\to^+ u$$

This concludes the proof. □

Note that neither of the two implications can be reversed. Counterexamples are

$$\nvdash \bot \to P, \quad \text{but} \quad \vdash_i \bot \to P,$$
$$\nvdash_i ((P \to Q) \to P) \to P, \quad \text{but} \quad \vdash_c ((P \to Q) \to P) \to P.$$

Apart from $\nvdash_i ((P \to Q) \to P) \to P$ the proofs are easy. However, to prove the non-derivability of the Peirce formula $((P \to Q) \to P) \to P$ in intuitionistic logic requires a more careful study of intuitionistic derivability.

LEMMA (Proof by cases). $\vdash_c (A \to B) \wedge (\neg A \to B) \to B$.

PROOF. Let C abbreviate $(A \to B) \wedge (\neg A \to B)$

$$\dfrac{\dfrac{\mathcal{D}_{\mathsf{Stab}}}{\neg\neg B \to B} \qquad \dfrac{\dfrac{w\colon \neg B \qquad \dfrac{\dfrac{u\colon C}{\neg A \to B} \qquad \dfrac{\dfrac{u\colon C}{A \to B} \qquad v\colon A}{B}}{B}}{\bot}\to^+ v}{\neg\neg B}\to^+ w}{B}$$

where $\mathcal{D}_{\mathsf{Stab}}$ is a derivation provided by the Stability Lemma.. □

1.2.2. Equivalence. Call two formulas A and B *equivalent* in minimal (classical, intuitionistic) logic if $\vdash A \leftrightarrow B$ ($\vdash_c A \leftrightarrow B$, $\vdash_i A \leftrightarrow B$).

LEMMA (Equivalence). *Let* $\vdash_{mic} \in \{\vdash, \vdash_i, \vdash_c\}$. *Then* $\vdash_{mic} A_1 \leftrightarrow A_2$ *and* B_2 *is obtained from* B_1 *by replacing one subformula* A_1 *of* B_1 *by* A_2, *then we also have* $\vdash_{mic} B_1 \leftrightarrow B_2$.

PROOF. Induction on B_1. If all of B_1 had been replaced, the claim is obvious. Otherwise B_1 must be a composed formula.

Case $C_1 \wedge D_1$. Assume the replacement takes places in C_1. We have to show $\vdash_{mic} C_1 \wedge D_1 \leftrightarrow C_2 \wedge D_1$. $\to$:

$$\dfrac{\dfrac{\mathcal{D} \qquad \dfrac{C_1 \wedge D_1}{C_1}}{C_2} \qquad \dfrac{C_1 \wedge D_1}{D_1}}{C_2 \wedge D_1}$$

where $\mathcal{D}$ is known by induction hypothesis. $\leftarrow$ is proved similarly.

Case $C_1 \to D_1$. Assume the replacement takes places in C_1. We have to show $\vdash_{mic} (C_1 \to D_1) \leftrightarrow (C_2 \to D_1)$. $\to$:

$$\dfrac{\dfrac{C_1 \to D_1 \qquad \dfrac{\begin{matrix}\mathcal{D}\\ C_2 \to C_1\end{matrix} \qquad u\colon C_2}{C_1}}{D_1}}{C_2 \to D_1}\to^+ u$$

where again $\mathcal{D}$ is known by induction hypothesis. $\leftarrow$ is proved similarly. Assume now that the replacement takes places in D_1. We must show $\vdash_{mic}$ $(C_1 \to D_1) \leftrightarrow (C_1 \to D_2)$. $\to$:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ D_1 \to D_2\end{matrix} \qquad \dfrac{C_1 \to D_1 \qquad u\colon C_1}{D_1}}{D_2}}{C_1 \to D_2}\to^+ u$$

where again $\mathcal{D}$ is known by induction hypothesis. $\leftarrow$ is proved similarly.

Case $\forall x C_1$. We must show $\vdash_{mic} \forall x C_1 \leftrightarrow \forall x C_2$. $\to$:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ C_1 \to C_2\end{matrix} \qquad \dfrac{\forall x C_1 \qquad x}{C_1}}{C_2}}{\forall x C_2}$$

where again $\mathcal{D}$ is known by induction hypothesis. Observe that $\mathcal{D}$ does not contain free assumptions. $\leftarrow$ is proved similarly.

Case $\exists x C_1$. Similar. □

1.3. Glivenko's Theorem

As an illustration of what can be done in intuitionistic propositional logic we prove Glivenko's theorem, which says that every negation, which is a classical tautology, can be derived intuitionistically.

1.3.1. Valuations.

LEMMA. $\vdash (A \to B) \to (\neg A \to B) \to \neg\neg B$.

PROOF. Assume $\neg B$. Then $\neg A$, hence B, hence $\bot$. □

For $\sigma \in \{0, 1\}$ let

$$A^\sigma := \begin{cases} A & \text{if } \sigma = 1 \\ \neg A & \text{if } \sigma = 0 \end{cases}$$

A *valuation* v is a mapping from propositional variables into *truth values*, which are taken here to be 0 and 1, for falsity and truth. Call a formula a *tautology* if for every valuation v its truth value $v(A)$ is 1. For $* \in \{\to, \wedge\}$ let $\sigma * \tau$ be the truth value given by the well-known truth table for the connective $*$.

1.3.2. Intuitionistic Derivability and Valuations.

LEMMA. $\vdash_i A^\sigma \to B^\tau \to (A * B)^{\sigma * \tau}$ *for* $* \in \{\to, \wedge\}$.

PROOF. *Case* $\to$.

$$\vdash B \to A \to B$$
$$\vdash A \to \neg B \to \neg(A \to B)$$
$$\vdash_i \neg A \to A \to B$$

Case $\wedge$.

$$\vdash A \to B \to A \wedge B$$
$$\vdash \neg A \to \neg(A \wedge B)$$
$$\vdash \neg B \to \neg(A \wedge B)$$

Notice that these formulas are slightly stronger than required. □

LEMMA. *Let A be a formula in propositional logic, and let $P_1, \dots, P_n$ be all propositional variables in A. The we have, for every valuation v,*

$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_n^{v(P_n)} \to A^{v(A)}.$$

PROOF. By induction on A. *Case* P_i. Clear. *Case* $A * B$. Use

$$\vdash_i A^{v(A)} \to B^{v(B)} \to (A * B)^{v(A) * v(B)},$$

which holds by the previous lemma. □

1.3.3. Glivenko's Theorem.

THEOREM (Glivenko). $\vdash_i \neg A$ *for* $\neg A$ *a tautology.*

PROOF. By the final lemma in 1.3.2 we have, for every valuation v,

$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_n^{v(P_n)} \to \neg A,$$

since by assumption $v(\neg A) = 1$. We show

$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_k^{v(P_k)} \to \neg A \quad \text{for every valuation } v,$$

by induction on $i := n - k$. *Base* $i = 0$. Clear, by what we just noted. *Step.* We must show

$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_{k-1}^{v(P_{k-1})} \to \neg A.$$

By IH we have

$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_{k-1}^{v(P_{k-1})} \to P_k \to \neg A,$$
$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_{k-1}^{v(P_{k-1})} \to \neg P_k \to \neg A.$$

Using the lemma in 1.3.1 we obtain

$$\vdash_i P_1^{v(P_1)} \to \cdots \to P_{k-1}^{v(P_{k-1})} \to \neg\neg\neg A.$$

The claim follows from $\vdash \neg\neg\neg A \to \neg A$. – Applying the inductively proven claim with $i = n$ (i.e., $k = 0$) yields $\vdash_i \neg A$, as required. □

1.4. Negative Translation

Having defined classical and intuitionistic logic out of minimal logic by adding axioms, we next show that in fact, both logics can be embedded into minimal logic, as long as we restrict ourselves to the language based on $\{\to, \wedge, \forall, \bot\}$. This restriction will be in force for the rest of the present chapter.

1.4.1. Negative Formulas. A formula A of the $\{\to, \wedge, \forall, \bot\}$-language is called *negative*, if every atomic formula of A different from $\bot$ occurs negated.

LEMMA. *For negative A we have $\vdash \neg\neg A \to A$.*

PROOF. Use the Stability Lemma and $\vdash \neg\neg\neg R\vec{t} \to \neg R\vec{t}$. □

We now define the Gödel-Gentzen (or "negative") translation of classical logic into minimal logic. The basic idea is to double negate every atomic formula.

DEFINITION (Negative translation ^{g} of Gödel-Gentzen).

$$\begin{aligned}
\bot^g &:= \bot,\\
R\vec{t}^g &:= \neg\neg R\vec{t},\\
(A \wedge B)^g &:= A^g \wedge B^g,\\
(A \to B)^g &:= A^g \to B^g,\\
(\forall x A)^g &:= \forall x A^g.
\end{aligned}$$

Notice that A^g is the same as A for negative A.

THEOREM. *For all formulas A we have*

(a) $\vdash_c A \leftrightarrow A^g$,
(b) $\Gamma \vdash_c A$ *iff* $\Gamma^g \vdash A^g$, *where* $\Gamma^g := \{\, B^g \mid B \in \Gamma \,\}$.

PROOF. (a). The claim follows easily from the Equivalence Lemma..

(b) $\Leftarrow$. Obvious. $\Rightarrow$. By induction on the classical derivation. For a stability assumption $\neg\neg R\vec{t} \to R\vec{t}$ we have $(\neg\neg R\vec{t} \to R\vec{t})^g = \neg\neg R\vec{t} \to \neg\neg R\vec{t}$, and this is easily derivable.

Case $\to^+$. Assume

$$\dfrac{\begin{array}{c}[u\colon A]\\ \mathcal{D}\\ B\end{array}}{A \to B}\,{\to^+}u$$

We have by induction hypothesis

$$\dfrac{\begin{array}{c}[u\colon A^g]\\ \mathcal{D}^g\\ B^g\end{array}}{A^g \to B^g}\,{\to^+}u$$

Case $\to^-$. Assume

$$\dfrac{\begin{array}{cc}\mathcal{D}_0 & \mathcal{D}_1\\ A \to B & A\end{array}}{B}$$

We have by induction hypothesis

$$\frac{\begin{array}{cc}\mathcal{D}_0^g & \mathcal{D}_1^g \\ A^g \to B^g & A^g\end{array}}{B^g}$$

The other cases are treated similarly. □

COROLLARY (Embedding of classical logic). *For negative A,*

$$\vdash_c A \iff \vdash A.$$

PROOF. By the theorem we have $\vdash_c A$ iff $\vdash A^g$. Since A is negative, every atom distinct from $\bot$ in A must occur negated, as $\neg R\vec{t}$, and hence in A^g it appears in same form. □

Since every formula is classically equivalent to a negative formula, we have achieved an embedding of classical logic into minimal logic.

Note that $\not\vdash \neg\neg P \to P$. The corollary therefore does not hold for all formulas A.

1.4.2. Formulas Implying Their Negative Translation. We introduce a further observation (due to Leivant; see Troelstra and van Dalen [**35**, Ch.2, Sec.3]) which will be useful for program extraction from classical proofs (cf. Chapter 6). There it will be necessary to actually transform a given classical derivation $\vdash_c A$ into a minimal logic derivation $\vdash A^g$. In particular, for every assumption constant C used in the given derivation we have to provide a derivation of C^g. Now for some formulas S – the so-called *spreading* formulas – this is immediate, for we can derive $S \to S^g$, and hence can use the original assumption constant.

Recall that our formulas may contain *predicate variables* denoted by X, which are place holders for comprehension terms, i.e. formulas with distinguished variables. We use the obvious notation $A[X:=\{\,\vec{x} \mid B\,\}]$ or shortly $A[\{\,\vec{x} \mid B\,\}]$ or even $A[B]$ for substitution for predicate variables. Clearly the Gödel-Gentzen translation of $X\vec{t}$ is $\neg\neg X\vec{t}$.

Recall also that an assumption constant consists of an uninstantiated formula (e.g. $X0 \to (\forall n.Xn \to X(n+1)) \to \forall n Xn$ for induction) together with a substitution of comprehension terms for predicate variables (e.g. $X \mapsto \{\,n \mid n < n+1\,\}$). Then in order to immediately obtain a derivation of C^g for C an assumption constant it suffices to know that its *uninstantiated* formula S is spreading, for then we generally have $\vdash S[\vec{A^g}] \to S[\vec{A}]^g$ (see the theorem below) and hence can use the same assumption constant with a different substitution.

We define *spreading* formulas S, *wiping* formulas W and *isolating* formulas I inductively.

$$\begin{array}{rcl} S & := & \bot \mid R\vec{t} \mid X\vec{t} \mid S \wedge S \mid I \to S \mid \forall x S, \\ W & := & \bot \mid X\vec{t} \mid W \wedge W \mid S \to W \mid \forall x W, \\ I & := & R\vec{t} \mid W \mid I \wedge I. \end{array}$$

Let $\mathcal{S}$ $(\mathcal{W}, \mathcal{I})$ be the class of spreading (wiping, isolating) formulas.

THEOREM.

$$\vdash S[\vec{A}^g] \to S[\vec{A}]^g \qquad \textit{for every spreading formula } S,$$
$$\vdash W[\vec{A}]^g \to W[\vec{A}^g] \qquad \textit{for every wiping formula } W,$$
$$\vdash I[\vec{A}]^g \to \neg\neg I[\vec{A}^g] \qquad \textit{for every isolating formula } I.$$

We assume here that all occurrences of predicate variables are substituted.

PROOF. For brevity we write S^g for $S[\vec{A}]^g$ and S for $S[\vec{A}^g]$, and similarly for W and I.

Case $\bot \in \mathcal{S}$. We must show $\vdash \bot \to \bot^g$. Take $\lambda u^{\bot} u$.

Case $R\vec{t} \in \mathcal{S}$. We must show $\vdash R\vec{t} \to \neg\neg R\vec{t}$. Take $\lambda u^{R\vec{t}} \lambda v^{\neg R\vec{t}}.vu$.

Case $X\vec{t} \in \mathcal{S}$, with X substituted by $\{\, \vec{x} \mid A \,\}$. We must show $\vdash A^g[\vec{t}] \to A^g[\vec{t}]$, which is trivial.

Case $S_1 \wedge S_2 \in \mathcal{S}$. We must show $\vdash S_1 \wedge S_2 \to S_1^g \wedge S_2^g$. Take

$$\dfrac{\dfrac{\dfrac{\text{IH}}{S_1 \to S_1^g} \quad \dfrac{u\colon S_1 \wedge S_2}{S_1}}{S_1^g} \qquad \dfrac{\dfrac{\text{IH}}{S_2 \to S_2^g} \quad \dfrac{u\colon S_1 \wedge S_2}{S_2}}{S_2^g}}{S_1^g \wedge S_2^g}$$

Case $I \to S \in \mathcal{S}$. We must show $\vdash (I \to S) \to I^g \to S^g$. Recall that $\vdash \neg\neg S^g \to S^g$ by the Stability Lemma, because S^g is negative. Take

$$\dfrac{\dfrac{\dfrac{\text{IH}}{I^g \to \neg\neg I} \quad v\colon I^g}{\neg\neg I} \qquad \dfrac{w_1\colon \neg S^g \qquad \dfrac{\dfrac{\text{IH}}{S \to S^g} \quad \dfrac{u\colon I \to S \quad w_2\colon I}{S}}{S^g}}{\dfrac{\bot}{\neg I} \to^+ w_2}}{\dfrac{\bot}{\neg\neg S^g} \to^+ w_1} \quad \text{Stab}}{S^g}$$

Case $\forall x S \in \mathcal{S}$. We must show $\vdash \forall x S \to \forall x S^g$. Take

$$\dfrac{\dfrac{\text{IH}}{S \to S^g} \quad \dfrac{u\colon \forall x S \qquad x}{S}}{S^g}$$

Case $\bot \in \mathcal{W}$. We must show $\vdash \bot^g \to \bot$. Take $\lambda u^{\bot} u$.

Case $X\vec{t} \in \mathcal{W}$, with X substituted by $\{\, \vec{x} \mid A \,\}$. We must show $\vdash A^g[\vec{t}] \to A^g[\vec{t}]$, which is trivial.

Case $W_1 \wedge W_2 \in \mathcal{W}$. We must show $\vdash W_1^g \wedge W_2^g \to W_1 \wedge W_2$. Take

$$\dfrac{\dfrac{\dfrac{\text{IH}}{W_1^g \to W_1} \quad \dfrac{u\colon W_1^g \wedge W_2^g}{W_1^g}}{W_1} \qquad \dfrac{\dfrac{\text{IH}}{W_2^g \to W_2} \quad \dfrac{u\colon W_1^g \wedge W_2^g}{W_2^g}}{W_2}}{W_1 \wedge W_2}$$

Case $S \to W \in \mathcal{W}$. We must show $\vdash (S^g \to W^g) \to S \to W$. Take

$$\dfrac{\dfrac{\text{IH}}{W^g \to W} \quad \dfrac{u\colon S^g \to W^g \qquad \dfrac{\dfrac{\text{IH}}{S \to S^g} \quad v\colon S}{S^g}}{W^g}}{W}$$

Case $\forall x W \in \mathcal{W}$. We must show $\vdash \forall x W^g \to \forall x W$. Take

$$\dfrac{\dfrac{\text{IH}}{W^g \to W} \qquad \dfrac{u\colon \forall x W^g \qquad x}{W^g}}{W}$$

Case $R\vec{t} \in \mathcal{I}$. We must show $\vdash \neg\neg R\vec{t} \to \neg\neg R\vec{t}$, which is trivial.

Case $W \in \mathcal{I}$. We must show $\vdash W^g \to \neg\neg W$, which trivially follows from the IH $\vdash W^g \to W$. Take

$$\dfrac{v\colon \neg W \qquad \dfrac{\dfrac{\text{IH}}{W^g \to W} \qquad u\colon W^g}{W}}{\bot}$$

Case $I_1 \wedge I_2 \in \mathcal{I}$. We must show $\vdash I_1^g \wedge I_2^g \to \neg\neg(I_1 \wedge I_2)$. Take

$$\dfrac{\dfrac{\dfrac{\text{IH}}{I_2^g \to \neg\neg I_2} \qquad \dfrac{I_1^g \wedge I_2^g}{I_2^g}}{\neg\neg I_2} \qquad \dfrac{\dfrac{\dfrac{\text{IH}}{I_1^g \to \neg\neg I_1} \qquad \dfrac{I_1^g \wedge I_2^g}{I_1^g}}{\neg\neg I_1} \qquad \dfrac{\dfrac{\neg(I_1 \wedge I_2) \qquad \dfrac{I_1 \quad I_2}{I_1 \wedge I_2}}{\bot}}{\neg I_1}}{\bot}}{\neg I_2}}{\bot}$$

This completes the proof. □

Notice that the Gödel-Gentzen translation double negates every atom, and hence may produce triple negations. However, because of $\vdash \neg\neg\neg A \leftrightarrow \neg A$ and the Equivalence Lemma we can systematically replace triple negations by single negations. As a guide for the implementation, we carry out some of the details here.

Let A^* (the *reduced form* of A) be obtained from A by replacing $\neg\neg\neg A$ by $\neg A$ whenever possible. So

$$\begin{aligned}
\bot^* &:= \bot,\\
R\vec{t}^* &:= R\vec{t},\\
(A \wedge B)^* &:= A^* \wedge B^*,\\
(A \to B)^* &:= \begin{cases} (\neg C)^* & \text{if } A \to B = \neg\neg\neg C\\ A^* \to B^*, & \text{otherwise}\end{cases}\\
(\forall x A)^* &:= \forall x A^*.
\end{aligned}$$

We simultaneously construct derivations of $A \to A^*$ and $A^* \to A$.

Lemma. (a) $\vdash A \to A^*$,
(b) $\vdash A^* \to A$.

Proof. *Case* A prime formula. Then A^* is A, and we can take $\lambda u^A u$.

Case $\forall x A$. (a). We must show $\vdash \forall x A \to \forall x A^*$. Take

$$\dfrac{\dfrac{\text{IH(a)}}{A \to A^*} \qquad \dfrac{u\colon \forall x A \qquad x}{A}}{A^*}$$

(b). We must show $\vdash \forall x A^* \to \forall x A$. Take

$$\dfrac{\dfrac{\text{IH(b)}}{A^* \to A} \qquad \dfrac{u\colon \forall x A^*}{A^*}\,x}{A}$$

Case $\neg\neg\neg A$. (a). We must show $\vdash \neg\neg\neg A \to (\neg A)^*$.

$$\dfrac{\dfrac{\text{IH(a)}}{\neg A \to (\neg A)^*} \qquad \dfrac{u\colon \neg\neg\neg A \qquad \dfrac{\dfrac{v\colon \neg A \qquad w\colon A}{\bot}}{\neg\neg A}\to^+ v}{\neg A}\to^+ w}{(\neg A)^*}$$

(b). We must show $\vdash (\neg A)^* \to \neg\neg\neg A$.

$$\dfrac{v\colon \neg\neg A \qquad \dfrac{\dfrac{\text{IH(b)}}{(\neg A)^* \to \neg A} \qquad u\colon (\neg A)^*}{\neg A}}{\bot}$$

Case $A \to B$ not of the form $\neg\neg\neg C$. (a). We must show $\vdash (A \to B) \to A^* \to B^*$.

$$\dfrac{\dfrac{\text{IH(a)}}{B \to B^*} \qquad \dfrac{u\colon A \to B \qquad \dfrac{\dfrac{\text{IH(b)}}{A^* \to A} \qquad v\colon A^*}{A}}{B}}{B^*}$$

(b). We must show $\vdash (A^* \to B^*) \to A \to B$.

$$\dfrac{\dfrac{\text{IH(b)}}{B^* \to B} \qquad \dfrac{u\colon A^* \to B^* \qquad \dfrac{\dfrac{\text{IH(a)}}{A \to A^*} \qquad v\colon A}{A^*}}{B^*}}{B}$$

□

1.5. Notes

The proof of Glivenko's theorem is taken form Mints' book [**28**].

CHAPTER 2

Algebras

A free algebra is given by *constructors*, for instance zero and successor for the natural numbers. We want to treat other data types as well, like lists and binary trees. When dealing with inductively defined sets, it will also be useful to explicitely refer to the generation tree. Such trees are quite often infinitely branching, and hence we allow infinitary free algebras from the outset.

The freeness of the constructors is expressed by requiring that their ranges are disjoint and that they are injective. Moreover, we view the free algebra as a domain and require that its bottom element is not in the range of the constructors. Hence the constructors are total and non-strict. For the notion of totality cf. [**34**, Chapter 8.3].

To make a free algebra into a domain and still have the constructors injective and with disjoint ranges, we model e.g. the natural numbers as shown in Figure 1. Notice that for more complex algebras we usually need many more "infinite" elements; this is a consequence of the closure of domains under suprema. To make dealing with such complex structures less annoying, we will normally restrict attention to the *total* elements of a domain, in this case – as expected – the elements labelled 0, $S0$, $S(S0)$ etc.

2.1. Examples of Finitary and Infinitary Algebras

We shall consider some examples of free algebras, generated from constructors.

(1) The easiest example is the algebra **U**, which has just one nullary constructor called Dummy. It consists of exactly one element.
(2) Almost as easy is the next example, the algebra **B**, which has just two nullary constructors called tt and ff (sometimes we use `True` and `False` instead). It consists of exactly two elements.

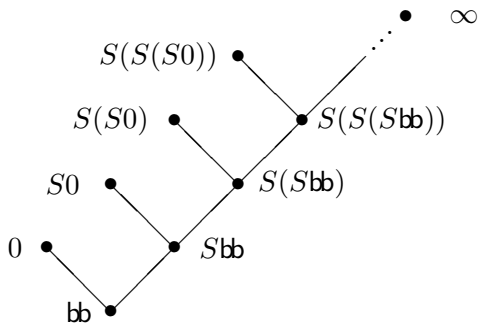

FIGURE 1. The domain of natural numbers

(3) The next example is the simplest algebra with infinitely many elements, the algebra $\mathbf{N}$ of natural numbers. Its constructors are a nullary constructor 0 (also written `Zero`) and a unary constructor S (also written `Succ`).
(4) We shall also allow parametrized algebras, depending on type parameters. The simplest example is the algebra $\mathbf{L}(\rho)$ (sometimes we use (`list` ρ) instead) of lists of objects of a given type ρ. Its constructors are a nullary constructor Nil and a binary constructor cons; the latter takes an object x of type ρ and a list l and constructs out of these the new list obtained by putting x to the front of l.
(5) Another important parametrized algebra is the disjoint union of the algebras given by types ρ_1 and ρ_2. It is called (ρ_1 `yplus` ρ_2) ("y" from t*y*pe), and is given by two unary constructors Inl and Inr of types $\rho_1 \to \rho_1 + \rho_2$ and $\rho_2 \to \rho_1 + \rho_2$, respectively.
(6) We also admit simultaneously generated free algebras. An example is provided by the obvious way to simultaneously generate trees and finite lists (of arbitrary length) of trees. The algebra tree has one nullary constructor Leaf (taking a natural number as parameter) and one unary constructor Branch, building a tree from a tree list. The algebra tlist has one nullary constructor Empty and one binary constructor Tcons; the latter takes an object t of type tree and a tree list l and constructs out of these the new tree list obtained by putting t to the front of l.

All these algebras are *finitary*, in the sense that every constructor takes only finitely many arguments. Then for any two elements of the algebra we can decide whether they are equal.

When dealing with inductively defined sets, it will also be useful to explicitely refer to the generation tree. Such trees are quite often countably branching, and hence we also allow *infinitary* free algebras from the outset, whose constructors may take infinitely many arguments. Notice that then equality is *not* decidable any more, and hence needs to be axiomatized.

For an example of an infinitary algebra, consider the countable ordinals. They can be seen as generated from a nullary constructor 0, a unary constructor for the successor and a constructor building the supremum out of a countably infinite list of ordinals (given by a function from the natural numbers to ordinals).

Clearly the generation process of the elements of a free algebra allows *recursive* definitions of functions on such algebras. For the examples above, they have (when no parameters are present) the following form.

(1) For the algebra $\mathbf{U}$ we may just define

$$f(\mathsf{Dummy}) = a$$

(2) Similarly for the algebra $\mathbf{B}$ we can explicitely define

$$f(\mathsf{tt}) = a,$$
$$f(\mathsf{ff}) = b.$$

(3) For the algebra $\mathbf{N}$ of natural numbers we obtain the familiar recursion scheme

$$f(0) = a,$$
$$f(\mathsf{S}(n)) = h(n, f(n)).$$

(4) For the parametrized algebra $\mathbf{L}(\rho)$ we have a very similar recursion scheme:

$$f(\mathsf{Nil}) = a,$$
$$f(\mathsf{cons}(x, l)) = h(x, l, f(l)).$$

(5) For the disjoint union of given types ρ_1 and ρ_2, the algebra does not require recursive calls, and hence the recursion scheme is simply

$$f(\mathsf{Inl}(x)) = g(x),$$
$$f(\mathsf{Inr}(y)) = h(y).$$

(6) Somewhat more interesting are simultaneously generated free algebras. Here we may define

$$f(\mathsf{Leaf}(n)) = h_1(n),$$
$$f(\mathsf{Branch}(l)) = h_2(l, g(l)),$$
$$g(\mathsf{Empty}) = b,$$
$$g(\mathsf{Tcons}(x, l)) = h_3(x, l, f(x), g(l)).$$

2.2. Recursion, Strong Normalization

We give a predicative proof of strong normalization for terms with recursion operators in a system of simultaneously defined free algebras. The proof uses a variant of the Tait's strong computability predicates.

It is well known that in a system of simultaneously defined free algebras every term (possibly involving recursion operators) is strongly normalizing. However, the standard proof reduces the problem to strong normalization of second order propositional logic (called system F by Girard [**20**]). This latter result requires a method not formalizable in analysis. Here we give a much simpler proof, which only uses predicative methods.

2.2.1. Types. Our type system is defined by three type forming operations: arrow types $\rho \to \sigma$, pair types $\rho \times \sigma$ and the formation of inductively generated types $\mu\vec{\alpha}\,\vec{\kappa}$, where $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ is a list of distinct "type variables", and $\vec{\kappa} = (\kappa_i)_{i=1,\dots,k}$ is a list of "constructor types", whose argument types contain $\alpha_1, \dots, \alpha_N$ in strictly positive positions only.

For instance, $\mu\alpha(\alpha, \alpha \to \alpha)$ is the type of natural numbers; here the list $(\alpha, \alpha \to \alpha)$ stands for two generation principles: α for "there is a natural number" (the 0), and $\alpha \to \alpha$ for "for every natural number there is another one" (its successor).

Let an infinite supply of *type variables* α, β be given.

Definition. Let $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ be a list of distinct type variables. *Types* $\rho, \sigma, \tau, \mu \in \mathsf{T}$ and *constructor types* $\kappa \in \mathsf{KT}(\vec{\alpha})$ are defined inductively

as follows.

$$\frac{\vec{\rho}, \vec{\sigma}_1, \ldots, \vec{\sigma}_n \in \mathsf{T}}{\vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})} \quad (n \geq 0)$$

$$\frac{\kappa_1, \ldots, \kappa_n \in \mathsf{KT}(\vec{\alpha})}{(\mu\vec{\alpha}\,(\kappa_1, \ldots, \kappa_n))_j \in \mathsf{T}} \quad (n \geq 1) \qquad \frac{\rho, \sigma \in \mathsf{T}}{\rho \to \sigma \in \mathsf{T}} \qquad \frac{\rho, \sigma \in \mathsf{T}}{\rho \times \sigma \in \mathsf{T}}$$

Here $\vec{\rho}$ is short for a list $\rho_1, \ldots, \rho_m$ $(m \geq 0)$ of types and $\vec{\rho} \to \sigma$ means $\rho_1 \to \cdots \to \rho_m \to \sigma$, associated to the right. We shall use μ for types of the form $(\mu\vec{\alpha}\,(\kappa_1, \ldots, \kappa_k))_j$ only, and for types $\vec{\tau}$ and a constructor type $\kappa = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$ let

$$\kappa[\vec{\tau}] := \vec{\rho} \to (\vec{\sigma}_1 \to \tau_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \tau_{j_n}) \to \tau_j.$$

Examples.

$$\begin{array}{ll}
\mathbf{U} & := \mu\alpha\,\alpha, \\
\mathbf{B} & := \mu\alpha\,(\alpha, \alpha), \\
\mathbf{N} & := \mu\alpha\,(\alpha, \alpha \to \alpha), \\
\mathbf{L}(\rho) & := \mu\alpha\,(\alpha, \rho \to \alpha \to \alpha), \\
\rho_1 + \rho_2 & := \mu\alpha\,(\rho_1 \to \alpha, \rho_2 \to \alpha), \\
(\mathsf{tree}, \mathsf{tlist}) & := \mu(\alpha, \beta)\,(\mathbf{N} \to \alpha, \beta \to \alpha, \beta, \alpha \to \beta \to \beta), \\
\mathbf{Bin} & := \mu\alpha\,(\alpha, \alpha \to \alpha \to \alpha), \\
\mathcal{O} & := \mu\alpha\,(\alpha, \alpha \to \alpha, (\mathbf{N} \to \alpha) \to \alpha), \\
\mathcal{T}_0 & := \mathbf{N}, \\
\mathcal{T}_{n+1} & := \mu\alpha\,(\alpha, (\mathcal{T}_n \to \alpha) \to \alpha).
\end{array}$$

Notice that there are many equivalent ways to define these types. For instance, we could take $\mathbf{U} + \mathbf{U}$ to be the type of booleans, and $\mathbf{L}(\mathbf{U})$ to be the type of natural numbers.

Notice also that we have added the pair type $\rho \times \sigma$ for simplicity only. Products could have been defined in two forms, as tensor products and as cartesian products, by

$$\begin{array}{ll}
\rho_1 \otimes \rho_2 & := \mu\alpha.\rho_1 \to \rho_2 \to \alpha, \\
\rho_1 \times_\sigma \rho_2 & := \mu\alpha.(\sigma \to \rho_1) \to (\sigma \to \rho_2) \to \sigma \to \alpha,
\end{array}$$

If we would allow ourselves to quantify over types, the cartesian product could be defined as

$$\rho_1 \times \rho_2 := \mu\alpha\forall\sigma.(\sigma \to \rho_1) \to (\sigma \to \rho_2) \to \sigma \to \alpha.$$

2.2.2. Terms. The inductive structure of the types $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ corresponds to two sorts of constants. With the *constructors* $\mathsf{constr}_i^{\vec{\mu}} \colon \kappa_i[\vec{\mu}]$ we can construct elements of a type μ_j, and with the *recursion operators* $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ we can construct mappings from μ_j to τ_j by recursion on the structure of $\vec{\mu}$. In order to define the type of the recursion operators w.r.t. $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ and result types $\vec{\tau}$, we first define for

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$$

the *step type*

$$\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \to (\vec{\sigma}_1 \to \mu_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \mu_{j_n}) \to \\ (\vec{\sigma}_1 \to \tau_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \tau_{j_n}) \to \tau_j.$$

Here $\vec{\rho}, (\vec{\sigma}_1 \to \mu_{j_1}), \ldots, (\vec{\sigma}_n \to \mu_{j_n})$ correspond to the *components* of the object of type μ_j under consideration, and $(\vec{\sigma}_1 \to \tau_{j_1}), \ldots, (\vec{\sigma}_n \to \tau_{j_n})$ to the previously defined values. The recursion operator $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ has type

$$\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}} \colon \delta_1^{\vec{\mu},\vec{\tau}} \to \cdots \to \delta_k^{\vec{\mu},\vec{\tau}} \to \mu_j \to \tau_j$$

(recall that k is the total number of constructors for all types $\mu_1, \ldots, \mu_N$).

We will often write $\mathcal{R}_j^{\vec{\mu},\vec{\tau}}$ for $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$, and omit the upper indices $\vec{\mu}, \vec{\tau}$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e. of type $\mu\alpha\,\kappa$, for $\mathcal{R}_\mu^{\mu,\tau}$ we normally write $\mathcal{R}_\mu^\tau$.

Definition. *Terms* are inductively defined from typed variables and the constants $\mathsf{constr}_i^{\vec{\mu}}$ and $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ by means of

- *abstraction* $(\lambda x^\rho M^\sigma)^{\rho\to\sigma}$,
- *application* $(M^{\rho\to\sigma} N^\rho)^\sigma$,
- *pairing* $\langle M^\rho, N^\sigma\rangle^{\rho\times\sigma}$ and
- *projections* $(M^{\rho\times\sigma} 0)^\rho$, $(M^{\rho\times\sigma} 1)^\sigma$.

Examples.

$$\mathsf{tt}^{\mathbf{B}} := \mathsf{constr}_1^{\mathbf{B}}, \quad \mathsf{ff}^{\mathbf{B}} := \mathsf{constr}_2^{\mathbf{B}},$$
$$\mathcal{R}_{\mathbf{B}}^\tau \colon \tau \to \tau \to \mathbf{B} \to \tau,$$

$$0^{\mathbf{N}} := \mathsf{constr}_1^{\mathbf{N}}, \quad \mathsf{S}^{\mathbf{N}\to\mathbf{N}} := \mathsf{constr}_2^{\mathbf{N}},$$
$$\mathcal{R}_{\mathbf{N}}^\tau \colon \tau \to (\mathbf{N} \to \tau \to \tau) \to \mathbf{N} \to \tau,$$

$$\mathsf{Nil}^{\mathbf{L}(\alpha)} := \mathsf{constr}_1^{\mathbf{L}(\alpha)}, \quad \mathsf{cons}^{\alpha\to\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)} := \mathsf{constr}_2^{\mathbf{L}(\alpha)},$$
$$\mathcal{R}_{\mathbf{L}(\alpha)}^\tau \colon \tau \to (\alpha \to \mathbf{L}(\alpha) \to \tau \to \tau) \to \mathbf{L}(\alpha) \to \tau,$$

$$\big(\mathsf{Inl}_{\rho_1\rho_2}\big)^{\rho_1\to\rho_1+\rho_2} := \mathsf{constr}_1^{\rho_1+\rho_2},$$
$$\big(\mathsf{Inr}_{\rho_1\rho_2}\big)^{\rho_2\to\rho_1+\rho_2} := \mathsf{constr}_2^{\rho_1+\rho_2},$$
$$\mathcal{R}_{\rho_1+\rho_2}^\tau \colon (\rho_1 \to \tau) \to (\rho_2 \to \tau) \to \rho_1 + \rho_2 \to \tau.$$

Remark. Notice that for the defined products the constructors and recursion operators are

$$\big(\otimes^+_{\rho_1\rho_2}\big)^{\rho_1\to\rho_2\to\rho_1\otimes\rho_2} := \mathsf{constr}_1^{\rho_1\otimes\rho_2},$$
$$\mathcal{R}_{\rho_1\otimes\rho_2}^\tau \colon (\rho_1 \to \rho_2 \to \tau) \to \rho_1\otimes\rho_2 \to \tau,$$

$$\big(\times^+_{\rho_1\rho_2\sigma}\big)^{(\sigma\to\rho_1)\to(\sigma\to\rho_2)\to\sigma\to\rho_1\times_\sigma\rho_2} := \mathsf{constr}_1^{\rho_1\times_\sigma\rho_2},$$
$$\mathcal{R}_{\rho_1\times_\sigma\rho_2}^\tau \colon ((\sigma \to \rho_1) \to (\sigma \to \rho_2) \to \sigma \to \tau) \to \rho_1 \times_\sigma \rho_2 \to \tau,$$

Examples. The *append*-function $:\!+\!:$ for lists is defined recursively by

$$\mathsf{Nil} :\!+\!: l_2 := l_2,$$
$$(\mathsf{cons}\, x\, l_1) :\!+\!: l_2 := \mathsf{cons}\, x (l_1 :\!+\!: l_2).$$

It can be defined as the term

$$M_{:+:} := \mathcal{R}^{\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)}(\lambda l_2 l_2)(\lambda x\lambda l_1\lambda p\lambda l_2.\mathsf{cons}\, x(pl_2)).$$

Using the append function $:+:$ we can define *list reversal* Rev by

$$\begin{aligned}\mathsf{Rev}\ \mathsf{Nil} &:= \mathsf{Nil},\\ \mathsf{Rev}(\mathsf{cons}\, x\, l) &= (\mathsf{Rev}\ l) :+: (\mathsf{cons}\, x\mathsf{Nil}).\end{aligned}$$

It can be defined as the term

$$\mathcal{R}^{\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)}\mathsf{Nil}(\lambda x\lambda l\lambda p.M_{:+:}\, p\, (\mathsf{cons}\, x\mathsf{Nil})).$$

Assume we want to define by simultaneous recursion two functions on $\mathbf{N}$, say $\mathsf{Even}, \mathsf{Odd}\colon \mathbf{N}\to\mathbf{B}$. We want

$$\begin{aligned}\mathsf{Even}(0) &:= \mathtt{tt} & \mathsf{Odd}(0) &:= \mathtt{ff}\\ \mathsf{Even}(\mathsf{S}\, n) &:= \mathsf{Odd}(n) & \mathsf{Odd}(\mathsf{S}\, n) &:= \mathsf{Even}(n)\end{aligned}$$

This can be achieved by using pair types: we recursively define the single function $\mathsf{EvenOdd}\colon \mathbf{N}\to\mathbf{B}\times\mathbf{B}$. The step types are

$$\begin{aligned}\delta_1 &= \mathbf{B}\times\mathbf{B},\\ \delta_2 &= \mathbf{N}\to\mathbf{B}\times\mathbf{B}\to\mathbf{B}\times\mathbf{B},\end{aligned}$$

and we can define $\mathsf{EvenOdd} := \mathcal{R}^{\mathbf{B}\times\mathbf{B}}_{\mathbf{N}}\langle\mathtt{ff},\mathtt{tt}\rangle(\lambda n\lambda p.\langle p1, p0\rangle)$.

Our final example concerns the simultaneously defined free algebras tree and tlist, whose constructors $\mathsf{constr}^{(\mathsf{tree},\mathsf{tlist})}_i$ for $i\in\{1,\dots,4\}$ are

$$\mathsf{Leaf}^{\mathbf{N}\to\mathsf{tree}}, \mathsf{Branch}^{\mathsf{tlist}\to\mathsf{tree}}, \mathsf{Empty}^{\mathsf{tlist}}, \mathsf{Tcons}^{\mathsf{tree}\to\mathsf{tlist}\to\mathsf{tlist}}.$$

Observe that the elements of the algebra tree are just the finitely branching trees, which carry natural numbers on their leaves.

Let us compute the types of the recursion operators w.r.t. the result types τ_1,τ_2, i.e. of $\mathcal{R}^{(\mathsf{tree},\mathsf{tlist}),(\tau_1,\tau_2)}_{\mathsf{tree}}$ and $\mathcal{R}^{(\mathsf{tree},\mathsf{tlist}),(\tau_1,\tau_2)}_{\mathsf{tlist}}$, or shortly $\mathcal{R}_{\mathsf{tree}}$ and $\mathcal{R}_{\mathsf{tlist}}$. The step types are

$$\begin{aligned}\delta_1 &:= \mathbf{N}\to\tau_1,\\ \delta_2 &:= \mathsf{tlist}\to\tau_2\to\tau_1,\\ \delta_3 &:= \tau_2,\\ \delta_4 &:= \mathsf{tree}\to\mathsf{tlist}\to\tau_1\to\tau_2\to\tau_2.\end{aligned}$$

Hence the types of the recursion operators are

$$\begin{aligned}\mathcal{R}_{\mathsf{tree}}&\colon \delta_1\to\delta_2\to\delta_3\to\delta_4\to\mathsf{tree}\to\tau_1,\\ \mathcal{R}_{\mathsf{tlist}}&\colon \delta_1\to\delta_2\to\delta_3\to\delta_4\to\mathsf{tlist}\to\tau_2.\end{aligned}$$

To see a concrete example, let us recursively define addition $+\colon \mathsf{tree}\to\mathsf{tree}\to\mathsf{tree}$ and $\oplus\colon \mathsf{tlist}\to\mathsf{tree}\to\mathsf{tlist}$. The recursion equations to be satisfied are

$$\begin{aligned}+\,(\mathsf{Leaf}\, n) &= \lambda a a,\\ +\,(\mathsf{Branch}\ bs) &= \lambda a.\mathsf{Branch}(\oplus\, bs\, a),\\ \\ \oplus\ \mathsf{Empty} &= \lambda a\,\mathsf{Empty},\\ \oplus\,(\mathsf{Tcons}\, b\ bs) &= \lambda a.\mathsf{Tcons}(+\, b\, a)(\oplus\ bs\, a).\end{aligned}$$

We define $+$ and $\oplus$ by means of the recursion operators $\mathcal{R}_{\mathsf{tree}}$ and $\mathcal{R}_{\mathsf{tlist}}$ with result types

$$\tau_1 := \mathsf{tree} \to \mathsf{tree},$$
$$\tau_2 := \mathsf{tree} \to \mathsf{tlist}.$$

The step terms are

$$M_1 := \lambda n \lambda a a,$$
$$M_2 := \lambda bs \lambda g^{\tau_2} \lambda a.\mathsf{Branch}(g\, a),$$
$$M_3 := \lambda a\, \mathsf{Empty},$$
$$M_4 := \lambda b \lambda bs \lambda f^{\tau_1} \lambda g^{\tau_2} \lambda a.\mathsf{Tcons}(f\, a)(g\, a).$$

Then

$$+ := \mathcal{R}_{\mathsf{tree}} \vec{M} \colon \mathsf{tree} \to \mathsf{tree} \to \mathsf{tree},$$
$$\oplus := \mathcal{R}_{\mathsf{tlist}} \vec{M} \colon \mathsf{tlist} \to \mathsf{tree} \to \mathsf{tlist}.$$

REMARK 2.2.1. It may happen that in a recursion on simultaneously defined algebras one only needs to recur on some of those algebras. Then we can simplify the type of the recursion operator accordingly, by

- omitting all step types $\delta_i^{\vec{\mu},\vec{\tau}}$ with irrelevant value type τ_j, and
- simplifying the remaining step types by omitting from the types $(\vec{\sigma}_1 \to \tau_{j_1}), \ldots, (\vec{\sigma}_n \to \tau_{j_n})$ of previously defined values all those with irrelevant τ_{j_ν}.

In the $\mathsf{tree}, \mathsf{tlist}$-example, if we only want to recur on tlist, then the step types are

$$\delta_3 := \tau_2,$$
$$\delta_4 := \mathsf{tree} \to \mathsf{tlist} \to \tau_2 \to \tau_2.$$

Hence the type of the simplified recursion operator is

$$\mathcal{R}_{\mathsf{tlist}} \colon \delta_3 \to \delta_4 \to \mathsf{tlist} \to \tau_2.$$

An example is the recursive definition of the length of a tlist. The recursion equations are

$$\mathsf{len}(\mathsf{Empty}) \quad = 0,$$
$$\mathsf{len}(\mathsf{Tcons}\, b\, bs) = \mathsf{len}(bs) + 1.$$

The step terms are

$$M_3 := \mathsf{Empty},$$
$$M_4 := \lambda b \lambda bs \lambda p.p + 1.$$

REMARK 2.2.2. There is an important variant of recursion, where no recursive calls occur. This variant is called the *cases operator*; it distinguishes cases according to the outer constructor form. Here all step types have the form

$$\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \to (\vec{\sigma}_1 \to \mu_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \mu_{j_n}) \to \tau_j.$$

The intended meaning of the cases operator is given by the conversion rule (cf. (5) below)

$$(\mathcal{C}_j \vec{M})^{\mu_j \to \tau_j}(\mathsf{constr}_i^{\vec{\mu}} \vec{N}) \mapsto M_i \vec{N}.$$

Notice that only those step terms are used whose value type is the present τ_j; this is due to the fact that there are no recursive calls. Therefore the type of the cases operator is

$$\mathcal{C}^{\vec{\mu}}_{\mu_j\to\tau_j}\colon \delta_{i_1}\to\cdots\to\delta_{i_q}\to\mu_j\to\tau_j,$$

where $\delta_{i_1},\dots,\delta_{i_q}$ consists of all δ_i with value type τ_j. We write $\mathcal{C}_{\mu_j\to\tau_j}$ or even $\mathcal{C}_j$ for $\mathcal{C}^{\vec{\mu}}_{\mu_j\to\tau_j}$.

The simplest example (for the type $\mathbf{B}$) is *if-then-else*. Another example is the predecessor function on $\mathbf{N}$, i.e. $\mathsf{P}(0) := 0$, $\mathsf{P}(\mathsf{S}(n)) := n$. It can formally be defined by the term

$$\mathcal{C}_{\mathbf{N}\to\mathbf{N}}0(\lambda nn).$$

In the $\mathsf{tree}, \mathsf{tlist}$-example we have

$$\mathcal{C}_{\mathsf{tlist}\to\tau_2}\colon \tau_2\to(\mathsf{tree}\to\mathsf{tlist}\to\tau_2)\to\mathsf{tlist}\to\tau_2.$$

REMARK 2.2.3. When computing the value of a cases operator, we may not want to (eagerly) evaluate all arguments, but rather evaluate the test argument first and depending on the result (lazily) evaluate at most one of the other arguments. This phenomenon is well known in functional languages; e.g. in SCHEME the `if`-construct is called a *special form* (as opposed to an operator). Therefore we also provide an `if`-construct to build terms, which differs from the cases operator only in that it employs lazy evaluation. The predecessor function could then be written in the form $\lambda m[\texttt{if}\ m\ 0\ \lambda nn]$.

2.2.3. Conversion Relation. It will be useful to employ the following notation. Let $\vec{\mu}=\mu\vec{\alpha}\,\vec{\kappa}$ and

$$\kappa_i=\rho_1\to\cdots\to\rho_m\to(\vec{\sigma}_1\to\alpha_{j_1})\to\cdots\to(\vec{\sigma}_n\to\alpha_{j_n})\to\alpha_j\in\mathsf{KT}(\vec{\alpha}),$$

and consider $\mathsf{constr}^{\vec{\mu}}_i\vec{N}$. Then we write $\vec{N}^P=N^P_1,\dots,N^P_m$ for the *parameter arguments* $N^{\rho_1}_1,\dots,N^{\rho_m}_m$ and $\vec{N}^R=N^R_1,\dots,N^R_n$ for the *recursive arguments* $N^{\vec{\sigma}_1\to\mu_{j_1}}_{m+1},\dots,N^{\vec{\sigma}_n\to\mu_{j_n}}_{m+n}$, and n^R for the number n of recursive arguments.

We define a *conversion relation* $\mapsto_\rho$ between terms of type ρ by

(1) $$(\lambda xM)N\mapsto M[x:=N]$$

(2) $$\langle M_0,M_1\rangle i\mapsto Mi\quad(i=0,1)$$

(3) $$\lambda x.Mx\mapsto M\quad\text{if } x\notin\mathsf{FV}(M)\ (M\text{ not an abstraction})$$

(4) $$\langle M0,M1\rangle\mapsto M\quad(M\text{ not a pair})$$

(5) $$(\mathcal{R}_j\vec{M})^{\mu_j\to\tau_j}(\mathsf{constr}^{\vec{\mu}}_i\vec{N})\mapsto M_i\vec{N}\big((\mathcal{R}_{j_1}\vec{M})\circ N^R_1\big)\dots\big((\mathcal{R}_{j_n}\vec{M})\circ N^R_n\big)$$

Here we have written $\mathcal{R}_j$ for $\mathcal{R}^{\vec{\mu},\vec{\tau}}_{\mu_j}$.

The *one step reduction relation* $\to$ can now be defined as follows. $M\to N$ if N is obtained from M by replacing a subterm M' in M by N', where $M'\mapsto N'$. The reduction relations $\to^+$ and $\to^*$ are the transitive and the reflexive transitive closure of $\to$, respectively. For $\vec{M}=M_1,\dots,M_n$ we write $\vec{M}\to\vec{M}'$ if $M_i\to M'_i$ for some $i\in\{1,\dots,n\}$ and $M_j=M'_j$ for all $i\neq j\in\{1,\dots,n\}$. A term M is *normal* (or in *normal form*) if there is no term N such that $M\to N$.

Clearly normal closed terms are of the form $\mathsf{constr}^{\vec{\mu}}_i\vec{N}$.

EXAMPLE. Let us check the conversion rules for the defined $+$ and $\oplus$ of our example above. We have

$$\begin{aligned} + \,(\mathsf{Leaf}\; n) &= \mathcal{R}_{\mathsf{tree}}\vec{M}(\mathsf{Leaf}\; n) \mapsto M_1 n \mapsto \lambda a a, \\ \oplus\; \mathsf{Empty} &= \mathcal{R}_{\mathsf{tlist}}\vec{M}\;\mathsf{Empty} \mapsto M_3 = \lambda a\,\mathsf{Empty} \end{aligned}$$

and

$$\begin{aligned} +(\mathsf{Branch}\; bs) &= \mathcal{R}_{\mathsf{tree}}\vec{M}(\mathsf{Branch}\; bs) \\ &\mapsto M_2\; bs\; \big((\mathcal{R}_{\mathsf{tlist}}\vec{M}) \circ bs\big) \\ &= M_2\; bs\; (\oplus\; bs) \\ &\to \lambda a.\mathsf{Branch}(\oplus\; bs\; a), \end{aligned}$$

$$\begin{aligned} \oplus(\mathsf{Tcons}\; b\; bs) &= \mathcal{R}_{\mathsf{tlist}}\vec{M}(\mathsf{Tcons}\; b\; bs) \\ &\mapsto M_4\; b\; bs\; \big((\mathcal{R}_{\mathsf{tree}}\vec{M}) \circ b\big)\big((\mathcal{R}_{\mathsf{tlist}}\vec{M}) \circ bs\big) \\ &= M_4\; b\; bs\; (+\,b)(\oplus\; bs) \\ &\to \lambda a.\mathsf{Tcons}(+\,b\; a)(\oplus\; bs\; a). \end{aligned}$$

2.2.4. Strong Computability Predicates.

DEFINITION. The set SN of *strongly normalizable* terms is inductively defined by

$$(\forall N.M \to N \Rightarrow N \in \mathsf{SN}) \Rightarrow M \in \mathsf{SN} \tag{6}$$

Note that with M clearly every subterm of M is strongly normalizable.

DEFINITION. We define *strong computability predicates* SC^ρ by induction on ρ.

Case $\mu_j = (\mu\vec{\alpha}\,\vec{\kappa})_j$. Then $M \in \mathsf{SC}^{\mu_j}$ if

$$\forall N.M \to N \Rightarrow N \in \mathsf{SC}, \text{ and} \tag{7}$$

$$M = \mathsf{constr}_i^{\vec{\mu}}\vec{N} \Rightarrow \vec{N}^P \in \mathsf{SC} \wedge \bigwedge\!\!\!\bigwedge_{p=1}^{n^R} (\forall \vec{K}{\in}\mathsf{SC})\, N_p^R\vec{K} \in \mathsf{SC}^{\mu_{j_p}}. \tag{8}$$

Case $\rho \to \sigma$.

$$M \in \mathsf{SC}^{\rho\to\sigma} \;:\Longleftrightarrow\; (\forall N{\in}\mathsf{SC}^\rho)\, MN \in \mathsf{SC}^\sigma.$$

Case $\rho \times \sigma$.

$$M \in \mathsf{SC}^{\rho\times\sigma} \;:\Longleftrightarrow\; M0 \in \mathsf{SC}^\rho \text{ and } M1 \in \mathsf{SC}^\sigma.$$

Notice that the reference to $\vec{N}^P \in \mathsf{SC}$ and $\vec{K}{\in}\mathsf{SC}$ in (8) is legal, because the types $\vec{\rho}, \vec{\sigma}_i$ of $\vec{N}, \vec{K}$ must have been generated *before* μ_j. Note also that by (8) $\mathsf{constr}_i^{\vec{\mu}}\vec{N} \in \mathsf{SC}$ implies $\vec{N} \in \mathsf{SC}$.

We now set up a sequence of lemmas leading to a proof that every term is strongly normalizing. In the proofs we disregard the product case, since it can be treated routinely.

LEMMA 2.2.4. *If* $M \in \mathsf{SC}^\rho$ *and* $M \to M'$, *then* $M' \in \mathsf{SC}$.

PROOF. Induction on ρ. *Case* μ. By (7). *Case* $\rho \to \sigma$. Assume $M \in \mathsf{SC}^{\rho\to\sigma}$ and $M \to M'$; we must show $M' \in \mathsf{SC}$. So let $N \in \mathsf{SC}^\rho$; we must show $M'N \in \mathsf{SC}^\sigma$. But this follows from $MN \to M'N$ and $MN \in \mathsf{SC}^\rho$ by induction hypothesis (IH) on σ. □

LEMMA 2.2.5. $(\forall \vec{M}{\in}\mathsf{SN}).\vec{M} \in \mathsf{SC} \Rightarrow (x\vec{M})^\mu \in \mathsf{SC}$.

PROOF. Induction on $\vec{M} \in \mathsf{SN}$. Assume $\vec{M} \in \mathsf{SN}$ and $\vec{M} \in \mathsf{SC}$; we must show $(x\vec{M})^\mu \in \mathsf{SC}$. So assume $x\vec{M} \to N$; we must show $N \in \mathsf{SC}$. Now by the form of the conversion rules N must be of the form $x\vec{M'}$ with $\vec{M} \to \vec{M'}$. But $\vec{M'} \in \mathsf{SC}$ by Lemma 2.2.4, hence $x\vec{M'} \in \mathsf{SC}$ by IH for $\vec{M'}$. □

LEMMA 2.2.6. (a) $\mathsf{SC}^\rho \subseteq \mathsf{SN}$,
(b) $x \in \mathsf{SC}^\rho$.

PROOF. By simultaneous induction on ρ. *Case* $\mu_j = (\mu\vec{\alpha}\,\vec{\kappa})_j$. (a). We show $M \in \mathsf{SC}^{\mu_j} \Rightarrow M \in \mathsf{SN}$ by (side) induction on $M \in \mathsf{SC}^{\mu_j}$. So assume $M \in \mathsf{SC}^{\mu_j}$; we must show $M \in \mathsf{SN}$. But for every N with $M \to N$ we have $N \in \mathsf{SC}$ by (7), hence $N \in \mathsf{SN}$ by the side induction hypothesis SIH. (b). $x \in \mathsf{SC}^{\mu_j}$ holds trivially.

Case $\rho \to \sigma$. (a). Assume $M \in \mathsf{SC}^{\rho\to\sigma}$; we must show $M \in \mathsf{SN}$. By IH(b) for ρ we have $x \in \mathsf{SC}^\rho$, hence $Mx \in \mathsf{SC}^\sigma$, hence $Mx \in \mathsf{SN}$ by IH(a) for σ. But $Mx \in \mathsf{SN}$ clearly implies $M \in \mathsf{SN}$. (b). Let $\vec{M} \in \mathsf{SC}^{\vec{\rho}}$ with $\rho_1 = \rho$; we must show $x\vec{M} \in \mathsf{SC}^\mu$. But this follows from Lemma 2.2.5, using IH(a) for $\vec{\rho}$. □

COROLLARY 2.2.7. $\vec{N} \in \mathsf{SC} \Rightarrow \mathsf{constr}_i^{\vec{\mu}}\vec{N} \in \mathsf{SC}$, *i.e.* $\mathsf{constr}_i^{\vec{\mu}} \in \mathsf{SC}$.

PROOF. First show $(\forall \vec{N}{\in}\mathsf{SN}).\vec{N} \in \mathsf{SC} \Rightarrow \mathsf{constr}_i^{\vec{\mu}}\vec{N} \in \mathsf{SC}$ by induction on $\vec{N} \in \mathsf{SN}$ as in Lemma 2.2.5, and then use Lemma 2.2.6(a). □

LEMMA 2.2.8.

$$(\forall M, N, \vec{N}{\in}\mathsf{SN}).M[x{:=}N]\vec{N} \in \mathsf{SC}^\mu \Rightarrow (\lambda xM)N\vec{N} \in \mathsf{SC}^\mu.$$

$$(\forall M_0, M_1, \vec{N}{\in}\mathsf{SN}).M_0\vec{N}, M_1\vec{N} \in \mathsf{SC}^\mu \Rightarrow \langle M_0, M_1\rangle i\vec{N} \in \mathsf{SC}^\mu.$$

PROOF. By induction on $M, N, \vec{N} \in \mathsf{SN}$. Let $M, N, \vec{N} \in \mathsf{SN}$ and assume $M[x{:=}N]\vec{N} \in \mathsf{SC}$; we must show $(\lambda xM)N\vec{N} \in \mathsf{SC}$. Assume $(\lambda xM)N\vec{N} \to K$; we must show $K \in \mathsf{SC}$. *Case* $K = (\lambda xM')N'\vec{N'}$ with $M, N, \vec{N} \to M', N', \vec{N'}$. Then $M[x{:=}N]\vec{N} \to^* M'[x{:=}N']\vec{N'}$, hence by (7) from our assumption $M[x{:=}N]\vec{N} \in \mathsf{SC}$ we can infer $M'[x{:=}N']\vec{N'} \in \mathsf{SC}$, therefore $(\lambda xM')N'\vec{N'} \in \mathsf{SC}$ by IH. *Case* $K = M[x{:=}N]\vec{N}$. Then $K \in \mathsf{SC}$ by assumption. □

COROLLARY 2.2.9.

$$(\forall M, N, \vec{N}{\in}\mathsf{SN}).M[x{:=}N]\vec{N} \in \mathsf{SC}^\rho \Rightarrow (\lambda xM)N\vec{N} \in \mathsf{SC}^\rho.$$

$$(\forall M_0, M_1, \vec{N}{\in}\mathsf{SN}).M_0\vec{N}, M_1\vec{N} \in \mathsf{SC}^\rho \Rightarrow \langle M_0, M_1\rangle i\vec{N} \in \mathsf{SC}^\rho.$$

PROOF. By induction on ρ, using Lemma 2.2.6(a). □

LEMMA 2.2.10. $(\forall N{\in}\mathsf{SC}^{\mu_j})(\forall \vec{M}, \vec{L}{\in}\mathsf{SN}).\vec{M}, \vec{L} \in \mathsf{SC} \Rightarrow \mathcal{R}_j\vec{M}N\vec{L} \in \mathsf{SC}^\mu$.

PROOF. By main induction on $N \in \mathsf{SC}^{\mu_j}$, and side induction on $\vec{M}, \vec{L} \in \mathsf{SN}$. Assume

$$\mathcal{R}_j \vec{M} N \vec{L} \to L.$$

We must show $L \in \mathsf{SC}$.

Case 1. $\mathcal{R}_j \vec{M}' N \vec{L}' \in \mathsf{SC}$ by the SIH.

Case 2. $\mathcal{R}_j \vec{M} N' \vec{L} \in \mathsf{SC}$ by the main induction hypothesis (IH).

Case 3. $N = \mathsf{constr}_i^{\vec{\mu}} \vec{N}$ and

$$L = M_i \vec{N}\big((\mathcal{R}_j \vec{M}) \circ N_1^R\big) \dots \big((\mathcal{R}_j \vec{M}) \circ N_n^R\big) \vec{L}.$$

$\vec{M}, \vec{L} \in \mathsf{SC}$ by assumption. $\vec{N} \in \mathsf{SC}$ follows from $N = \mathsf{constr}_i^{\vec{\mu}} \vec{N} \in \mathsf{SC}$ by (8). Note that for all recursive arguments N_p^R of N and all strongly computable $\vec{K}$ by (8) we have the IH for $N_p^R \vec{K}$ available. It remains to show $(\mathcal{R}_j \vec{M}) \circ N_p^R = \lambda \vec{x}_p . \mathcal{R}_j \vec{M}(N_p^R \vec{x}_p) \in \mathsf{SC}$. So let $\vec{K}, \vec{Q} \in \mathsf{SC}$ be given. We must show $(\lambda \vec{x}_p . \mathcal{R}_j \vec{M}(N_p^R \vec{x}_p)) \vec{K} \vec{Q} \in \mathsf{SC}$. By IH for $N_p^R \vec{K}$ we have $\mathcal{R}_j \vec{M}(N_p^R \vec{K}) \vec{Q} \in \mathsf{SC}$, since by Lemma 2.2.6(a) $\vec{K}, \vec{Q} \in \mathsf{SN}$. Now Corollary 2.2.9 yields the claim. □

COROLLARY 2.2.11. $\mathcal{R}_j \in \mathsf{SC}$. □

DEFINITION. A substitution ξ is *strongly computable*, if $\xi(x) \in \mathsf{SC}$ for all variables x. A term M is *strongly computable under substitution*, if $M\xi \in \mathsf{SC}$ for all strongly computable substitutions ξ.

THEOREM 2.2.12. *Every term is strongly computable under substitution.*

PROOF. Induction on the term M. *Case* x. $x\xi \in \mathsf{SC}$, since ξ is strongly computable. *Case* $\mathsf{constr}_i^{\vec{\mu}}$. By Corollary 2.2.7. *Case* $\mathcal{R}_j$. By Corollary 2.2.11. *Case* MN. By IH $M\xi, N\xi \in \mathsf{SC}$, hence $(MN)\xi = (M\xi)(N\xi) \in \mathsf{SC}$. *Case* $\lambda x M$. Let ξ be a strongly computable substitution; we must show $(\lambda x M)\xi = \lambda x M \xi_x^x \in \mathsf{SC}$. So let $N \in \mathsf{SC}$; we must show $(\lambda x M \xi_x^x) N \in \mathsf{SC}$. By IH $M\xi_x^N \in \mathsf{SC}$, hence $(\lambda x M \xi_x^x) N \in \mathsf{SC}$ by Corollary 2.2.9. □

COROLLARY 2.2.13. *Every term is strongly normalizable.* □

2.3. Rewrite Rules

The elimination constants corresponding to the constructors are called primitive recursion operators $\mathcal{R}$. They have been described in detail in Section 2.2. In this setup, every closed term reduces to a numeral.

For convenience, we shall also use constants for rather arbitrary computable functionals, and axiomatize them according to their intended meaning by means of rewrite rules. An example is the general fixed point operator fix, which is axiomatized by $\mathsf{fix}F = F(\mathsf{fix}F)$. Clearly then it cannot be true any more that every closed term reduces to a numeral. We may have non-terminating terms, but this just means that not always it is a good idea to try to normalize a term.

An important consequence of admitting non-terminating terms is that our notion of proof is not decidable: when checking e.g. whether two terms are equal we may run into a non-terminating computation. To avoid this somewhat unpleasant undecidability phenomenon, we may view our proofs

as abbreviated forms of full proofs, with certain equality arguments left implicit. If some information sufficient to recover the full proof (e.g. for each node a bound on the number of rewrite steps needed to verify it) is stored as part of the proof, then we retain decidability of proofs.

However, even without such additional information we still have semi-decidability of proofs, i.e., an algorithm to check the correctness of a proof that can only give correct results, but may not terminate. In practice this is sufficient.

We now describe in some detail our concept of rewrite rules. For every program constant c^ρ we assume that some rewrite rules of the form $c\vec{K} \mapsto N$ are given, where $\mathsf{FV}(N) \subseteq \mathsf{FV}(\vec{K})$ and $c\vec{K}$, N have the same type (not necessarily a ground type). Moreover, for any two rules $c\vec{K} \mapsto N$ and $c\vec{K}' \mapsto N'$ we require that $\vec{K}$ and $\vec{K}'$ are of the same length, called the *arity* of c.

Given a set of rewrite rules, we want to treat some rules - which we call *computation rules* - in a different, more efficient way (cf. [**9**]). The idea is that a computation rule can be understood as a description of a computation in a suitable *semantical model*, provided the syntactic constructors correspond to semantic ones in the model, whereas the other rules describe *syntactic* transformations.

In order to define what we mean by computation rules, we need the notion of a *constructor pattern*. These are special terms defined inductively as follows.

- Every variable is a constructor pattern.
- If c is a constructor and $P_1, \dots, P_n$ are constructor patterns (or projection markers 0 or 1) such that $c\vec{P}$ is of ground type, then $c\vec{P}$ is a constructor pattern.

From the given set of rewrite rules we choose a subset Comp with the following properties.

- If $c\vec{P} \mapsto Q \in$ Comp, then $P_1, \dots, P_n$ are constructor patterns or projection markers.
- The rules are left-linear, i.e. if $c\vec{P} \mapsto Q \in$ Comp, then every variable in $c\vec{P}$ occurs only once in $c\vec{P}$.
- The rules are non-overlapping, i.e. for different rules $c\vec{K} \mapsto M$ and $c\vec{L} \mapsto N$ in Comp the left hand sides $c\vec{K}$ and $c\vec{L}$ are non-unifiable.

We write $c\vec{M} \mapsto_{\mathsf{comp}} Q$ to indicate that the rule is in Comp. All other rules will be called (proper) rewrite rules, written $c\vec{M} \mapsto_{\mathsf{rew}} K$.

In our reduction strategy computation rules will always be applied first, and since they are non-overlapping, this part of the reduction is unique. However, since we allowed almost arbitrary rewrite rules, it may happen that in case no computation rule applies a term may be rewritten by different rules $\notin$ Comp. In order to obtain a deterministic procedure we then select the first applicable rewrite rule (This is a slight simplification of [**9**], where special functions sel_c were used for this purpose).

2.4. Axioms

The intended model of our theory is a many-sorted structure, with one sort for every type. We assume that the model consists of *domains*, in the sense of domain theory (cf. [**34**]). The reason for the setting is that we want to deal with *computable functionals*. Since their (mathematically correct) domains are the Scott-Ershov partial continuous functionals, this is the intended range of the quantifiers.

2.4.1. Languages for Algebras. We now define the specific logical language we use for our algebras. It should be thought of as a form of arithmetical language, since it is supposed to describe our particular intended model.

A *variable* of a given type is interpreted by a continuous functional (object) of that type. We use the word "variable" and not "program variable", since continuous functionals are not necessarily computable. So for each type ρ we have *general variables* $\hat{x}^\rho, \hat{y}^\rho, \ldots$ of type ρ.

In most cases we need to argue about existing (i.e. total) objects only. For the notion of totality we have to refer to [**34**, Chapter 8.3]; particularly relevant here is exercise 8.5.7. To make formal arguments with quantifiers relativized to total objects more managable, we use a special sort of variables intended to range over such objects only. So for each type ρ we have *total variables* $x^\rho, y^\rho, \ldots$ of type ρ.

For readable in- and output, and also for ease in parsing, we may reserve certain strings as names for variables of a given type, e.g. n, m for variables of type $\mathbf{N}$. Then also $n_0, n_1, n_2, \ldots, m_0, \ldots$ can be used for the same purpose. For example, $n_0, n_1, n_2, \ldots, m_0, \ldots$ range over total natural numbers, and $\hat{n}_0, \hat{n}_1, \hat{n}_2, \ldots$ are general variables. We say that the *degree of totality* for the former is 1, and for the latter 0.

A *predicate variable* $\hat{P}$ of arity $\rho_1, \ldots, \rho_n$ is a placeholder for a formula A with distinguished (different) variables $\hat{x}_1, \ldots, \hat{x}_n$ of types $\rho_1, \ldots, \rho_n$. Such an entity is called a *comprehension term*, written $\{\, \hat{x}_1, \ldots, \hat{x}_n \mid A \,\}$. By default we have the predicate variable $\bot$ (of empty arity), called (logical) *falsity*. It is viewed as a predicate variable rather than a predicate constant, since (when translating a classical proof into a constructive one) we want to substitute for $\bot$.

Often we will argue about *Harrop formulas* only, i.e. formulas without computational content. For convenience we use a special sort of predicate variables intended to range over comprehension terms with Harrop formulas only. For example, $P_0, P_1, P_2, \ldots, Q_0, \ldots$ range over comprehension terms with Harrop formulas, and $\hat{P}_0, \hat{P}_1, \hat{P}_2, \ldots$ are general predicate variables. We say that *Harrop degree* for the former is 1, and for the latter 0.

We also allow predicate constants with a fixed intended meaning. Predicate variables and constants are both called predicate symbols. The need for predicate constants comes up when e.g. an inductively defined set is expressed via a formula stating the existence of a generation tree; the kernel of this formula is to be axiomatized, using the tree constructors. Prime formulas built from predicate constants do not give rise to extracted terms, and cannot be substituted for.

Specific predicate constants are

- atom of arity $(\mathbf{B})$,
- for every type ρ *equality* $\approx_\rho$ of arity (ρ, ρ), and
- for every type ρ *totality* Total_ρ of arity (ρ).

Notice that for finitary algebras, e.g. $\mathbf{N}$, we have continuous boolean valued functions equality $=_{nat}\colon \mathbf{N} \to \mathbf{N} \to \mathbf{B}$ and existence (definedness, totality) $e_{nat}\colon \mathbf{N} \to \mathbf{B}$. Then we can express equality $r = s$ by $\mathsf{atom}(=(r, s))$ and existence $E(r)$ by $\mathsf{atom}(e(r))$.

A prime formula has the form $P(r_1, \dots, r_n)$ with a predicate variable or constant P and terms $r_1, \dots, r_n$. Write

- T, F for $\mathsf{atom}(\mathsf{tt})$, $\mathsf{atom}(\mathsf{ff})$,
- $r = s$ for $\mathsf{atom}(=(r, s))$,
- $E(r)$ for $\mathsf{atom}(e(r))$, and
- $r \approx s$ for $\approx (r, s)$.

Formulas are built from prime formulas by

- implication $A \to B$,
- conjunction $A \wedge B$,
- tensor $A \otimes B$,
- all quantification $\forall \hat{x}^\rho A$ and
- existential quantification $\exists \hat{x}^\rho A$.

Moreover we have classical existential quantification in an arithmetical and a logical form:

$$\exists^{\mathsf{ca}} \hat{x}_1 \dots \hat{x}_n . A_1 \otimes \dots \otimes A_m \quad \text{arithmetical version}$$
$$\exists^{\mathsf{cl}} \hat{x}_1 \dots \hat{x}_n . A_1 \otimes \dots \otimes A_m \quad \text{logical version.}$$

For all quantifiers we allow that the quantified variable is formed without $\hat{}$, i.e. ranges over total objects only.

Remark. When dealing with the classical existential quantifier, it is – for obvious reasons – useful to be able to unfold and fold it as it seems appropriate. In Minlog the commands are `fold-formula` and `unfold-formula`. However, notice that there are some slight difficulties in this context. To see the problems, consider

$$\exists^{\mathsf{cl}} x, y A$$

If we read this as $\exists^{\mathsf{cl}} x \exists^{\mathsf{cl}} y A$, then the unfolded form would be

$$\neg \forall x \neg \neg \forall y \neg A$$

However, it would be simpler if we could unfold this formula into the equivalent

$$\neg \forall x \forall y \neg A$$

To achive this effect, we allow lists of variables after a classical existential quantifier, and unfold $\exists^{\mathsf{cl}} x, y A$ into the latter (shorter) formula, but $\exists^{\mathsf{cl}} x \exists^{\mathsf{cl}} y A$ into the former.

Another (small) problem arises when we want to fold

$$\neg \forall x . A \to B \to \bot \tag{9}$$

The result should be $\exists^{\mathsf{cl}}x.A \wedge B$, but this is not quite correct, since the latter formula unfolds into $\neg\forall x.A \wedge B \to \bot$. Therefore in MINLOG there is a connective called *tensor* (written $\otimes$) with the property that (9) folds into

$$\exists^{\mathsf{cl}}x.A \otimes B$$

and unfolds again into (9).

Formulas can be *unfolded* in the sense that all classical existential quantifiers are replaced according to their definiton

$$\exists^{\mathsf{ca}}\hat{x}_1 \dots \hat{x}_n.A_1 \otimes \dots \otimes A_m := (\forall \hat{x}_1 \dots \hat{x}_n.A_1 \to \dots \to A_m \to F) \to F$$
$$\exists^{\mathsf{cl}}\hat{x}_1 \dots \hat{x}_n.A_1 \otimes \dots \otimes A_m := (\forall \hat{x}_1 \dots \hat{x}_n.A_1 \to \dots \to A_m \to \bot) \to \bot$$

Inversely a formula can be *folded* in the sense that classical existential quantifiers are introduced wherever possible.

Comprehension terms have the form $\{\, \vec{x} \mid A \,\}$; note that the formula A may contain further free variables.

2.4.2. Algebras and Totality. We use the natural numbers as a prototypical finitary algebra; recall Figure 1. Assume that n, p are variables of type $\mathbf{N}$, $\mathbf{B}$. Let $\approx$ denote the equality relation in the model. Recall the domain of type $\mathbf{B}$, consisting of tt, ff and the bottom element bb. The boolean valued functions equality $=_{nat}\colon \mathbf{N} \to \mathbf{N} \to \mathbf{B}$ and existence (definedness, totality) $e_{nat}\colon \mathbf{N} \to \mathbf{B}$ need to be continuous. So we have

$$\begin{aligned} =(0,0) &\approx \mathsf{tt} & & \\ =(0, S\hat{n}) \approx =(S\hat{n}, 0) &\approx \mathsf{ff} & e(0) &\approx \mathsf{tt} \\ =(S\hat{n}_1, S\hat{n}_2) &\approx =(\hat{n}_1, \hat{n}_2) & e(S\hat{n}) &\approx e(\hat{n}) \\ =(\mathsf{bb}_{nat}, \hat{n}) \approx =(\hat{n}, \mathsf{bb}_{nat}) &\approx \mathsf{bb} & e(\mathsf{bb}_{\mathbf{N}}) &\approx \mathsf{bb} \\ =(\infty_{nat}, \hat{n}) \approx =(\hat{n}, \infty_{nat}) &\approx \mathsf{bb} & e(\infty_{\mathbf{N}}) &\approx \mathsf{bb} \end{aligned}$$

We stipulate as axioms

T	`Truth-Axiom`
$\hat{x} \approx \hat{x}$	`Eq-Refl`
$\hat{x}_1 \approx \hat{x}_2 \to \hat{x}_2 \approx \hat{x}_1$	`Eq-Symm`
$\hat{x}_1 \approx \hat{x}_2 \to \hat{x}_2 \approx \hat{x}_3 \to \hat{x}_1 \approx \hat{x}_3$	`Eq-Trans`
$\forall \hat{x} \hat{f}_1 \hat{x} \approx \hat{f}_2 \hat{x} \to \hat{f}_1 \approx \hat{f}_2$	`Eq-Ext`
$\hat{x}_1 \approx \hat{x}_2 \to \hat{P}(\hat{x}_1) \to \hat{P}(\hat{x}_2)$	`Eq-Compat`
$\forall \hat{x}_1, \hat{x}_2 \hat{P}(\langle \hat{x}_1, \hat{x}_2 \rangle) \to \forall \hat{p} \hat{P}(\hat{p})$	`Pair-Elim`
$\mathsf{Total}_{\rho\to\sigma}(\hat{f}) \leftrightarrow \forall \hat{x}.\mathsf{Total}_{\rho}(\hat{x}) \to \mathsf{Total}_{\sigma}(\hat{f}\hat{x})$	`Total`
$\mathsf{Total}_{\rho}(c)$	`Constr-Total`
$\mathsf{Total}(c\vec{\hat{x}}) \to \mathsf{Total}(\hat{x}_i)$	`Constr-Total-Args`

and for every finitary algebra, e.g. `nat`

$\hat{n}_1 \approx \hat{n}_2 \to E(\hat{n}_1) \to \hat{n}_1 = \hat{n}_2$	`Eq-to-=-1-nat`

$\hat{n}_1 \approx \hat{n}_2 \to E(\hat{n}_2) \to \hat{n}_1 = \hat{n}_2$ `Eq-to-=-2-nat`

$\hat{n}_1 = \hat{n}_2 \to \hat{n}_1 \approx \hat{n}_2$ `=-to-Eq-nat`

$\hat{n}_1 = \hat{n}_2 \to E(\hat{n}_1)$ `=-to-E-1-nat`

$\hat{n}_1 = \hat{n}_2 \to E(\hat{n}_2)$ `=-to-E-2-nat`

$\mathsf{Total}(\hat{n}) \to E(\hat{n})$ `Total-to-E-nat`

$E(\hat{n}) \to \mathsf{Total}(\hat{n})$ `E-to-Total-nat`

Here c is a constructor. Notice that in $\mathsf{Total}(c\vec{\hat{x}}) \to \mathsf{Total}(\hat{x}_i)$, the type of $c\vec{\hat{x}}$ need not be a finitary algebra, and hence $\hat{x}_i$ may have a function type.

REMARK. $(E(\hat{n}_1) \to \hat{n}_1 = \hat{n}_2) \to (E(\hat{n}_2) \to \hat{n}_1 = \hat{n}_2) \to \hat{n}_1 \approx \hat{n}_2$ is *not* valid in our intended model (see Figure 1), since we have *two* distinct undefined objects $\mathbb{bb}_{nat}$ and ∞_{nat}.

We abbreviate

$$\forall \hat{x}.\mathsf{Total}_\rho(\hat{x}) \to A \quad \text{by} \quad \forall x A,$$
$$\exists \hat{x}.\mathsf{Total}_\rho(\hat{x}) \wedge A \quad \text{by} \quad \exists x A.$$

Formally, these abbreviations appear as axioms

$\forall x \hat{P}(x) \to \forall \hat{x}.\mathsf{Total}(\hat{x}) \to \hat{P}(\hat{x})$ `All-AllPartial`

$(\forall \hat{x}.\mathsf{Total}(\hat{x}) \to \hat{P}(\hat{x})) \to \forall x \hat{P}(x)$ `AllPartial-All`

$\exists x \hat{P}(x) \to \exists \hat{x}.\mathsf{Total}(\hat{x}) \wedge \hat{P}(\hat{x})$ `Ex-ExPartial`

$(\exists \hat{x}.\mathsf{Total}(\hat{x}) \wedge \hat{P}(\hat{x})) \to \exists x \hat{P}(x)$ `ExPartial-Ex`

and for every finitary algebra, e.g. `nat`

$\forall n \hat{P}(n) \to \forall \hat{n}.E(\hat{n}) \to \hat{P}(\hat{n})$ `All-AllPartial-nat`

$(\exists \hat{n}.E(\hat{n}) \wedge \hat{P}(\hat{n})) \to \exists n \hat{P}(n)$ `ExPartial-Ex-nat`

Notice that `AllPartial-All-nat` i.e. $(\forall \hat{n}.E(\hat{n}) \to \hat{P}(\hat{n})) \to \forall n \hat{P}(n)$ is provable (since $E(n) \mapsto T$). Similarly, `Ex-ExPartial-nat`, i.e. $\exists n \hat{P}(n) \to \exists \hat{n}.E(\hat{n}) \wedge \hat{P}(\hat{n})$ is provable.

Finally we have axioms for the existential quantifier

$\forall \hat{x}^\alpha.\hat{P}(\hat{x}) \to \exists \hat{x}^\alpha \hat{P}(\hat{x})$ `Ex-Intro`

$\exists \hat{x}^\alpha \hat{P}(\hat{x}) \to (\forall \hat{x}^\alpha.\hat{P}(\hat{x}) \to \hat{Q}) \to \hat{Q}$ `Ex-Elim`

2.4.3. Induction. We now spell out what we mean by induction over simultaneous free algebras $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$, with goal formulas $\forall x_j^{\mu_j} \hat{P}_j(x_j)$. For the constructor type

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$$

we have the *step formula*

$$D_i := \forall y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1 \to \mu_{j_1}}, \ldots, y_{m+n}^{\vec{\sigma}_n \to \mu_{j_n}}.\forall \vec{x}^{\vec{\sigma}_1} \hat{P}_{j_1}(y_{m+1}\vec{x}) \to \cdots \to \forall \vec{x}^{\vec{\sigma}_n} \hat{P}_{j_n}(y_{m+n}\vec{x}) \to \hat{P}_j(\mathsf{constr}_i^{\vec{\mu}}(\vec{y})).$$

Here $\vec{y} = y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1 \to \mu_{j_1}}, \ldots, y_{m+n}^{\vec{\sigma}_n \to \mu_{j_n}}$ are the *components* of the object $\mathsf{constr}_i^{\vec{\mu}}(\vec{y})$ of type μ_j under consideration, and

$$\forall \vec{x}^{\vec{\sigma}_1}\, \hat{P}_{j_1}(y_{m+1}\vec{x}), \ldots, \forall \vec{x}^{\vec{\sigma}_n}\, \hat{P}_{j_n}(y_{m+n}\vec{x})$$

are the hypotheses available when proving the induction step. The induction axiom $\mathsf{Ind}_{\mu_j}^{\vec{x},\vec{A}}$ with $\vec{x} = (x_j^{\mu_j})_{j=1,\ldots,N}$ and $\vec{A} = (A_j)_{j=1,\ldots,N} = (\hat{P}_j(x_j^{\mu_j}))_{j=1,\ldots,N}$ then proves the formula

$$D_1 \to \cdots \to D_k \to \forall x_j^{\mu_j}\, \hat{P}_j(x_j).$$

We will often write $\mathsf{Ind}_j^{\vec{x},\vec{A}}$ for $\mathsf{Ind}_{\mu_j}^{\vec{x},\vec{A}}$, and omit the upper indices $\vec{x}, \vec{A}$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e. of type $\mu\alpha\,\kappa$, for $\mathsf{Ind}_\mu^{x,A}$ we normally write $\mathsf{Ind}_{x,A}$.

Examples.

$$\mathsf{Ind}_{p,A}\colon A[p{:=}\mathsf{tt}] \to A[p{:=}\mathsf{ff}] \to \forall p^{\mathbf{B}} A,$$
$$\mathsf{Ind}_{n,A}\colon A[n{:=}0] \to (\forall n.A \to A[n{:=}\mathsf{S}n]) \to \forall n^{\mathbf{N}} A,$$
$$\mathsf{Ind}_{l,A}\colon A[l{:=}\mathsf{Nil}] \to (\forall x, l.A \to A[l{:=}\mathsf{cons}(x,l)]) \to \forall l^{\mathbf{L}(\alpha)} A$$
$$\mathsf{Ind}_{x,A}\colon \forall y_1 A[x{:=}\mathsf{Inl}(y_1)] \to \forall y_2 A[x{:=}\mathsf{Inr}(y_2)] \to \forall x^{\rho_1+\rho_2} A.$$

For the simultaneously defined algebras tree and tlist the induction axiom $\mathsf{Ind}_{\mathsf{tree}}^{b,bs,\hat{P}_1(b),\hat{P}_2(bs)}$ is

$$D_1 \to D_2 \to D_3 \to D_4 \to \forall b^{\mathsf{tree}} \hat{P}_1(b)$$

with

$$D_1 := \forall n \hat{P}_1(\mathsf{Leaf}(n)),$$
$$D_2 := \forall bs^{\mathsf{tlist}}.\hat{P}_2(bs) \to \hat{P}_1(\mathsf{Branch}(bs)),$$
$$D_3 := \hat{P}_2(\mathsf{Empty}),$$
$$D_4 := \forall b^{\mathsf{tree}}, bs^{\mathsf{tlist}}.\hat{P}_1(b) \to \hat{P}_2(bs) \to \hat{P}_2(\mathsf{Tcons}(b, bs)).$$

Remark 2.4.1. It may happen that in an induction on simultaneously defined algebras one only needs to induct on some of those algebras. Then we can simplify the induction formula accordingly, by

- omitting all step formulas D_i corresponding to constructor types with irrelevant value type τ_j, and
- simplifying the remaining step formulas by omitting from the induction hypotheses $\forall \vec{x}^{\vec{\sigma}_1}\, \hat{P}_{j_1}(y_{m+1}\vec{x})$, $\ldots$, $\forall \vec{x}^{\vec{\sigma}_n}\, \hat{P}_{j_n}(y_{m+n}\vec{x})$ all those corresponding to constructor types with irrelevant value type τ_{j_ν}.

In the $\mathsf{tree}, \mathsf{tlist}$-example, if we only want to induct on tlist, then the step formulas are

$$D_3 := \hat{P}_2(\mathsf{Empty}),$$
$$D_4 := \forall b^{\mathsf{tree}}, bs^{\mathsf{tlist}}.\hat{P}_2(bs) \to \hat{P}_2(\mathsf{Tcons}(b, bs)).$$

Hence the simplified induction axiom is

$$\mathsf{Ind}_{bs,\hat{P}_2(bs)}\colon D_3 \to D_4 \to \forall bs^{\mathsf{tlist}} \hat{P}_2(bs).$$

2.4.4. Cases. There is an important variant of the induction axiom, where no induction hypotheses are used, i.e. all step formulas have the form

$$D_i := \forall y_1^{\rho_1}, \dots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1 \to \mu_{j_1}}, \dots, y_{m+n}^{\vec{\sigma}_n \to \mu_{j_n}} . \hat{P}_j(\mathsf{constr}_i^{\vec{\mu}}(\vec{y})).$$

This variant is called the *cases axiom*; it distinguishes cases according to the outer constructor form. The formula of the cases axiom is

$$\mathsf{Cases}_{x_j, \hat{P}_j(x_j)} \colon D_{i_1}, \dots, D_{i_q} \to \forall x_j^{\mu_j} \hat{P}_j(x_j),$$

where $D_{i_1}, \dots, D_{i_q}$ consists of all D_i concerning constructors for μ_j.

Examples are

$$\mathsf{Cases}_{n,A} \colon A[n{:=}0] \to \forall n A[n{:=}\mathsf{S}n] \to \forall n^{\mathbf{N}} A,$$

$$\mathsf{Cases}_{l,A} \colon A[l{:=}\mathsf{Nil}] \to \forall x, l A[l{:=}\mathsf{cons}(x,l)] \to \forall l^{\mathbf{L}(\alpha)} A.$$

In the $\mathsf{tree}, \mathsf{tlist}$-example, if we want to distinguish cases on tlist, then the step formulas are

$$D_3 := \hat{P}(\mathsf{Empty}),$$

$$D_4 := \forall b^{\mathsf{tree}} \forall bs^{\mathsf{tlist}} \hat{P}(\mathsf{Tcons}(b, bs)).$$

Hence the cases axiom is

$$\mathsf{Cases}_{bs, \hat{P}(bs)} \colon \hat{P}(\mathsf{Empty}) \to \forall b^{\mathsf{tree}} \forall bs^{\mathsf{tlist}} \hat{P}(\mathsf{Tcons}(b, bs)) \to \forall bs^{\mathsf{tlist}} \hat{P}(bs).$$

2.5. Notes

Section 2.2 is based on an extension of Tait's method of strong computability predicates. The definition of these predicates and also the proof are related to Zucker's proof of strong normalization of his term system for recursion on the first three number or tree classes. However, Zucker uses a combinatory term system and defines strong computability for closed terms only. Following some ideas in an unpublished note of Berger, Benl (in his diploma thesis [**3**]) adapted this proof to terms in the simply typed λ-calculus, possibly involving free variables. Here this proof is extended to the case of simultaneously defined free algebras.

In a recent paper of Abel and Altenkirch [**1**], a similar result is proved with a different method, involving Aczel's notion of a set-based relation. It seems worthwile to verify that an appropriate variant of the standard Tait proof also yields this result. However, an additional merit of the method of Abel and Altenkirch is that they are also able to treat co-inductive types. We have not tried to extend our's in this direction as well.

CHAPTER 3

Unification and Proof Search

We describe a proof search method suitable for minimal logic with higher order functionals. It is based on Huet's [**22**] unification algorithm for the simply typed lambda calculus, which is treated first.

Huet's unification algorithm does not terminate in general; this cannot be avoided, since it is well known that higher order unification is undecidable. This non-termination can be avoided if we restrict ourselves to a certain fragment of higher order (simply typed) minimal logic. This fragment is determined by requiring that every higher order variable Y can only occur in a context $Y\vec{x}$, where $\vec{x}$ are distinct bound variables in the scope of the operator binding Y, and of opposite polarity. Note that for first order logic this restriction does not mean anything, since there are no higher order variables. However, when designing a proof search algorithm for first order logic only, one is naturally led into this fragment of higher order logic, where the algorithm works as well.

3.1. Huet's Unification Algorithm

We work in the simply typed λ-calculus, with the usual conventions. For instance, whenever we write a term we assume that it is correctly typed. *Substitutions* are denoted by φ, ψ, ρ. The result of applying a substitution φ to a term r or a formula A is written as $r\varphi$ or $A\varphi$, with the understanding that after the substitution all terms are brought into long normal form.

Q always denotes a $\forall\exists\forall$-prefix, say $\forall\vec{x}\exists\vec{y}\forall\vec{z}$, with distinct variables. We call $\vec{x}$ the *signature variables*, $\vec{y}$ the *flexible variables* and $\vec{z}$ the *forbidden variables* of Q, and write $Q_\exists$ for the existential part $\exists\vec{y}$ of Q.

A *Q-term* is a term with all its free variables in Q, and similarly a *Q-formula* is a formula with all its free variables in Q. A *Q-substitution* is a substitution of Q-terms.

A *unification problem* $\mathcal{U}$ consists of a $\forall\exists\forall$-prefix Q and a conjunction C of equations between Q-terms of the same type, i.e. $\bigwedge_{i=1}^n r_i = s_i$. We may assume that each such equation is of the form $\lambda\vec{x}r = \lambda\vec{x}s$ with the same $\vec{x}$ (which may be empty) and r, s of ground type.

A *solution* to such a unification problem $\mathcal{U}$ is a Q-substitution φ such that for every i, $r_i\varphi = s_i\varphi$ holds (i.e. $r_i\varphi$ and $s_i\varphi$ have the same normal form). We sometimes write C as $\vec{r} = \vec{s}$, and (for obvious reasons) call it a list of unification pairs.

We now define the unification algorithm. It takes a unification problem $\mathcal{U} = QC$ and produces a not necessarily well-founded tree (called *matching tree* by Huet [**22**]) with nodes labelled by unification problems and vertices labelled by substitutions.

Definition (Unification algorithm). We distinguish cases according to the form of the unification problem, and either give the transition done by the algorithm, or else state that it fails.

Case identity, i.e. $Q.r = r \wedge C$. Then

$$(Q.r = r \wedge C) \Longrightarrow_\varepsilon QC.$$

Case ξ, i.e. $Q.\lambda\vec{x}\, r = \lambda\vec{x}\, s \wedge C$. We may assume here that the bound variables $\vec{x}$ are the same on both sides.

$$(Q.\lambda\vec{x}\, r = \lambda\vec{x}\, s \wedge C) \Longrightarrow_\varepsilon Q\forall\vec{x}.r = s \wedge C.$$

Case rigid-rigid, i.e. $Q.f\vec{r} = g\vec{s} \wedge C$ with both f and g rigid, that is either a signature variable or else a forbidden variable. If f is different from g then fail. If f equals g,

$$(Q.f\vec{r} = f\vec{s} \wedge C) \Longrightarrow_\varepsilon Q.\vec{r} = \vec{s} \wedge C.$$

Case flex-rigid, i.e. $Q.u\vec{r} = f\vec{s} \wedge C$ with f rigid. Then the algorithm branches into one *imitation* branch and m *projection* branches, where $r = r_1, \ldots, r_m$. Imitation replaces the flexible head u, using the substitution $\rho = [u{:=}\lambda\vec{x}.f(h_1\vec{x})\ldots(h_n\vec{x})]$ with new variables $\vec{h}$ and $\vec{x}$. For r_i we have a projection if and only if the final value type of r_i is the (ground) type of $f\vec{s}$. Then the i-th projections pulls r_i in front, by $\rho = [u{:=}\lambda\vec{x}.x_i(h_1\vec{x})\ldots(h_{n_i}\vec{x})]$. In each of these branches we have

$$(Q.u\vec{r} = f\vec{s} \wedge C) \Longrightarrow_\rho Q'.(u\vec{r} = f\vec{s} \wedge C)\rho,$$

where Q' is obtained from Q by removing $\exists u$ and adding $\exists\vec{h}$.

Case flex-flex, i.e. $Q.u\vec{r} = v\vec{s} \wedge C$. If there is a first flex-rigid or rigid-flex equation in C, pull this equation (possibly swapped) to the front and apply case flex-rigid. Otherwise, i.e. if all equations are between terms with flexible heads, pick a new variable z of ground type and let ρ be the substitution mapping each of these flexible heads u to $\lambda\vec{x}z$.

$$(Q.u\vec{r} = v\vec{s} \wedge C) \Longrightarrow_\rho Q.\emptyset.$$

This concludes the definition of the unification algorithm.

Clearly ρ is defined on flexible variables of Q only, and its value terms have no free occurrences of forbidden variables from Q. Our next task is to prove correctness and completeness of this algorithm.

Theorem 3.1.1 (Huet). *Let a unification problem $\mathcal{U}$ consisting of a $\forall\exists\forall$-prefix Q and a list $\vec{r} = \vec{s}$ of unification pairs be given. Then either*

- *the unification algorithm can make a transition, and*
  - *(correctness) for every transition $\mathcal{U} \Longrightarrow_\rho \mathcal{U}'$ and $\mathcal{U}'$-solution φ' the substitution $(\rho \circ \varphi')\restriction Q_\exists$ is a $\mathcal{U}$-solution, and*
  - *(completeness) for every $\mathcal{U}$-solution φ there is a transition $\mathcal{U} \Longrightarrow_\rho \mathcal{U}'$ and $\mathcal{U}'$-solution φ' such that $\varphi = (\rho \circ \varphi')\restriction Q_\exists$, and moreover $\mu(\varphi') \leq \mu(\varphi)$ with $<$ in case flex-rigid, or else*
- *the unification algorithm fails, and there is no $\mathcal{U}$-solution, or else*
- *the unification algorithm succeeds, and $\vec{r} = \vec{s}$ is empty.*

Here $\mu(\varphi)$ denotes the number of applications in the value terms of φ.

PROOF. *Case* identity, i.e. $Q.r = r \wedge C \Longrightarrow_\varepsilon QC$. Then correctness and completeness are obvious.

Case ξ, i.e. $Q.\lambda\vec{x}\, r = \lambda\vec{x}\, s \wedge C \Longrightarrow_\varepsilon Q\forall\vec{x}.r = s \wedge C$. Again correctness and completeness are obvious.

Case rigid-rigid, i.e. $Q.f\vec{r} = g\vec{s} \wedge C \Longrightarrow_\varepsilon Q.\vec{r} = \vec{s} \wedge C$. If $f \neq g$, then the unification algorithm fails and there is no $\mathcal{U}$-solution. If $f = g$, again correctness and completeness are obvious.

Case flex-rigid, i.e. $\mathcal{U}$ is $Q.u\vec{r} = f\vec{s} \wedge C$.

Correctness. Assume $\mathcal{U} \Longrightarrow_\rho \mathcal{U}'$, which is to say $(Q.u\vec{r} = f\vec{s} \wedge C) \Longrightarrow_\rho Q'.(u\vec{r} = f\vec{s} \wedge C)\rho$. Let φ' be a $\mathcal{U}'$-solution, i.e. $(u\vec{r} = f\vec{s} \wedge C)\rho\varphi'$. Then clearly $(\rho \circ \varphi')\restriction Q_\exists$ is a $\mathcal{U}$-solution.

Completeness. Assume φ is a $\mathcal{U}$-solution, i.e. $(u\vec{r} = f\vec{s} \wedge C)\varphi$. We have to find a transition $\mathcal{U} \Longrightarrow_\rho \mathcal{U}'$ and a $\mathcal{U}'$-solution φ' such that $\varphi = (\rho\circ\varphi')\restriction Q_\exists$, and moreover $\mu(\varphi') < \mu(\varphi)$. Now $u\varphi$ must be of the form $\lambda\vec{x}.a\vec{t}$ with a either f or x_i.

Subcase a is f. Then take the imitation branch, i.e. replace the flexible head u using the substitution $\rho = [u{:=}\lambda\vec{x}.f(h_1\vec{x})\ldots(h_n\vec{x})]$ with new variables $\vec{h}$ and $\vec{x}$. Recall $(Q.u\vec{r} = f\vec{s} \wedge C) \Longrightarrow_\rho Q'.(u\vec{r} = f\vec{s} \wedge C)\rho$, where Q' is obtained from Q by removing $\exists u$ and adding $\exists\vec{h}$. Define φ' on the new variables $\vec{h}$ by $h_j\varphi' := \lambda\vec{x}t_j$, and as φ on all other variables. Then $\varphi = (\rho \circ \varphi')\restriction Q_\exists$ because of

$$u\rho\varphi' = \lambda\vec{x}.f\big((\vec{h}\varphi')\vec{x}\big) = \lambda\vec{x}.f\vec{t} = u\varphi,$$

and our assumption says that

$$f\vec{t}[\vec{x}{:=}\vec{r}\varphi] = f(\vec{s}\varphi)$$

Now φ' is a $\mathcal{U}'$-solution because of

$$(u\vec{r})\rho\varphi' = f\big(\vec{t}[\vec{x}{:=}\vec{r}\rho\varphi']\big) = f\big(\vec{t}[\vec{x}{:=}\vec{r}\varphi]\big) = f(\vec{s}\varphi) = f(\vec{s}\rho\varphi').$$

Subcase a is x_i. Then take the i-th projections branch, i.e. replace the flexible head u using the substitution $\rho = [u{:=}\lambda\vec{x}.x_i(h_1\vec{x})\ldots(h_{n_i}\vec{x})]$ with new variables $\vec{h}$ and $\vec{x}$. Recall $(Q.u\vec{r} = f\vec{s} \wedge C) \Longrightarrow_\rho Q'.(u\vec{r} = f\vec{s} \wedge C)\rho$, where Q' is obtained from Q by removing $\exists u$ and adding $\exists\vec{h}$. Define φ' on the new variables $\vec{h}$ by $h_j\varphi' := \lambda\vec{x}t_j$, and as φ on all other variables. Then $\varphi = (\rho \circ \varphi')\restriction Q_\exists$ because of

$$u\rho\varphi' = \lambda\vec{x}.x_i\big((\vec{h}\varphi')\vec{x}\big) = \lambda\vec{x}.x_i\vec{t} = u\varphi,$$

and our assumption says that

$$(r_i\varphi)\vec{t}[\vec{x}{:=}\vec{r}\varphi] = f(\vec{s}\varphi)$$

Now φ' is a $\mathcal{U}'$-solution because of

$$(u\vec{r})\rho\varphi' = (r_i\rho\varphi')\big(\vec{t}[\vec{x}{:=}\vec{r}\rho\varphi']\big) = (r_i\varphi)\big(\vec{t}[\vec{x}{:=}\vec{r}\varphi]\big) = f(\vec{s}\varphi) = f(\vec{s}\rho\varphi').$$

Case flex-flex, i.e. $Q.C$ where all equations in C are between terms with flexible heads. Then for a new variable z of ground type we have taken ρ to be the substitution mapping each of these flexible heads to $\lambda\vec{x}z$, and

$$(QC) \Longrightarrow_\rho Q.\emptyset.$$

Correctness. $(\rho \circ \varphi')\restriction Q_\exists$ clearly is a $\mathcal{U}$-solution for every φ'.

Completeness. To simplify the notation let $C = (ur = vs)$. Assume that φ is a $\mathcal{U}$-solution, i.e. $(ur = vs)\varphi$. Then $u\varphi = \lambda x.f_1t_1$ and $v\varphi = \lambda x.f_2t_2$, and by assumption $f_1t_1[x{:=}r] = f_2t_2[x{:=}s]$. Define φ' to be φ with the assignments to u, v removed and $z \mapsto f_1t_1[x{:=}r]$ $(= f_2t_2[x{:=}s])$ added. Then clearly $(\rho \circ \varphi')\restriction Q_\exists = \varphi$, and $\mu(\varphi') \leq \mu(\varphi)$. □

COROLLARY 3.1.2. *Given a unification problem $\mathcal{U} = QC$, and a success node in the matching tree, labelled with a prefix Q' (i.e. a unification problem $\mathcal{U}'$ with no unification pairs). Then by composing the substitution labels on the branch leading to this node we obtain a pair (Q', ρ) with a "transition" substitution ρ and such that for any Q'-substitution φ', $(\rho \circ \varphi')\restriction Q_\exists$ is an $\mathcal{U}$-solution. Moreover, every $\mathcal{U}$-solution can be obtained in this way, for an appropriate success node. Since the empty substitution is a Q'-substitution, $\rho\restriction Q_\exists$ is a $\mathcal{U}$-solution, which is most general in the sense stated.* □

3.2. The Pattern Unification Algorithm

We modify restrict the notion of Q-term as follows. *Q-terms* are inductively defined by the following clauses.

- If u is a universally quantified variable in Q or a constant, and $\vec{r}$ are Q-terms, then $u\vec{r}$ is a Q-term.
- For any flexible variable y and distinct forbidden variables $\vec{z}$ from Q, $y\vec{z}$ is a Q-term.
- If r is a $Q\forall z$-term, then λzr is a Q-term.

Explicitely, r is a Q-term iff all its free variables are in Q, and for every subterm $y\vec{r}$ of r with y free in r and flexible in Q, the $\vec{r}$ are distinct variables either λ-bound in r (such that $y\vec{r}$ is in the scope of this λ) or else forbidden in Q.

Q-goals and *Q-clauses* are simultaneously defined by

- If $\vec{r}$ are Q-terms, then $P\vec{r}$ is a Q-goal as well as a Q-clause.
- If D is a Q-clause and G is a Q-goal, then $D \to G$ is a Q-goal.
- If G is a Q-goal and D is a Q-clause, then $G \to D$ is a Q-clause.
- If G is a $Q\forall x$-goal, then $\forall xG$ is a Q-goal.
- If $D[y{:=}Y\vec{z}]$ is a $\forall\vec{x}\exists\vec{y}, Y\forall\vec{z}$-clause, then $\forall yD$ is a $\forall\vec{x}\exists\vec{y}\forall\vec{z}$-clause.

Explicitely, a formula A is a *Q-goal* iff all its free variables are in Q, and for every subterm $y\vec{r}$ of A with y either existentially bound in A (with $y\vec{r}$ in the scope) or else free in A and flexible in Q, the $\vec{r}$ are distinct variables either λ- or universally bound in A (such that $y\vec{r}$ is in the scope) or else free in A and forbidden in Q.

A *Q-substitution* is a substitution of Q-terms.

A *pattern unification problem* $\mathcal{U}$ consists of a $\forall\exists\forall$-prefix Q and a conjunction C of equations between Q-terms of the same type, i.e. $\bigwedge_{i=1}^{n} r_i = s_i$. We may assume that each such equation is of the form $\lambda\vec{x}r = \lambda\vec{x}s$ with the same $\vec{x}$ (which may be empty) and r, s of ground type.

A *solution* to such a unification problem $\mathcal{U}$ is a Q-substitution φ such that for every i, $r_i\varphi = s_i\varphi$ holds (i.e. $r_i\varphi$ and $s_i\varphi$ have the same normal form). We sometimes write C as $\vec{r} = \vec{s}$, and (for obvious reasons) call it a list of unification pairs.

We now define the pattern unification algorithm. It takes a unification problem $\mathcal{U} = QC$ and returns a substitution ρ and another unification problem $\mathcal{U}' = Q'C'$. Note that ρ will be neither a Q-substitution nor a Q'-substitution, but will have the property that

- ρ is defined on flexible variables of Q only, and its value terms have no free occurrences of forbidden variables from Q,
- if G is a Q-goal, then $G\rho$ is a Q'-goal, and
- whenever φ' is a $\mathcal{U}'$-solution, then $(\rho \circ \varphi')\restriction_{Q_\exists}$ is a $\mathcal{U}$-solution.

Definition (Pattern Unification Algorithm). We distinguish cases according to the form of the unification problem, and either give the transition done by the algorithm, or else state that it fails.

Case identity, i.e. $Q.r = r \wedge C$. Then

$$(Q.r = r \wedge C) \Longrightarrow_\varepsilon QC.$$

Case ξ, i.e. $Q.\lambda\vec{x}\, r = \lambda\vec{x}\, s \wedge C$. We may assume here that the bound variables $\vec{x}$ are the same on both sides.

$$(Q.\lambda\vec{x}\, r = \lambda\vec{x}\, s \wedge C) \Longrightarrow_\varepsilon Q\forall\vec{x}.r = s \wedge C.$$

Case rigid-rigid, i.e. $Q.f\vec{r} = f\vec{s} \wedge C$ with f either a signature variable or else a forbidden variable.

$$(Q.f\vec{r} = f\vec{s} \wedge C) \Longrightarrow_\varepsilon Q.\vec{r} = \vec{s} \wedge C.$$

Case flex-flex with equal heads, i.e. $Q.u\vec{y} = u\vec{z} \wedge C$.

$$(Q.u\vec{y} = u\vec{z} \wedge C) \Longrightarrow_\rho Q'.C\rho$$

with $\rho = [u{:=}\lambda\vec{y}.u'\vec{w}]$, Q' is Q with $\exists u$ replaced by $\exists u'$, and $\vec{w}$ an enumeration of those y_i which are identical to z_i (i.e. the variable at the same position in $\vec{z}$). Notice that $\lambda\vec{y}.u'\vec{w} = \lambda\vec{z}.u'\vec{w}$.

Case flex-flex with different heads, i.e. $Q.u\vec{y} = v\vec{z} \wedge C$.

$$(Q.u\vec{y} = v\vec{z} \wedge C) \Longrightarrow_\rho Q'C\rho,$$

where ρ and Q' are defined as follows. Let $\vec{w}$ be an enumeration of the variables both in $\vec{y}$ and in $\vec{z}$. Then $\rho = [u, v{:=}\lambda\vec{y}.u'\vec{w}, \lambda\vec{z}.u'\vec{w}]$, and Q' is Q with $\exists u, \exists v$ removed and $\exists u'$ inserted.

Case flex-rigid, i.e. $Q.u\vec{y} = t \wedge C$ with t rigid, i.e. not of the form $v\vec{z}$ with flexible v.

Subcase occurrence check: t contains (a critical subterm with head) u. Fail.

Subcase pruning: t contains a subterm $v\vec{w}_1 z\vec{w}_2$ with $\exists v$ in Q, and z free in t but not in $\vec{y}$.

$$(Q.u\vec{y} = t \wedge C) \Longrightarrow_\rho Q'.u\vec{y} = t\rho \wedge C\rho$$

where $\rho = [v{:=}\lambda\vec{w}_1\lambda z\lambda\vec{w}_2.v'\vec{w}_1\vec{w}_2]$, Q' is Q with $\exists v$ replaced by $\exists v'$.

Subcase pruning impossible: $\lambda\vec{y}t$ (after all pruning steps are done still) has a free occurrence of a forbidden variable z. Fail.

Subcase explicit definition: otherwise.

$$(Q.u\vec{y} = t \wedge C) \Longrightarrow_\rho Q'C\rho$$

where $\rho = [u{:=}\lambda\vec{y}t]$, and Q' is obtained from Q by removing $\exists u$. This concludes the definition of the pattern unification algorithm.

Our next task is to prove that this algorithm indeed has the three properties stated above. The first one (ρ is defined on flexible variables of Q only, and its value terms have no free occurrences of forbidden variables from Q) is obvious from the definition. We now prove the second one; the third one will be proved next.

LEMMA 3.2.1. *If $Q \Longrightarrow_\rho Q'$ and G is a Q-goal, then $G\rho$ is a Q'-goal.*

PROOF. We distinguish cases according to the definition of the unification algorithm. All cases are straightforward:

Cases identity, ξ and rigid-rigid. Then $\rho = \varepsilon$ and the claim is trivial.

Case flex-flex with equal heads. Then $\rho = [u{:=}\lambda\vec{y}.u'\vec{w}]$ with $\vec{w}$ a sublist of $\vec{y}$, and Q' is Q with $\exists u$ replaced by $\exists u'$. Then clearly $G[u{:=}\lambda\vec{y}.u'\vec{w}]$ is a Q'-goal (recall that after a substitution we always normalize).

Case flex-flex with different heads. Then $\rho = [u, v{:=}\lambda\vec{y}.u'\vec{w}, \lambda\vec{z}.u'\vec{w}]$ with $\vec{w}$ an enumeration of the variables both in $\vec{y}$ and in $\vec{z}$, and Q' is Q with $\exists u, \exists v$ removed and $\exists u'$ inserted. Again clearly $G[u, v{:=}\lambda\vec{y}.u'\vec{w}, \lambda\vec{z}.u'\vec{w}]$ is a Q'-goal.

Case flex-rigid, *Subcase* pruning: Then $\rho = [v{:=}\lambda\vec{w}_1, z, \vec{w}_2.v'\vec{w}_1\vec{w}_2]$, and Q' is Q with $\exists v$ replaced by $\exists v'$. Suppose G is a Q-goal. Then clearly $G[v{:=}\lambda\vec{w}_1, z, \vec{w}_2.v'\vec{w}_1\vec{w}_2]$ is a Q'-goal.

Case flex-rigid, *Subcase* explicit definition: Then $\rho = [u{:=}\lambda\vec{y}t]$ with a Q-term $\lambda\vec{y}t$ without free occurrences of forbidden variables, and Q' is obtained from Q by removing $\exists u$. Suppose G is a Q-goal. Then clearly $G[u{:=}\lambda\vec{y}t]$ form) is a Q'-goal. □

Let $Q \longrightarrow_\rho Q'$ mean that for some C, C' we have $QC \Longrightarrow_\rho Q'C'$. Write $Q \longrightarrow_\rho^* Q'$ if there are $\rho_1, \ldots, \rho_n$ and $Q_1, \ldots, Q_{n-1}$ such that

$$Q \longrightarrow_{\rho_1} Q_1 \longrightarrow_{\rho_2} \cdots \longrightarrow_{\rho_{n-1}} Q_{n-1} \longrightarrow_{\rho_n} Q',$$

and $\rho = \rho_1 \circ \cdots \circ \rho_n$.

COROLLARY 3.2.2. *If $Q \longrightarrow_\rho^* Q'$ and G is a Q-goal, then $G\rho$ is a Q'-goal.*
□

LEMMA 3.2.3. *Let a unification problem $\mathcal{U}$ consisting of a $\forall\exists\forall$-prefix Q and a list $\vec{r} = \vec{s}$ of unification pairs be given. Then either*

- *the unification algorithm makes a transition $\mathcal{U} \Longrightarrow_\rho \mathcal{U}'$, and*

$$\Phi' : \mathcal{U}'\text{-solutions} \to \mathcal{U}\text{-solutions}$$
$$\varphi' \mapsto (\rho \circ \varphi')\restriction Q_\exists$$

  is well-defined and we have $\Phi : \mathcal{U}$-solutions $\to \mathcal{U}'$-solutions such that Φ' is inverse to Φ, i.e. $\Phi'(\Phi\varphi) = \varphi$, or else
- *the unification algorithm fails, and there is no $\mathcal{U}$-solution.*

PROOF. *Case* identity, i.e. $Q.r = r \wedge C \Longrightarrow_\varepsilon QC$. Let Φ be the identity.

Case ξ, i.e. $Q.\lambda\vec{x}\,r = \lambda\vec{x}\,s \wedge C \Longrightarrow_\varepsilon Q\forall\vec{x}.r = s \wedge C$. Let again Φ be the identity.

Case rigid-rigid, i.e. $Q.f\vec{r} = f\vec{s} \wedge C \Longrightarrow_\varepsilon Q.\vec{r} = \vec{s} \wedge C$. Let again Φ be the identity.

Case flex-flex with equal heads, i.e. $Q.u\vec{y} = u\vec{z} \wedge C \Longrightarrow_\rho Q'.C\rho$ with $\rho = [u{:=}\lambda\vec{y}.u'\vec{w}]$, Q' is Q with $\exists u$ replaced by $\exists u'$, and $\vec{w}$ an enumeration of

those y_i which are identical to z_i (i.e. the variable at the same position in $\vec{z}$). Notice that $\lambda\vec{y}.u'\vec{w} = \lambda\vec{z}.u'\vec{w}$.

1. Φ' is well-defined: Let φ' be a $\mathcal{U}'$-solution, i.e. assume that $C\rho\varphi'$ holds. We must show that $\varphi := (\rho \circ \varphi')\restriction Q_\exists$ is a $\mathcal{U}$-solution.

For $u\vec{y} = u\vec{z}$: We need to show $(u\varphi)\vec{y} = (u\varphi)\vec{z}$. But $u\varphi = u\rho\varphi' = (\lambda\vec{y}.u'\vec{w})\varphi'$. Hence $(u\varphi)\vec{y} = (u\varphi)\vec{z}$ by the construction of $\vec{w}$.

For $(r = s) \in C$: We need to show $(r = s)\varphi$. But by assumption $(r = s)\rho\varphi'$ holds, and $r = s$ has all its flexible variables from $Q_\exists$.

2. Definition of $\Phi\colon \mathcal{U}$-solutions $\to \mathcal{U}'$-solutions. Let a Q-substitution φ be given such that $(u\vec{y} = u\vec{z})\varphi$ and $C\varphi$. Define $u'(\Phi\varphi) := \lambda\vec{w}.(u\varphi)\vec{w}\vec{0}$ (w.l.o.g), and $v(\Phi\varphi) := v$ for every other variable v in $Q_\exists$.

$\Phi\varphi =: \varphi'$ is a $\mathcal{U}'$-solution: Let $(r = s) \in C$. Then $(r = s)\varphi$ by assumption, for φ is a Q-substitution such that $C\varphi$ holds. We must show

$$(r = s)\rho\varphi'.$$

Notice that our assumption $(u\varphi)\vec{y} = (u\varphi)\vec{z}$ implies that the normal form of both sides can only contain the variables in $\vec{w}$. Therefore

$$\begin{aligned} u\rho\varphi' &= (\lambda\vec{y}.u'\vec{w})\varphi' \\ &= \lambda\vec{y}.(\lambda\vec{w}.(u\varphi)\vec{w}\vec{0})\vec{w} \\ &= \lambda\vec{y}.(u\varphi)\vec{w}\vec{0} \\ &= \lambda\vec{y}.(u\varphi)\vec{y} \\ &= u\varphi \end{aligned}$$

and hence $(r = s)\rho\varphi'$.

3. $\Phi'(\Phi\varphi) = \varphi$: So let φ be an $\mathcal{U}$-solution, and $\varphi' := \Phi\varphi$. Then

$$\begin{aligned} u\big(\Phi'\varphi'\big) &= u\big((\rho \circ \varphi')\restriction Q_\exists\big) \\ &= u\rho\varphi' \\ &= u\varphi, \qquad\qquad \text{as proved in 2.} \end{aligned}$$

For every other variable v in $Q_\exists$ we obtain

$$\begin{aligned} v\big(\Phi'\varphi'\big) &= v\big((\rho \circ \varphi')\restriction Q_\exists\big) \\ &= v\rho\varphi' \\ &= v\varphi' \\ &= v\varphi. \end{aligned}$$

Case flex-flex with different heads, i.e. $\mathcal{U}$ is $Q.u\vec{y} = v\vec{z} \wedge C$. Let $\vec{w}$ be an enumeration of the variables both in $\vec{y}$ and in $\vec{z}$. Then $\rho = [u, v{:=}\lambda\vec{y}.u'\vec{w}, \lambda\vec{z}.u'\vec{w}]$, Q' is Q with $\exists u, \exists v$ removed and $\exists u'$ inserted, and $\mathcal{U}' = Q'C\rho$.

1. Φ' is well-defined: Let φ' be a $\mathcal{U}'$-solution, i.e. assume that $C\rho\varphi'$ holds. We must show that $\varphi := (\rho \circ \varphi')\restriction Q_\exists$ is a $\mathcal{U}$-solution.

For $u\vec{y} = v\vec{z}$: We need to show $(u\varphi)\vec{y} = (v\varphi)\vec{z}$. But $(u\varphi)\vec{y} = (u\rho\varphi')\vec{y} = (\lambda\vec{y}.(u'\varphi')\vec{w})\vec{y} = (u'\varphi')\vec{w}$, and similarly $(v\varphi)\vec{z} = (u'\varphi')\vec{w}$.

For $(r = s) \in C$: We need to show $(r = s)\varphi$. But since u' is a new variable, φ and $\rho \circ \varphi'$ coincide on all variables free in $r = s$, and we have $(r = s)\rho\varphi'$ by assumption.

2. Definition of $\Phi\colon \mathcal{U}$-solutions $\to \mathcal{U}'$-solutions. Let a Q-substitution φ be given such that $(u\vec{y} = v\vec{z})\varphi$ and $C\varphi$. Define

$$\begin{aligned} u'(\Phi\varphi) &:= \lambda\vec{w}.(u\varphi)\vec{w}\vec{0} \quad \text{w.l.o.g.; } \vec{0} \text{ arbitrary} \\ v'(\Phi\varphi) &:= \lambda\vec{w}.(v\varphi)\vec{0}\vec{w} \\ w(\Phi\varphi) &:= w\varphi \qquad\qquad \text{otherwise, i.e. } w \neq u', v', u \text{ flexible.} \end{aligned}$$

Since by assumption $(u\varphi)\vec{y} = (v\varphi)\vec{z}$, the normal forms of both $(u\varphi)\vec{y}$ and $(v\varphi)\vec{z}$ can only contain the common variables $\vec{w}$ from $\vec{y}, \vec{z}$ free. Hence, for $\varphi' := \Phi\varphi$, $u\rho\varphi' = u\varphi$ by the argument in the previous case, and similarly $v\rho\varphi' = v\varphi$. Since $r\varphi = s\varphi$ ($(r = s) \in C$ arbitrary) by assumption, and ρ only affects u and v, we obtain $r\rho\varphi' = s\rho\varphi'$, as required. $\Phi'(\Phi\varphi) = \varphi$ can now be proved as in the previous case.

Case flex-rigid, $\mathcal{U}$ is $Q.u\vec{y} = t \wedge C$.

Subcase occurrence check: t contains (a critical subterm with head) u. Then clearly there is no Q-substitution φ such that $(u\varphi)\vec{y} = t\varphi$.

Subcase pruning: Here t contains a subterm $v\vec{w}_1 z\vec{w}_2$ with $\exists v$ in Q, and z free in t. Then $\rho = [v{:=}\lambda\vec{w}_1, z, \vec{w}_2.v'\vec{w}_1\vec{w}_2]$, Q' is Q with $\exists v$ replaced by $\exists v'$, and $\mathcal{U}' = Q'.u\vec{y} = t\rho \wedge C\rho$.

1. Φ' is well-defined: Let φ' be a $\mathcal{U}'$-solution, i.e. $(u\varphi')\vec{y} = t\rho\varphi'$, and $r\rho\varphi' = s\rho\varphi'$ for $(r = s) \in C$. We must show that $\varphi := (\rho \circ \varphi')\restriction Q_\exists$ is a $\mathcal{U}$-solution.

For $u\vec{y} = t$: We need to show $(u\varphi)\vec{y} = t\rho\varphi'$. But

$$\begin{aligned} (u\varphi)\vec{y} &= (u\rho\varphi')\vec{y} \\ &= (u\varphi')\vec{y} \qquad \text{since } \rho \text{ does not touch } u \\ &= t\rho\varphi' \qquad\quad \text{by assumption.} \end{aligned}$$

For $(r = s) \in C$: We need to show $(r = s)\varphi$. But since v' is a new variable, $\varphi = (\rho \circ \varphi')\restriction Q_\exists$ and $\rho \circ \varphi'$ coincide on all variables free in $r = s$, and the claim follows from $(r = s)\rho\varphi'$.

2. Definition of $\Phi\colon \mathcal{U}$-solutions $\to \mathcal{U}'$-solutions. For a $\mathcal{U}$-solution φ define

$$\begin{aligned} v'(\Phi\varphi) &:= \lambda\vec{w}_1, \vec{w}_2.(v\varphi)\vec{w}_1 0\vec{w}_2 \\ w(\Phi\varphi) &:= w\varphi \qquad\qquad \text{otherwise, i.e. } w \neq v', v \text{ flexible.} \end{aligned}$$

Since by assumption $(u\varphi)\vec{y} = t\varphi$, the normal form of $t\varphi$ cannot contain z free. Therefore, for $\varphi' := \Phi\varphi$,

$$\begin{aligned} v\rho\varphi' &= (\lambda\vec{w}_1, z, \vec{w}_2.v'\vec{w}_1\vec{w}_2)\varphi' \\ &= \lambda\vec{w}_1, z, \vec{w}_2.(\lambda\vec{w}_1, \vec{w}_2.(v\varphi)\vec{w}_1 0\vec{w}_2)\vec{w}_1\vec{w}_2 \\ &= \lambda\vec{w}_1, z, \vec{w}_2.(v\varphi)\vec{w}_1 0\vec{w}_2 \\ &= \lambda\vec{w}_1, z, \vec{w}_2.(v\varphi)\vec{w}_1 z\vec{w}_2 \\ &= v\varphi. \end{aligned}$$

Hence $\varphi' = \Phi\varphi$ satisfies $(u\varphi')\vec{y} = t\rho\varphi'$. For $r = s$ this follows by the same argument. $\Phi'(\Phi\varphi) = \varphi$ can again be proved as in the previous case.

Subcase pruning impossible: Then $\lambda\vec{y}t$ has an occurrence of a universally quantified (i.e. forbidden) variable z. Therefore clearly there is no Q-substitution φ such that $(u\varphi)\vec{y} = t\varphi$.

Subcase explicit definition. Then $\rho = [u{:=}\lambda\vec{y}t]$, Q' is obtained from Q by removing $\exists u$, and $\mathcal{U}' = Q'C\rho$. Note that ρ is a Q'-substitution, for we have performed the pruning steps.

1. Φ' is well-defined: Let φ' be a $\mathcal{U}'$-solution, i.e. $r\rho\varphi' = s\rho\varphi'$ for $(r = s) \in C$. We must show that $\varphi := (\rho \circ \varphi')\restriction Q_\exists$ is an $\mathcal{U}$-solution.

For $u\vec{y} = t$: We need to show $(u\rho\varphi')\vec{y} = t\rho\varphi'$. But

$$\begin{aligned}(u\rho\varphi')\vec{y} &= ((\lambda\vec{y}t)\varphi')\vec{y}\\ &= t\varphi'\\ &= t\rho\varphi' \qquad \text{since } u \text{ does not appear in } t.\end{aligned}$$

For $(r = s) \in C$: We need to show $(r = s)\varphi$. But this clearly follows from $(r = s)\rho\varphi'$.

2. Definition of $\Phi\colon \mathcal{U}$-solutions $\to \mathcal{U}'$-solutions, and proof of $\Phi'(\Phi\varphi) = \varphi$. For a $\mathcal{U}$-solution φ define $\Phi\varphi = \varphi\restriction Q_\exists$. Then

$$u\rho\varphi' = \lambda\vec{y}t\varphi' = \lambda\vec{y}t\varphi = u\varphi,$$

and clearly $v\rho\varphi' = v\varphi$ for all other flexible φ. For $(r = s) \in C$, from $r\varphi = s\varphi$ we easily obtain $r\varphi' = s\varphi'$. □

It is not hard to see that the unification algorithm terminates, by defining a measure that decreases with each transition.

Corollary 3.2.4. *Given a unification problem $\mathcal{U} = QC$, the unification algorithm either fails, and there is no $\mathcal{U}$-solution, or else returns a pair (Q', ρ) with a "transition" substitution ρ and a prefix Q' (i.e. a unification problem $\mathcal{U}'$ with no unification pairs) such that for any Q'-substitution φ', $(\rho \circ \varphi')\restriction Q_\exists$ is an $\mathcal{U}$-solution, and every $\mathcal{U}$-solution can be obtained in this way. Since the empty substitution is a Q'-substitution, $\rho\restriction Q_\exists$ is a $\mathcal{U}$-solution, which is most general in the sense stated.* □

3.3. Proof Search

A *Q-sequent* has the form $\mathcal{P} \Rightarrow G$, where $\mathcal{P}$ is a list of Q-clauses and G is a Q-goal.

We write $M[\mathcal{P}]$ to indicate that all assumption variables in the derivation M are assumptions of clauses in $\mathcal{P}$.

Write $\vdash^n S$ for a set S of sequents if there are derivations $M_i^{G_i}[\mathcal{P}_i]$ in long normal form for all $(\mathcal{P}_i \Rightarrow G_i) \in S$ such that $\sum \mathsf{dp}(M_i) \le n$. Let $\vdash^{<n} S$ mean $\exists m{<}n\ \vdash^m S$.

We now prove correctness and completeness of the proof search procedure: correctness is the if-part of the two lemmata to follow, and completeness the only-if-part.

Lemma 3.3.1. *Let Q be a $\forall\exists\forall$-prefix, $\{\mathcal{P} \Rightarrow \forall\vec{x}.\vec{D} \to A\} \cup S$ Q-sequents with $\vec{x}, \vec{D}$ not both empty. Then we have for every substitution φ:*

$$\varphi \text{ is a } Q\text{-substitution such that } \vdash^n \big(\{\mathcal{P} \Rightarrow \forall\vec{x}.\vec{D} \to A\} \cup S\big)\varphi$$

if and only if

$$\varphi \text{ is a } Q\forall\vec{x}\text{-substitution such that } \vdash^{<n} \big(\{\mathcal{P} \cup \vec{D} \Rightarrow A\} \cup S\big)\varphi.$$

PROOF. *"If"*. Let φ be a $Q\forall\vec{x}$-substitution and $\vdash^{<n} \big(\{\mathcal{P} \cup \vec{D} \Rightarrow A\} \cup S\big)\varphi$. So we have

$$N^{A\varphi}[\vec{D}\varphi \cup \mathcal{P}\varphi].$$

Since φ is a $Q\forall\vec{x}$-substitution, no variable in $\vec{x}$ can be free in $\mathcal{P}\varphi$, or free in $y\varphi$ for some $y \in \mathsf{dom}(\varphi)$. Hence

$$M^{(\forall\vec{x}.\vec{D}\to A)\varphi}[\mathcal{P}\varphi] := \lambda\vec{x}\lambda\vec{u}^{\vec{D}\varphi}N$$

is a correct derivation.

"Only if". Let φ be a Q-substitution and $\vdash^n \big(\{\mathcal{P} \Rightarrow \forall\vec{x}.\vec{D} \to A\} \cup S\big)\varphi$. This means we have a derivation (in long normal form)

$$M^{(\forall\vec{x}.\vec{D}\to A)\varphi}[\mathcal{P}\varphi] = \lambda\vec{x}\lambda\vec{u}^{\vec{D}\varphi}.N^{A\varphi}[\vec{D}\varphi \cup \mathcal{P}\varphi].$$

Now $\mathsf{dp}(N) < \mathsf{dp}(M)$, hence $\vdash^{<n} \big(\{\mathcal{P} \cup \vec{D} \Rightarrow A\} \cup S\big)\varphi$, and φ clearly is a $Q\forall\vec{x}$-substitution. □

LEMMA 3.3.2. *Let Q be a $\forall\exists\forall$-prefix, $\{\mathcal{P} \Rightarrow P\vec{r}\} \cup S$ Q-sequents and φ a substitution. Then*

$$\varphi \textit{ is a } Q\textit{-substitution such that } \vdash^n \big(\{\mathcal{P} \Rightarrow P\vec{r}\} \cup S\big)\varphi$$

if and only if there is a clause $\forall\vec{x}.\vec{G} \to P\vec{s}$ in $\mathcal{P}$ such that the following holds. Let $\vec{z}$ be the final universal variables in Q, $\vec{X}$ be new ("raised") variables such that $X_i\vec{z}$ has the same type as x_i, let Q^* be Q with the existential variables extended by $\vec{X}$, and let $*$ indicate the substitution $[x_1, \ldots, x_n{:=}X_1\vec{z}, \ldots, X_n\vec{z}]$. Then there is a result (Q', ρ) of either Huet's or the pattern unification algorithm applied to $Q^*.\vec{r} = \vec{s}^*$ and a Q'-substitution φ' such that $\vdash^{<n} \big(\{\mathcal{P} \Rightarrow \vec{G}^*\} \cup S\big)\rho\varphi'$, and $\varphi = (\rho \circ \varphi')\restriction Q_\exists$.

PROOF. *"If"*. Let (Q', ρ) be such a result, and assume that φ' is a Q'-substitution such that $N_i \vdash \big(\mathcal{P} \Rightarrow \vec{G}^*\big)\rho\varphi'$. Let $\varphi := (\rho \circ \varphi')\restriction Q_\exists$. From $\mathsf{unif}(Q^*, \vec{r} = \vec{s}^*) = (Q', \rho)$ we know $\vec{r}\rho = \vec{s}^*\rho$, hence $\vec{r}\varphi = \vec{s}^*\rho\varphi'$. Then

$$u^{(\forall\vec{x}.\vec{G}\to P\vec{s})\varphi}\big((\vec{X}\rho\varphi')\vec{z}\big)\vec{N}^{\vec{G}^*\rho\varphi'}$$

derives $P\vec{s}^*\rho\varphi'$ (i.e. $P\vec{r}\varphi$) from $\mathcal{P}\varphi$.

"Only if". Assume φ is a Q-substitution such that $\vdash (\mathcal{P} \Rightarrow P\vec{r})\varphi$, say by $u^{(\forall\vec{x}.\vec{G}\to P\vec{s})\varphi}\vec{t}\vec{N}^{(\vec{G}\varphi)[\vec{x}:=\vec{t}]}$, with $\forall\vec{x}.\vec{G} \to P\vec{s}$ a clause in $\mathcal{P}$, and with additional assumptions from $\mathcal{P}\varphi$ in $\vec{N}$. Then $\vec{r}\varphi = (\vec{s}\varphi)[\vec{x}:=\vec{t}]$. Since we can assume that the variables $\vec{x}$ are new and in particular not range variables of φ, with

$$\vartheta := \varphi \cup [\vec{x}:=\vec{t}]$$

we have $\vec{r}\varphi = \vec{s}\vartheta$. Let $\vec{z}$ be the final universal variables in Q, $\vec{X}$ be new ("raised") variables such that $X_i\vec{z}$ has the same type as x_i, let Q^* be Q with the existential variables extended by $\vec{X}$, and for terms and formulas let $*$ indicate the substitution $[x_1, \ldots, x_n{:=}X_1\vec{z}, \ldots, X_n\vec{z}]$. Moreover, let

$$\vartheta^* := \varphi \cup [X_1, \ldots, X_n{:=}\lambda\vec{z}.t_1, \ldots, \lambda\vec{z}.t_n].$$

Then $\vec{r}\vartheta^* = \vec{r}\varphi = \vec{s}\vartheta = \vec{s}^*\vartheta^*$, i.e. ϑ^* is a solution to the unification problem given by Q^* and $\vec{r} = \vec{s}$. Hence by Corollary 3.1.2 $\mathsf{unif}(Q^*, \vec{r} = \vec{s}^*) = (Q', \rho)$ and there is a Q'-substitution φ' such that $\vartheta^* = (\rho \circ \varphi')\restriction Q^*_\exists$, hence $\varphi = (\rho \circ \varphi')\restriction Q_\exists$. Also, $(\vec{G}\varphi)[\vec{x}:=\vec{t}] = \vec{G}\vartheta = \vec{G}^*\vartheta^* = \vec{G}^*\rho\varphi'$. □

A *state* is a pair (Q, S) with Q a prefix and S a finite set of Q-sequents. By the two lemmas just proved we have *state transitions*

$$(Q, \{\mathcal{P} \Rightarrow \forall\vec{x}.\vec{D} \to A\} \cup S) \mapsto^{\varepsilon} (Q\forall\vec{x}, \{\mathcal{P} \cup \vec{D} \Rightarrow A\} \cup S)$$
$$(Q, \{\mathcal{P} \Rightarrow P\vec{r}\} \cup S) \mapsto^{\rho} (Q', (\{\mathcal{P} \Rightarrow \vec{G}^*\} \cup S)\rho),$$

where in the latter case there is a clause $\forall\vec{x}.\vec{G} \to P\vec{s}$ in $\mathcal{P}$ such that the following holds. Let $\vec{z}$ be the final universal variables in Q, $\vec{X}$ be new ("raised") variables such that $X_i\vec{z}$ has the same type as x_i, let Q^* be Q with the existential variables extended by $\vec{X}$, and let $*$ indicate the substitution $[x_1, \ldots, x_n{:=}X_1\vec{z}, \ldots, X_n\vec{z}]$, and $\mathsf{unif}(Q^*, \vec{r} = \vec{s}^*) = (Q', \rho)$.

Notice that by Lemma 3.2.1, if $\mathcal{P} \Rightarrow P\vec{r}$ is a Q-*sequent* (which means that $\bigwedge\!\!\!\bigwedge \mathcal{P} \to P\vec{r}$ is a Q-goal), then $(\mathcal{P} \Rightarrow \vec{G}^*)\rho$ is a Q'-sequent.

THEOREM 3.3.3. *Let Q be a prefix, and S be a set of Q-sequents. For every substitution φ we have: φ is a Q-substitution satisfying $\vdash S\varphi$ iff there is a prefix Q', a substitution ρ and a Q'-substitution φ' such that*

$$(Q, S) \mapsto^{\rho *} (Q', \emptyset),$$
$$\varphi = (\rho \circ \varphi')\restriction Q_{\exists}.$$

EXAMPLES. (i) The sequent $\forall y.\forall z Ryz \to Q, \forall y_1, y_2 Ry_1y_2 \Rightarrow Q$ leads first to $\forall y_1, y_2 Ry_1y_2 \Rightarrow Ryz$ under $\exists y\forall z$, then to $y_1 = y \wedge y_2 = z$ under $\exists y\forall z\exists y_1, y_2$, and finally to $Y_1z = y \wedge Y_2z = z$ under $\exists y, Y_1, Y_2\forall z$, which has the solution $Y_1 = \lambda zy$, $Y_2 = \lambda zz$.

(ii) $\forall y.\forall z Ryz \to Q, \forall y_1 Ry_1y_1 \Rightarrow Q$ leads first to $\forall y_1 Ry_1y_1 \Rightarrow Ryz$ under $\exists y\forall z$, then to $y_1 = y \wedge y_1 = z$ under $\exists y\forall z\exists y_1$, and finally to $Y_1z = y \wedge Y_1z = z$ under $\exists y, Y_1\forall z$, which has no solution.

(iii) Here is a more complex example (derived from proofs of the Orevkov-formulas), for which we only give the derivation tree.

$$\dfrac{\dfrac{\forall y.(\forall z Ryz \to \bot) \to \bot}{(\forall z R0z \to \bot) \to \bot} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\forall z S0z \to \bot}{S0z_1 \to \bot} \quad \dfrac{(*) \quad R0z \quad Rzz_1}{S0z_1}}{\bot}}{Rzz_1 \to \bot}}{\forall z_1 Rzz_1 \to \bot}}{\dfrac{\forall y.(\forall z_1 Ryz_1 \to \bot) \to \bot}{(\forall z_1 Rzz_1 \to \bot) \to \bot} \qquad \dfrac{\bot}{R0z \to \bot}}{\forall z R0z \to \bot}}{\bot}$$

where $(*)$ is a derivation from $\mathrm{Hyp}_1 \colon \forall z, z_1.R0z \to Rzz_1 \to S0z_1$.

3.4. Extension by ∧ and ∃

The extension by conjunction is rather easy; it is even superfluous in principle, since conjunctions can always be avoided at the expense of having lists of formulas instead of single formulas.

However, having conjunctions available is clearly useful at times, so let's add it. This requires the notion of an *elaboration path* for a formula (cf. [**27**]). The reason is that the property of a formula to have a unique atom as its *head* is lost when conjunctions are present. An elaboration path is meant to

give the directions (left or right) to go when we encounter a conjunction as a strictly positive subformula. For example, the elaboration paths of $\forall xA \wedge (B \wedge C \to D \wedge \forall yE)$ are $(\mathtt{left})$, $(\mathtt{right}, \mathtt{left})$ and $(\mathtt{right}, \mathtt{right})$. Clearly, a formula is equivalent to the conjunction (over all elaboration paths) of all formulas obtained from it by following an elaboration path (i.e. always throwing away the other part of the conjunction). In our example,

$$\forall xA \wedge (B \wedge C \to D \wedge \forall yE) \leftrightarrow \forall xA \wedge (B \wedge C \to D) \wedge (B \wedge C \to \forall yE).$$

In this way we regain the property of a formula to have a unique head, and our previous search procedure continues to work.

For the existential quantifier $\exists$ the problem is of a different nature. We chose to introduce $\exists$ by means of axiom schemata. Then the problem is which of such schemes to use in proof search, given a goal G and a set $\mathcal{P}$ of clauses. We might proceed as follows.

List all prime, positive and negative existential subformulas of $\mathcal{P} \Rightarrow G$, and remove any formula from those lists which is of the form of another one[1]. For every positive existential formula – say $\exists xB$ – add (the generalization of) the existence introduction scheme

$$\exists^+_{x,B} \colon \forall x.B \to \exists xB$$

to $\mathcal{P}$. Moreover, for every negative existential formula – say $\exists xA$ – and every (prime or existential) formula C in any of those two lists, exept the formula $\exists xA$ itself, add (the generalization of) the existence elimination scheme

$$\exists^-_{x,A,B} \colon \exists xA \to (\forall x.A \to B) \to B$$

to $\mathcal{P}$. Then start the search algorithm as described in Section 3.3. The normal form theorem for the natural deduction system of minimal logic with $\exists$ then guarantees completeness.

However, experience has shown that this complete search procedure tends to be trapped in too large a search space. Therefore in our actual implementation we decided to only take instances of the existence elimination scheme with *existential* conclusions.

Moreover, it seems appropriate that – before the search is started – one eliminates in a preprocessing step as many existential quantifiers as possible. We shall discuss in the following to what extent this might be done. As a preparation, we first prove a lemma.

Call a formula *decidable*, if it is built from atoms $\mathsf{atom}(t)$ and contains boolean quantifiers only (i.e., only quantifiers $\forall b$ or $\exists b$ with variables b of type $\mathbf{B}$).

Remark. Notice that this is a rather crude syntactical notion of decidability for formulas; a more elaborate version would also allow bounded quantification over e.g. the natural numbers. The lemma to follow says that it implies the derivability of $D \vee \neg D$, for the usual second order definition of $\vee$.

[1]To do this, for patterns the dual of the theory of "most general unifiers", i.e. a theory of "most special generalizations", needs to be developed.

LEMMA (Case distinction on decidable formulas). *For decidable formulas D we have*

$$\vdash_i (D \to A) \to (\neg D \to A) \to A.$$

PROOF. By induction on D. *Case* $\mathsf{atom}(t)$. To prove

$$\vdash (\mathsf{atom}(p) \to A) \to (\neg\mathsf{atom}(p) \to A) \to A,$$

use boolean induction, and the truth axiom $\mathsf{atom}(\mathtt{tt})$.

Case $D \to E$. We have to show

$$\vdash_i ((D \to E) \to A) \to (\neg(D \to E) \to A) \to A.$$

By IH it suffices to argue by cases on D and E. We proceed informally. If E holds, then we have $D \to E$ and hence A. If $\neg E$ holds, then distinguish cases on D. If D holds, then $\neg(D \to E)$, hence A. If $\neg D$ holds, then $D \to E$ by ex-falso, hence A.

Case $D \wedge E$. Easy, again using the IH, and cases on D and E.

Case $\exists pD$. We must show

$$\vdash_i (\exists pD \to A) \to (\neg\exists pD \to A) \to A.$$

By IH it suffices to argue by cases on $D[p{:=}\mathtt{tt}]$ and $D[p{:=}\mathtt{ff}]$. If either $D[p{:=}\mathtt{tt}]$ or $D[p{:=}\mathtt{ff}]$ hold, then we clearly have A, by the assumption $\exists pD \to A$. If both $\neg D[p{:=}\mathtt{tt}]$ and $\neg D[p{:=}\mathtt{ff}]$ hold, then $\forall p\neg D$ (by boolean induction), hence $\neg\exists pD$, hence A by the assumption $\neg\exists pD \to A$.

Case $\forall pD$. We must show

$$\vdash_i (\forall pD \to A) \to (\neg\forall pD \to A) \to A.$$

Use $\forall pD \leftrightarrow D[p{:=}\mathtt{tt}] \wedge D[p{:=}\mathtt{ff}]$ (provable by boolean induction), and the IH for $D[p{:=}\mathtt{tt}]$ and $D[p{:=}\mathtt{ff}]$. □

LEMMA (Independence of premise for decidable formulas). *For decidable formulas D we have*

$$\vdash_i (D \to \exists xA) \to \exists x.D \to A.$$

PROOF. One can see easily that

$$\vdash (D \to \exists xA) \to D \to \exists x.D \to A,$$
$$\vdash_i \neg D \to \exists x.D \to A.$$

Now use case distinction on decidable formulas (i.e., the lemma above). □

We now assign to every formula A an "existentially reduced" formula A^* equivalent to A. It is obtained by first moving existential quantifiers to the front (as much as possible), and then – if they appear to the left of an implication – changing them into universal quantifiers (again as much as possible). Both processes are done simultaneously in the following definition.

DEFINITION (A^*). In the recursive cases below, assume $A^* = \exists\vec{x}A_0$ with A_0 not an existential formula, and similarly for B.

$$\mathsf{atom}(t)^* := \mathsf{atom}(t),$$
$$(A \wedge B)^* := \exists\vec{x}\exists\vec{y}.A_0 \wedge B_0,$$

$$(A \to B)^* := \begin{cases} \exists \vec{y}.A^* \to B_0 & \text{if } A \text{ is decidable,} \\ \forall \vec{x}.A_0 \to B^* & \text{else, and } \vec{x} \text{ not empty,} \\ A^* \to B^* & \text{otherwise,} \end{cases}$$
$$(\forall x A)^* := \forall x A^*,$$
$$(\exists x A)^* := \exists x A^*.$$

THEOREM. $\vdash_i A \leftrightarrow A^*$.

PROOF. Use independence of premise for decidable formulas. □

3.5. Notes

I have benefitted from a presentation of Miller's [**27**] given by Ulrich Berger, in a logic seminar in München in June 1991. The type of restriction to higher order terms described in the text has been introduced in [**27**]; it has been called *patterns* by Nipkow [**30**]. Miller also noted its relevance for extensions of logic programming, and showed that the unification problem for patterns is solvable and admits most general unifiers. The present treatment was motivated by the desire to use Miller's approach as a basis for an implementation of a simple proof search engine for (first and higher order) minimal logic. This goal prompted us into several simplifications, optimizations and extensions, in particular the following.

- Instead of arbitrarily mixed prefixes we only use those of the form $\forall\exists\forall$. Nipkow in [**30**] already had presented a version of Miller's pattern unification algorithm for such prefixes, and Miller in [**27**, Section 9.2] notes that in such a situation any two unifiers can be transformed into each other by a variable renaming substitution. Here we restrict ourselves to $\forall\exists\forall$-prefixes throughout, i.e. in the proof search algorithm as well.
- The order of events in the pattern unification algorithm is changed slightly, by postponing the raising step until it is really needed. This avoids unnecessary creation of new higher type variables. – Already Miller noted in [**27**, p.515] that such optimizations are possible.
- The extensions concern the (strong) existential quantifier, which has been left out in Miller's treatment, and also conjunction. The latter can be avoided in principle, but of course is a useful thing to have.

Moreover, since part of the motivation to write this exposition was the necessity to have a guide for our implementation, we have paid particular attention to write at least the parts of the proofs with algorithmic content as clear and complete as possible.

CHAPTER 4

Program Extraction from Constructive Proofs

4.1. Quantifiers Without Computational Content

For program extraction it is useful to distinguish between quantifiers with and without computational content.

4.1.1. Rules for $\forall^{\mathsf{nc}}$ and $\exists^{\mathsf{nc}}$. We introduce $\forall^{\mathsf{nc}}$ and $\exists^{\mathsf{nc}}$ to indicate that there should be **n**o computational **c**ontent. The logical meaning of these quantifiers is unchanged. However, we have to observe a special *variable condition for $\forall^{\mathsf{nc}}$-introduction*: the variable to be abstracted should not be a *computational variable* in the given proof, i.e. the extracted program of this proof should not depend on x. So the rules for $\forall^{\mathsf{nc}}$ are

$$\frac{\begin{array}{c}\mid M\\ A\end{array}}{\forall^{\mathsf{nc}}xA}\,\forall^{\mathsf{nc}+}x \quad (\ldots \textit{ and } x \notin \mathsf{FV}([\![M]\!])) \qquad \frac{\begin{array}{cc}\mid M & \\ \forall^{\mathsf{nc}}xA & t\end{array}}{A[x{:=}t]}\,\forall^{\mathsf{nc}-}$$

Here $[\![M]\!]$ is the extracted term (or program) of M, defined below. The derivation term for the rule $\forall^{\mathsf{nc}+}x$ is written $\lambda^{\mathsf{nc}}xM$.

The existence introduction and elimination schemes

$$\exists^{+}_{x,B}\colon \forall x.B \to \exists xB$$
$$\exists^{-}_{x,A,B}\colon \exists xA \to (\forall x.A \to B) \to B \quad \text{with } x \notin \mathsf{FV}(B).$$

need to be modified for the nc-versions, to

$$(\exists^{\mathsf{nc}})^{+}_{x,B}\colon \forall^{\mathsf{nc}}x.B \to \exists^{\mathsf{nc}}xB$$
$$(\exists^{\mathsf{nc}})^{-}_{x,A,B}\colon \exists^{\mathsf{nc}}xA \to (\forall^{\mathsf{nc}}x.A \to B) \to B \quad \text{with } x \notin \mathsf{FV}(B).$$

4.1.2. Properties of $\forall^{\mathsf{nc}}$ and $\exists^{\mathsf{nc}}$.

Lemma ($\forall^{\mathsf{nc}}$ implies $\forall$). $\vdash \forall^{\mathsf{nc}}xA \to \forall xA$.

Proof.

$$\frac{\begin{array}{cc}u\colon \forall^{\mathsf{nc}}xA & x\end{array}}{\dfrac{A}{\forall xA}\,\forall^{+}x}$$

Here only the usual variable condition needs to be observed, which clearly is satisfied. □

Lemma ($\forall$ implies $\forall^{\mathsf{nc}}$). $\vdash \forall xA \to \forall^{\mathsf{nc}}xA$, *provided* $\tau(A) = \varepsilon$.

Proof.

$$\frac{\begin{array}{cc}u\colon \forall xA & x\end{array}}{\dfrac{A}{\forall^{\mathsf{nc}}xA}\,\forall^{\mathsf{nc}+}x} \quad \text{(with var.cond.)}$$

For the final inference to be correct we need $\tau(A) = \varepsilon$. □

We show that $\exists x A$ implies $\exists^{\mathsf{nc}} x A$. The converse cannot be expected, since there is no way to provide the x necessary to prove $\exists x A$ when we only know $\exists^{\mathsf{nc}} x A$.

LEMMA ($\exists$ implies $\exists^{\mathsf{nc}}$). $\vdash \exists x A \to \exists^{\mathsf{nc}} x A$.

PROOF.

$$\dfrac{(\exists^{\mathsf{nc}})^-_{x,A,B} \qquad \exists x A \qquad \dfrac{\text{L.}(\forall^{\mathsf{nc}} \text{ imp. } \forall) \qquad (\exists^{\mathsf{nc}})^+_{x,A} \colon \forall^{\mathsf{nc}} x.A \to \exists^{\mathsf{nc}} x A}{\forall x.A \to \exists^{\mathsf{nc}} x A}}{\exists^{\mathsf{nc}} x A}$$

□

LEMMA ($\exists^{\mathsf{nc}}$ implies $\exists^{\mathrm{ca}}$). $\vdash \exists^{\mathsf{nc}} x A \to \exists^{\mathrm{ca}} x A$.

PROOF. By definition, $\exists^{\mathrm{ca}} x A$ is $(\forall x.A \to F) \to F$. Assume $\forall x.A \to F$. Since this formula has no computational content (i.e., $\tau(\forall x.A \to F) = \varepsilon$), we obtain $\forall^{\mathsf{nc}} x.A \to F$. Now from the assumption $\exists^{\mathsf{nc}} x A$ the claim F follows by the axiom $(\exists^{\mathsf{nc}})^-_{x,A.F}$. □

4.1.3. Refining Axioms Using $\forall^{\mathsf{nc}}$ and $\exists^{\mathsf{nc}}$. This distinction between quantifiers with and without computational content allows us to refine our axioms. Generally, the universal closure of such an axiom should now be done with $\forall^{\mathsf{nc}}$-quantifiers. For instance, the compatibility axiom is taken to be

$$\mathsf{Eq\text{-}Compat}_{x_1,A} \colon \forall^{\mathsf{nc}} \vec{p}, x_1, x_2.x_1 \approx x_2 \to A \to A[x_1 := x_2].$$

The induction axioms can be refined; in case of the natural numbers we obtain a version corresponding to iteration.

4.2. Computational Content of Proofs

4.2.1. The Type of a Formula. We assign to every formula A an object $\tau(A)$ (a type or the symbol ε). $\tau(A)$ is intended to be the type of the program to be extracted from a proof of A. In case $\tau(A) = \varepsilon$ proofs of A have no computational content; such formulas A are called *Harrop formulas*.

Recall that we allow free predicate variables, to be viewed as placeholders for formulas (or more precisely, comprehension terms). Since we do not know in advance which formula will be substituted for a predicate variable, we use a type variable as the type of the program to be extracted from a proof of an atom involving a predicate variable. Therefore our definition of $\tau(A)$ is relative to a given assignment of type variables to some (see below) predicate variables.

$$\tau(P(\vec{s})) := \begin{cases} \alpha_P & \text{if } P \text{ is a predicate variable with assigned } \alpha_P \\ \varepsilon & \text{otherwise} \end{cases}$$

$$\tau(\exists x^\rho A) := \begin{cases} \rho & \text{if } \tau(A) = \varepsilon \\ \rho \times \tau(A) & \text{otherwise} \end{cases}$$

$$\tau(\forall x^\rho A) := \begin{cases} \varepsilon & \text{if } \tau(A) = \varepsilon \\ \rho \to \tau(A) & \text{otherwise} \end{cases}$$

$$\tau(\exists^{\mathsf{nc}} x^\rho A) := \tau(A)$$

$$\tau(\forall^{\mathsf{nc}} x^\rho A) := \tau(A)$$

$$\tau(A_0 \wedge A_1) := \begin{cases} \tau(A_i) & \text{if } \tau(A_{1-i}) = \varepsilon \\ \tau(A_0) \times \tau(A_1) & \text{otherwise} \end{cases}$$

$$\tau(A \to B) := \begin{cases} \tau(B) & \text{if } \tau(A) = \varepsilon \\ \varepsilon & \text{if } \tau(B) = \varepsilon \\ \tau(A) \to \tau(B) & \text{otherwise} \end{cases}$$

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

$$[\![u^A]\!] := x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated with } u^A)$$

$$[\![\lambda u^A M]\!] := \begin{cases} [\![M]\!] & \text{if } \tau(A) = \varepsilon \\ \lambda x_u^{\tau(A)} [\![M]\!] & \text{otherwise} \end{cases}$$

$$[\![M^{A\to B} N]\!] := \begin{cases} [\![M]\!] & \text{if } \tau(A) = \varepsilon \\ [\![M]\!][\![N]\!] & \text{otherwise} \end{cases}$$

$$[\![\langle M_0^{A_0}, M_1^{A_1}\rangle]\!] := \begin{cases} [\![M_i]\!] & \text{if } \tau(A_{1-i}) = \varepsilon \\ \langle [\![M_0]\!], [\![M_1]\!]\rangle & \text{otherwise} \end{cases}$$

$$[\![M^{A_0\wedge A_1} i]\!] := \begin{cases} [\![M]\!] & \text{if } \tau(A_{1-i}) = \varepsilon \\ [\![M]\!] i & \text{otherwise} \end{cases}$$

$$[\![(\lambda x^\rho M)^{\forall x A}]\!] := \lambda x^\rho [\![M]\!]$$

$$[\![M^{\forall x A} t]\!] := [\![M]\!] t$$

$$[\![(\lambda x^\rho M)^{\forall^{\mathsf{nc}} x A}]\!] := [\![M]\!]$$

$$[\![M^{\forall^{\mathsf{nc}} x A} t]\!] := [\![M]\!]$$

We also need extracted programs for induction, cases and $\exists$-axioms; these will be defined below. For derivations M^A where $\tau(A) = \varepsilon$ (i.e. A is a Harrop formula) we define $[\![M]\!] := \varepsilon$ (ε some new symbol). This applies in particular if A is $\exists$-free and contains no predicate variables.

4.2.3. Extracted Program of an Induction Axiom. Recall the general form of induction over simultaneous free algebras $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$, with goal formulas $\forall x_j^{\mu_j} A_j$, from Section 2.4.3. For the constructor type

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$$

we have the *step formula*

$$\begin{aligned} D_i := \forall y_1^{\rho_1}, \ldots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1\to\mu_{j_1}}, \ldots, y_{m+n}^{\vec{\sigma}_n\to\mu_{j_n}}.\ & \\ \forall \vec{x}^{\vec{\sigma}_1} A_{j_1}[x_{j_1} := y_{m+1}\vec{x}] \to \cdots \to & \\ \forall \vec{x}^{\vec{\sigma}_n} A_{j_n}[x_{j_n} := y_{m+n}\vec{x}] \to & \\ A_j[x_j := \mathsf{constr}_i^{\vec{\mu}}(\vec{y})]. & \end{aligned} \tag{10}$$

Here $\vec{y} = y_1^{\rho_1}, \dots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1 \to \mu_{j_1}}, \dots, y_{m+n}^{\vec{\sigma}_n \to \mu_{j_n}}$ are the *components* of the object $\mathsf{constr}_i^{\vec{\mu}}(\vec{y})$ of type μ_j under consideration, and

$$\forall \vec{x}^{\vec{\sigma}_1} A_{j_1}[x_{j_1} := y_{m+1}\vec{x}], \dots, \forall \vec{x}^{\vec{\sigma}_n} A_{j_n}[x_{j_n} := y_{m+n}\vec{x}]$$

are the hypotheses available when proving the induction step. The induction axiom $\mathsf{Ind}_{\mu_j}^{\vec{x},\vec{A}}$ or shortly Ind_j then proves the universal closure (w.r.t. $\forall^{\mathsf{nc}}$) of the formula

$$D_1 \to \cdots \to D_k \to \forall x_j^{\mu_j} A_j.$$

$[\![\mathsf{Ind}_j]\!]$ is defined to be the recursion operator $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$. Here $\vec{\mu}, \vec{\tau}$ list only the types μ_j, τ_j with $\tau_j := \tau(A_j) \neq \varepsilon$, i.e. the recursion operator is simplified accordingly, as indicated in Remark 2.2.1.

Remark. It is possible to use variants of the induction scheme, were some or all of the universal quantifiers in the step formula (10) have no computational content.

Example. For the induction scheme

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

we have

$$[\![\mathsf{Ind}_{n,A}]\!] := \mathcal{R}_{\mathbf{N}}^{\tau} \colon \tau \to (\mathbf{N} \to \tau \to \tau) \to \mathbf{N} \to \tau,$$

where $\tau := \tau(A) \neq \varepsilon$. The variant

$$A[n{:=}0] \to (\forall^{\mathsf{nc}} n.A \to A[n{:=}n+1]) \to \forall n A$$

has as extracted term the *iteration operator* of type $\tau \to (\tau \to \tau) \to \mathbf{N} \to \tau$.

4.2.4. Extracted Program of a Cases Axiom. Recall the cases axioms from Section 2.4.4. They again refer to simultaneous free algebras $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$, and goal formulas $\forall x_j^{\mu_j} A_j$. For the constructor type

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_1 \to \alpha_{j_1}) \to \cdots \to (\vec{\sigma}_n \to \alpha_{j_n}) \to \alpha_j \in \mathsf{KT}(\vec{\alpha})$$

we have the *step formula*

$$D_i := \forall y_1^{\rho_1}, \dots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1 \to \mu_{j_1}}, \dots, y_{m+n}^{\vec{\sigma}_n \to \mu_{j_n}}.A_j[x_j{:=}\mathsf{constr}_i^{\vec{\mu}}(\vec{y})].$$

Here $\vec{y} = y_1^{\rho_1}, \dots, y_m^{\rho_m}, y_{m+1}^{\vec{\sigma}_1 \to \mu_{j_1}}, \dots, y_{m+n}^{\vec{\sigma}_n \to \mu_{j_n}}$ are the *components* of the object $\mathsf{constr}_i^{\vec{\mu}}(\vec{y})$ of type μ_j under consideration; notice that no induction hypotheses are available. The cases axiom Cases_{x_j,A_j}or shortly Cases_j then proves the universal closure of the formula

$$D_{i_1}, \dots, D_{i_q} \to \forall x_j^{\mu_j} A_j.$$

where $D_{i_1}, \dots, D_{i_q}$ consists of all D_i concerning constructors for μ_j.

$[\![\mathsf{Cases}_j]\!]$ is defined by means of the `if`-construct – for the reasons given in Remark 2.2.3 – to be

$$[\![\mathsf{Cases}_j]\!] := \lambda f_1 \dots \lambda f_q \lambda x[\mathtt{if}\ x\ f_1\ \dots\ f_q],$$

provided $\tau(A_j) \neq \varepsilon$.

EXAMPLE. For the cases axioms

$$\mathsf{Cases}_{n,A}\colon A[n{:=}0] \to \forall n A[n{:=}\mathsf{S}n] \to \ \forall n^{\mathbf{N}} A,$$
$$\mathsf{Cases}_{l,A}\colon A[l{:=}\mathsf{Nil}] \to \forall x, l A[l{:=}\mathsf{cons}(x,l)] \to \forall l^{\mathbf{L}(\alpha)} A.$$

we have

$$[\![\mathsf{Cases}_{n,A}]\!] := \lambda f_1 \lambda f_2 \lambda n[\mathtt{if}\ n\ f_1\ f_2],$$
$$[\![\mathsf{Cases}_{l,A}]\!] := \lambda f_1 \lambda f_2 \lambda l[\mathtt{if}\ l\ f_1\ f],$$

where we assume $\tau(A) \neq \varepsilon$.

4.2.5. Extracted Programs of Existence Axioms. For the axioms

$$\exists^+_{x,B}\colon \forall x^\rho.A \to \exists x^\rho A \qquad \mathtt{Ex\text{-}Intro}$$
$$\exists^-_{x,A,B}\colon \exists x^\rho A \to (\forall x^\rho.A \to B) \to B \qquad \mathtt{Ex\text{-}Elim}$$

we set

$$[\![\exists^+_{x^\rho,A}]\!] := \begin{cases} \lambda x^\rho x & \text{if } \tau(A) = \varepsilon \\ \lambda x^\rho \lambda y^{\tau(A)} \langle x, y\rangle & \text{otherwise} \end{cases}$$
$$[\![\exists^-_{x^\rho,A,B}]\!] := \begin{cases} \lambda x^\rho \lambda f^{\rho\to\tau(B)}.fx & \text{if } \tau(A) = \varepsilon \\ \lambda z^{\rho\times\tau(A)} \lambda f^{\rho\to\tau(A)\to\tau(B)}.f(z0)(z1) & \text{otherwise} \end{cases}$$

and for the axioms

$$(\exists^{\mathsf{nc}})^+_{x,B}\colon \forall^{\mathsf{nc}} x^\rho.A \to \exists^{\mathsf{nc}} x^\rho A \qquad \mathtt{Exnc\text{-}Intro}$$
$$(\exists^{\mathsf{nc}})^-_{x,A,B}\colon \exists^{\mathsf{nc}} x^\rho A \to (\forall^{\mathsf{nc}} x^\rho.A \to B) \to B \qquad \mathtt{Exnc\text{-}Elim}$$

we set

$$[\![(\exists^{\mathsf{nc}})^+_{x^\rho,A}]\!] := \lambda y^{\tau(A)} y$$
$$[\![(\exists^{\mathsf{nc}})^-_{x^\rho,A,B}]\!] := \begin{cases} \lambda z^{\tau(B)}.z & \text{if } \tau(A) = \varepsilon \\ \lambda y^{\tau(A)} \lambda f^{\tau(A)\to\tau(B)}.fy & \text{otherwise} \end{cases}$$

4.2.6. Extracted Program of a Compatibility Axiom. For the compatibility axiom

$$\mathsf{Eq\text{-}Compat}_{x_1,A}\colon \forall^{\mathsf{nc}} \vec{p}, x_1, x_2.x_1 \approx x_2 \to A \to A[x_1{:=}x_2]$$

we set

$$[\![\mathsf{Eq\text{-}Compat}_{x_1,A}]\!] := \lambda y^{\tau(A)} y.$$

4.3. Realizability

Finally we define the notion of *(modified) realizability*. The term "modified" is used sometimes for historical reasons, to distinguish this form of realizability from the (earlier) Kleene-style realizability. More precisely, we define formulas $r\,\mathbf{r}\,A$, where A is a formula and r is either a term of type $\tau(A)$ if the latter is a type, or the symbol ε if $\tau(A) = \varepsilon$.

4.3.1. Definition of Realizability. In order to define realizability for atoms built with predicate variables, we have to provide for every predicate variable P of arity $\vec{\rho}$ with assigned α_P a new predicate variable $P^{\mathbf{r}}$ of arity $\alpha_P, \vec{\rho}$.

$$r\,\mathbf{r}\,P(\vec{s}) \;=\; \begin{cases} P^{\mathbf{r}}(r,\vec{s}) & \text{if } P \text{ is a predicate variable with assigned } \alpha_P \\ P(\vec{s}) & \text{if } P \text{ is a predicate constant} \end{cases}$$

$$r\,\mathbf{r}\,(\exists x A) \;=\; \begin{cases} \varepsilon\,\mathbf{r}\,A[x{:=}r] & \text{if } \tau(A)=\varepsilon \\ r1\,\mathbf{r}\,A[x{:=}r0] & \text{otherwise} \end{cases}$$

$$r\,\mathbf{r}\,(\forall x A) \;=\; \begin{cases} \forall x.\varepsilon\,\mathbf{r}\,A & \text{if } \tau(A)=\varepsilon \\ \forall x.rx\,\mathbf{r}\,A & \text{otherwise} \end{cases}$$

$$r\,\mathbf{r}\,(\exists^{\mathsf{nc}} x A) \;=\; \begin{cases} \exists^{\mathsf{nc}} x.\varepsilon\,\mathbf{r}\,A & \text{if } \tau(A)=\varepsilon \\ \exists^{\mathsf{nc}} x.r\,\mathbf{r}\,A & \text{otherwise} \end{cases}$$

$$r\,\mathbf{r}\,(\forall^{\mathsf{nc}} x A) \;=\; \begin{cases} \forall^{\mathsf{nc}} x.\varepsilon\,\mathbf{r}\,A & \text{if } \tau(A)=\varepsilon \\ \forall^{\mathsf{nc}} x.r\,\mathbf{r}\,A & \text{otherwise} \end{cases}$$

$$r\,\mathbf{r}\,(A\to B) \;=\; \begin{cases} \varepsilon\,\mathbf{r}\,A \;\to\; r\,\mathbf{r}\,B & \text{if } \tau(A)=\varepsilon \\ \forall x.x\,\mathbf{r}\,A \;\to\; \varepsilon\,\mathbf{r}\,B & \text{if } \tau(A)\neq\varepsilon=\tau(B) \\ \forall x.x\,\mathbf{r}\,A \;\to\; rx\,\mathbf{r}\,B & \text{otherwise} \end{cases}$$

$$r\,\mathbf{r}\,(A_0\wedge A_1) \;=\; \begin{cases} \varepsilon\,\mathbf{r}\,A_0 \;\wedge\; r\,\mathbf{r}\,A_1 & \text{if } \tau(A_0)=\varepsilon \\ r\,\mathbf{r}\,A_0 \;\wedge\; \varepsilon\,\mathbf{r}\,A_1 & \text{if } \tau(A_1)=\varepsilon \\ r0\,\mathbf{r}\,A_0 \;\wedge\; r1\,\mathbf{r}\,A_1 & \text{otherwise} \end{cases}$$

Formulas which do not contain the existence quantifier $\exists$ play a special role in this context; we call them $\exists$-*free* (or *invariant*) formulas; in the literature such formulas are also called "negative". Their crucial property is that for an $\exists$-free formula A without predicate variables P with assigned α_P we have $\tau(A)=\varepsilon$ and $\varepsilon\,\mathbf{r}\,A = A$. In particular, we have to assign an α_P to every predicate variable P of "existence degree" $\neq 0$, which means that it can be substituted by a formula containing $\exists$ or other predicate variables Q with assigned α_Q.

For the formulation of the soundness theorem below it will be useful to let $x_u := \varepsilon$ if u^A is an assumption variable with a Harrop formula A. The soundness theorem says that for every derivation M of a formula B there is a derivation $\mu(M)$ of $[\![M]\!]\,\mathbf{r}\,B$ from assumptions $\{\, x_u\,\mathbf{r}\,C \mid u^C \in \mathsf{FA}(M)\,\}$. We first tackle this for the axioms.

4.3.2. Realizing an Induction Axiom. The induction axiom $\mathsf{Ind}_{\mu_j}^{\vec{x},\vec{A}}$ or shortly Ind_j proves the universal closure (w.r.t. $\forall^{\mathsf{nc}}$) of the formula

$$D_1 \to \cdots \to D_k \to \forall x_j^{\mu_j}\, A_j,$$

with step formulas D_i from (10). We may assume $\tau(A_j)\neq\varepsilon$ for at least one j, for otherwise $[\![\mathsf{Ind}_j]\!]=\varepsilon$ and we can easily find a derivation $\mu(\mathsf{Ind}_j)$ of

$$\begin{aligned} &\varepsilon\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{p}.D_1\to\cdots\to D_k\to\forall x_j^{\mu_j}\,A_j\big) \\ &= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,\big(D_1\to\cdots\to D_k\to\forall x_j^{\mu_j}\,A_j\big) \end{aligned}$$

$$= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,D_1 \to \cdots \to \varepsilon\,\mathbf{r}\,D_k \to \forall x_j^{\mu_j}\varepsilon\,\mathbf{r}\,A_j.$$

Indeed, $\mu(\mathsf{Ind}_j)$ is just $\mathsf{Ind}_{\mu_j}^{x_1,\dots,x_N,\varepsilon\,\mathbf{r}\,A_1,\dots,\varepsilon\,\mathbf{r}\,A_N}$.

Recall $[\![\mathsf{Ind}_j]\!] = \mathcal{R}_j$. Hence we must find a derivation $\mu(\mathsf{Ind}_j)$ of

$$\begin{aligned}
&\mathcal{R}_j\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{p}.D_1 \to \cdots \to D_k \to \forall x_j^{\mu_j}\,A_j\big)\\
&= \forall^{\mathsf{nc}}\vec{p}.\mathcal{R}_j\vec{p}\,\mathbf{r}\,\big(D_1 \to \cdots \to D_k \to \forall x_j^{\mu_j}\,A_j\big)\\
&= \forall^{\mathsf{nc}}\vec{p}, f_1.f_1\,\mathbf{r}\,D_1 \to \cdots \to \forall f_k.f_k\,\mathbf{r}\,D_k \to \forall x_j^{\mu_j}\mathcal{R}_j\vec{f}x_j^{\mu_j}\,\mathbf{r}\,A_j.
\end{aligned}$$

Notice that for the step formula D_i spelled out in (10) the formula $f_i\,\mathbf{r}\,D_i$ is defined to be (we assume for simplicity $\tau(A_j) \neq \varepsilon$)

$$\begin{aligned}
f_i\,\mathbf{r}\,D_i :=& \forall y_1^{\rho_1},\dots,y_m^{\rho_m},y_{m+1}^{\vec{\sigma}_1\to\mu_{j_1}},\dots,y_{m+n}^{\vec{\sigma}_n\to\mu_{j_n}}\\
&\forall z_1.\big(\forall\vec{x}^{\vec{\sigma}_1}\,z_1\vec{x}\,\mathbf{r}\,A_{j_1}[x_{j_1}{:=}y_{m+1}\vec{x}]\big) \to \cdots \to\\
&\forall z_n.\big(\forall\vec{x}^{\vec{\sigma}_n}\,z_n\vec{x}\,\mathbf{r}\,A_{j_n}[x_{j_n}{:=}y_{m+n}\vec{x}]\big) \to\\
&f_i\vec{y}z_1\dots z_n\,\mathbf{r}\,A_j[x_j{:=}\mathsf{constr}_i^{\vec{\mu}}(\vec{y})].
\end{aligned}$$

We proceed informally. Assume $\vec{p}$, f_1, $u_1\colon f_1\,\mathbf{r}\,D_1$, …, f_k, $u_k\colon f_k\,\mathbf{r}\,D_k$; our goal is $\forall x_j^{\mu_j}\mathcal{R}_j\vec{f}x_j^{\mu_j}\,\mathbf{r}\,A_j$. For the proof we use simultaneous induction w.r.t. all $\forall x_j^{\mu_j}\mathcal{R}_j\vec{f}x_j^{\mu_j}\,\mathbf{r}\,A_j$. Hence it suffices to prove the step formulas w.r.t. the latter induction, i.e. all formulas

$$\begin{aligned}
D_i^{\mathbf{r}} :=& \forall y_1^{\rho_1},\dots,y_m^{\rho_m},y_{m+1}^{\vec{\sigma}_1\to\mu_{j_1}},\dots,y_{m+n}^{\vec{\sigma}_n\to\mu_{j_n}}.\\
&\forall\vec{x}^{\vec{\sigma}_1}\,\mathcal{R}_{j_1}\vec{f}(y_{m+1}\vec{x})\,\mathbf{r}\,A_{j_1}[x_{j_1}{:=}y_{m+1}\vec{x}] \to \cdots \to\\
&\forall\vec{x}^{\vec{\sigma}_n}\,\mathcal{R}_{j_n}\vec{f}(y_{m+n}\vec{x})\,\mathbf{r}\,A_{j_n}[x_{j_n}{:=}y_{m+n}\vec{x}] \to\\
&\mathcal{R}_j\vec{f}(\mathsf{constr}_i^{\vec{\mu}}(\vec{y}))\,\mathbf{r}\,A_j[x_j{:=}\mathsf{constr}_i^{\vec{\mu}}(\vec{y})].
\end{aligned}$$

So assume $y_1^{\rho_1},\dots,y_m^{\rho_m},y_{m+1}^{\vec{\sigma}_1\to\mu_{j_1}},\dots,y_{m+n}^{\vec{\sigma}_n\to\mu_{j_n}}$ and

$$\begin{aligned}
&v_1\colon \forall\vec{x}^{\vec{\sigma}_1}\,\mathcal{R}_{j_1}\vec{f}(y_{m+1}\vec{x})\,\mathbf{r}\,A_{j_1}[x_{j_1}{:=}y_{m+1}\vec{x}],\dots,\\
&v_n\colon \forall\vec{x}^{\vec{\sigma}_n}\,\mathcal{R}_{j_n}\vec{f}(y_{m+n}\vec{x})\,\mathbf{r}\,A_{j_n}[x_{j_n}{:=}y_{m+n}\vec{x}].
\end{aligned}$$

We must show $\mathcal{R}_j\vec{f}(\mathsf{constr}_i^{\vec{\mu}}(\vec{y}))\,\mathbf{r}\,A_j[x_j{:=}\mathsf{constr}_i^{\vec{\mu}}(\vec{y})]$. For the proof use $u_i^{f_i\,\mathbf{r}\,D_i}$ with

$$\vec{y},(\mathcal{R}_{j_1}\vec{f})\circ y_{m+1},v_1,\dots,(\mathcal{R}_{j_n}\vec{f})\circ y_{m+n},v_n$$

and recall that

$$\mathcal{R}_j\vec{f}(\mathsf{constr}_i^{\vec{\mu}}(\vec{y})) = f_i\vec{y}\big((\mathcal{R}_{j_1}\vec{f})\circ y_{m+1}\big)\dots,\big((\mathcal{R}_{j_n}\vec{f})\circ y_{m+n}\big).$$

Examples. Consider a particular induction axiom, say on the natural numbers $\mathbf{N}$:

$$\mathsf{Ind}_{n,A}\colon \forall^{\mathsf{nc}}\vec{p}.A[n{:=}0] \to (\forall n.A \to A[n{:=}n+1]) \to \forall nA.$$

We must find a derivation $\mu(\mathsf{Ind}_{n,A})$ of

$$[\![\mathsf{Ind}_{n,A}]\!]\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{p}.A[n{:=}0] \to (\forall n.A \to A[n{:=}n+1]) \to \forall nA\big).$$

Case $\tau(A) \neq \varepsilon$. Then $[\![\mathsf{Ind}_{n,A}]\!] = \mathcal{R}_{\mathbf{N}}^{\tau}$ and we obtain

$$\begin{aligned}
&\mathcal{R}_{\mathbf{N}}^{\tau}\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.A[n{:=}0] \to (\forall n.A \to A[n{:=}n+1]) \to \forall nA\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f_1.f_1\,\mathbf{r}\,A[n{:=}0] \to \forall f_2.f_2\,\mathbf{r}\,(\forall n.A \to A[n{:=}n+1]) \to
\end{aligned}$$

$$\begin{aligned}
&\forall n(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)\\
=\ &\forall^{\mathsf{nc}}\vec{p}\forall f_1.f_1\,\mathbf{r}\,A[n{:=}0] \to \forall f_2.(\forall n, y.y\,\mathbf{r}\,A \to f_2 n y\,\mathbf{r}\,A[n{:=}n+1]) \to\\
&\forall n(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)
\end{aligned}$$

Hence we can define $\mu(\mathsf{Ind}_{n,A})$ to be the obvious inductive proof of this formula, which uses the induction axiom

$$\begin{aligned}
&\mathsf{Ind}_{n,(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)} \colon \forall^{\mathsf{nc}}\vec{p}.\\
&\quad (\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)[n{:=}0] \to\\
&\quad (\forall n.\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A \to (\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)[n{:=}n+1]) \to\\
&\quad \forall n(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A).
\end{aligned}$$

Notice that because of our identification of $\beta\eta\mathcal{R}$-equivalent terms this is the same as

$$\begin{aligned}
&\mathsf{Ind}_{n,(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)} \colon \forall^{\mathsf{nc}}\vec{p}.\\
&\quad f_1\,\mathbf{r}\,A[n{:=}0] \to\\
&\quad (\forall n.\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A \to f_2 n(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n)\,\mathbf{r}\,A[n{:=}n+1]) \to\\
&\quad \forall n(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A).
\end{aligned}$$

Let

$$\begin{aligned}
u_1 &\colon f_1\,\mathbf{r}\,A[n{:=}0]\\
u_2 &\colon \forall n, y.y\,\mathbf{r}\,A \to f_2 n y\,\mathbf{r}\,A[n{:=}n+1]\\
v &\colon \mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A
\end{aligned}$$

be assumption variables. Then the proof term is

$$\lambda\vec{p}\lambda f_1\lambda u_1\lambda f_2\lambda u_2.\mathsf{Ind}_{n,(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n\,\mathbf{r}\,A)}\vec{p}u_1(\lambda n\lambda v.u_2 n(\mathcal{R}^\tau_{\mathbf{N}} f_1 f_2 n)v)$$

Case $\tau(A) = \varepsilon$. Then $[\![\mathsf{Ind}_{n,A}]\!] = \varepsilon$ and we obtain

$$\begin{aligned}
&\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.A[n{:=}0] \to (\forall n.A \to A[n{:=}n+1]) \to \forall n A\\
&= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,A[n{:=}0] \to (\forall n.\varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,A[n{:=}n+1]) \to\\
&\quad \forall n(\varepsilon\,\mathbf{r}\,A)\\
&= \forall^{\mathsf{nc}}\vec{p}.(\varepsilon\,\mathbf{r}\,A)[n{:=}0] \to (\forall n.\varepsilon\,\mathbf{r}\,A \to (\varepsilon\,\mathbf{r}\,A)[n{:=}n+1]) \to\\
&\quad \forall n(\varepsilon\,\mathbf{r}\,A).
\end{aligned}$$

Hence we can define $\mu(\mathsf{Ind}_{n,A}) = \mathsf{Ind}_{n,(\varepsilon\,\mathbf{r}\,A)}$.

Let us also treat an example of simultaneous induction. Consider an induction axiom for the simultaneously generated free algebras tree and tlist:

$$\begin{aligned}
&\forall^{\mathsf{nc}}\vec{p}.\forall n A[b{:=}\mathsf{Leaf}\,n] \to\\
&\quad (\forall bs.B \to A[b{:=}\mathsf{Branch}\,bs]) \to\\
&\quad B[bs{:=}\mathsf{Empty}] \to\\
&\quad (\forall b, bs.A \to B \to B[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\quad \forall b A
\end{aligned}$$

Case $\tau(A) \neq \varepsilon \neq \tau(B)$. Then $[\![\mathsf{Ind}_{b,bs,A,B}]\!] = \lambda\vec{p}\mathcal{R}_{\mathsf{tree}}$ and we obtain (writing $\vec{f}$ for $f_1 f_2 f_3 f_4$)

$$\lambda\vec{p}\mathcal{R}_{\mathsf{tree}}\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\forall n A[b{:=}\mathsf{Leaf}\,n] \to$$

$$
\begin{aligned}
&(\forall bs.B \to A[b{:=}\mathsf{Branch}\ bs]) \to \\
&B[bs{:=}\mathsf{Empty}] \to \\
&(\forall b, bs.A \to B \to B[bs{:=}\mathsf{Tcons}\, b\ bs]) \to \\
&\forall bA \\
= \forall^{\mathsf{nc}}\vec{p}.\forall f_1.f_1\,\mathbf{r}\,\forall nA[b{:=}\mathsf{Leaf}\ n] &\to \\
&\forall f_2.f_2\,\mathbf{r}\,(\forall bs.B \to A[b{:=}\mathsf{Branch}\ bs]) \to \\
&\forall f_3.f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to \\
&\forall f_4.f_4\,\mathbf{r}\,(\forall b, bs.A \to B \to B[bs{:=}\mathsf{Tcons}\, b\ bs]) \to \\
&\forall b(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A) \\
= \forall^{\mathsf{nc}}\vec{p}.\forall f_1.\forall n(f_1 n\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\ n]) &\to \\
&\forall f_2.(\forall bs, y.y\,\mathbf{r}\,B \to f_2\ bs\ y\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\ bs]) \to \\
&\forall f_3.f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to \\
&\forall f_4.(\forall b, bs, y_1.y_1\,\mathbf{r}\,A \to \forall y_2.y_2\,\mathbf{r}\,B \to \\
&\qquad f_4\ b\ bs\ y_1\ y_2\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\, b\ bs]) \to \\
&\forall b(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A)
\end{aligned}
$$

Hence we can define $\mu(\mathsf{Ind}_{b,bs,A,B})$ to be the expected inductive proof of this formula, which uses the induction axiom

$$
\begin{aligned}
&\mathsf{Ind}_{b,bs,(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A),(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B)} : \forall^{\mathsf{nc}}\vec{p}\forall\vec{f}. \\
&\forall n(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A)[b{:=}\mathsf{Leaf}\ n] \to \\
&(\forall bs.\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B \to (\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A)[b{:=}\mathsf{Branch}\ bs]) \to \\
&(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B)[bs{:=}\mathsf{Empty}] \to \\
&(\forall b, bs.\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A \to \mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B \to \\
&\qquad (\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B)[bs{:=}\mathsf{Tcons}\, b\ bs]) \to \\
&\forall b(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A)
\end{aligned}
$$

Because of the recursion equations

$$
\begin{aligned}
\mathcal{R}_{\mathsf{tree}}\vec{f}(\mathsf{Leaf}\ n) &= f_1 n \\
\mathcal{R}_{\mathsf{tree}}\vec{f}(\mathsf{Branch}\ bs) &= f_2\ bs(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs) \\
\mathcal{R}_{\mathsf{tlist}}\vec{f}\mathsf{Empty} &= f_3 \\
\mathcal{R}_{\mathsf{tlist}}\vec{f}(\mathsf{Tcons}\, b\ bs) &= f_4\ b\ bs(\mathcal{R}_{\mathsf{tree}}\vec{f}b)(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs)
\end{aligned}
$$

and our identification of $\beta\eta\mathcal{R}$-equivalent terms this is the same as

$$
\begin{aligned}
&\mathsf{Ind}_{b,bs,(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A),(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B)} : \forall^{\mathsf{nc}}\vec{p}\forall\vec{f}. \\
&\forall n(f_1 n\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\ n]) \to \\
&(\forall bs.\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B \to f_2\ bs(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs)\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\ bs]) \to \\
&f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to \\
&(\forall b, bs.\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A \to \mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B \to
\end{aligned}
$$

$$f_4\, b\, bs(\mathcal{R}_{\mathsf{tree}}\vec{f}b)(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs)\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\,b\,bs]) \to$$
$$\forall b(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A)$$

Let

$$u_1\colon \forall n(f_1 n\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n])$$
$$u_2\colon \forall bs, y.y\,\mathbf{r}\,B \to f_2\,bs\,y\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs]$$
$$u_3\colon f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}]$$
$$u_4\colon \forall b, bs, y_1.y_1\,\mathbf{r}\,A \to \forall y_2.y_2\,\mathbf{r}\,B \to f_4\,b\,bs\,y_1\,y_2\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\,b\,bs]$$

be assumption variables. Then the proof term is

$$\lambda\vec{p}\lambda f_1\lambda u_1\lambda f_2\lambda u_2\lambda f_3\lambda u_3\lambda f_4\lambda u_4.\mathsf{Ind}_{b,bs,(\mathcal{R}_{\mathsf{tree}}\vec{f}b\,\mathbf{r}\,A),(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs\,\mathbf{r}\,B)}\vec{p}\vec{f}$$
$$u_1(\lambda bs.u_2\,bs(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs))u_3(\lambda b\lambda bs.u_4\,b\,bs(\mathcal{R}_{\mathsf{tree}}\vec{f}b)v(\mathcal{R}_{\mathsf{tlist}}\vec{f}bs))$$

Case $\tau(A) = \varepsilon \neq \tau(B)$. Then $[\![\mathsf{Ind}_{b,bs,A,B}]\!] = \varepsilon$ and we obtain

$$\begin{aligned}
&\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\forall nA[b{:=}\mathsf{Leaf}\,n] \to\\
&\qquad(\forall bs.B \to A[b{:=}\mathsf{Branch}\,bs]) \to\\
&\qquad B[bs{:=}\mathsf{Empty}] \to\\
&\qquad(\forall b, bs.A \to B \to B[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\qquad\forall bA\\
&= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,\forall nA[b{:=}\mathsf{Leaf}\,n] \to\\
&\qquad\varepsilon\,\mathbf{r}\,(\forall bs.B \to A[b{:=}\mathsf{Branch}\,bs]) \to\\
&\qquad\forall f_3.f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to\\
&\qquad\forall f_4.f_4\,\mathbf{r}\,(\forall b, bs.A \to B \to B[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\qquad\forall b(\varepsilon\,\mathbf{r}\,A)\\
&= \forall^{\mathsf{nc}}\vec{p}.\forall n(\varepsilon\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n]) \to\\
&\qquad(\forall bs, y.y\,\mathbf{r}\,B \to \varepsilon\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs]) \to\\
&\qquad\forall f_3.f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to\\
&\qquad\forall f_4.(\forall b, bs.\varepsilon\,\mathbf{r}\,A \to \forall y_2.y_2\,\mathbf{r}\,B \to\\
&\qquad\qquad f_4\,b\,bs\,y_2\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\qquad\forall b(\varepsilon\,\mathbf{r}\,A)
\end{aligned}$$

Hence we can define $\mu(\mathsf{Ind}_{b,bs,A,B})$ to be the expected inductive proof of this formula, which uses the induction axiom

$$\begin{aligned}
&\mathsf{Ind}_{b,bs,(\varepsilon\,\mathbf{r}\,A),(F\,f_3\,f_4\,bs\,\mathbf{r}\,B)}\colon \forall^{\mathsf{nc}}\vec{p}\forall f_3, f_4.\\
&\quad\forall n(\varepsilon\,\mathbf{r}\,A)[b{:=}\mathsf{Leaf}\,n] \to\\
&\quad(\forall bs.F\,f_3\,f_4\,bs\,\mathbf{r}\,B \to (\varepsilon\,\mathbf{r}\,A)[b{:=}\mathsf{Branch}\,bs]) \to\\
&\quad(F\,f_3\,f_4\,bs\,\mathbf{r}\,B)[bs{:=}\mathsf{Empty}] \to\\
&\quad(\forall b, bs.\varepsilon\,\mathbf{r}\,A \to F\,f_3\,f_4\,bs\,\mathbf{r}\,B \to\\
&\qquad(F\,f_3\,f_4\,bs\,\mathbf{r}\,B)[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\quad\forall b(\varepsilon\,\mathbf{r}\,A)
\end{aligned}$$

with f defined by the recursion equations

$$\begin{aligned} F f_3 f_4 \mathsf{Empty} &= f_3, \\ F f_3 f_4 (\mathsf{Tcons}\, b\, bs) &= f_4\, b\, bs (F f_3 f_4 bs). \end{aligned}$$

By our identification of $\beta\eta\mathcal{R}$-equivalent terms this is the same as

$$\begin{aligned} &\mathsf{Ind}_{b,bs,(\varepsilon\,\mathbf{r}\,A),(F f_3 f_4 bs\,\mathbf{r}\,B)} : \forall^{\mathsf{nc}}\vec{p}\forall f_3, f_4. \\ &\quad \forall n(\varepsilon\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n]) \to \\ &\quad (\forall bs. F f_3 f_4 bs\,\mathbf{r}\,B \to \varepsilon\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs]) \to \\ &\quad f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to \\ &\quad (\forall b, bs. \varepsilon\,\mathbf{r}\,A \to F f_3 f_4 bs\,\mathbf{r}\,B \to \\ &\qquad f_4\, b\, bs (F f_3 f_4 bs)\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\, b\, bs]) \to \\ &\quad \forall b(\varepsilon\,\mathbf{r}\,A) \end{aligned}$$

Let

$$\begin{aligned} &u_1 : \forall n(\varepsilon\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n]) \\ &u_2 : \forall bs, y. y\,\mathbf{r}\,B \to \varepsilon\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs] \\ &u_3 : f_3\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \\ &u_4 : \forall b, bs. \varepsilon\,\mathbf{r}\,A \to \forall y_2. y_2\,\mathbf{r}\,B \to f_4\, b\, bs\, y_2\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\, b\, bs] \\ &v_1 : \varepsilon\,\mathbf{r}\,A \\ &v_2 : F f_3 f_4 bs\,\mathbf{r}\,B \end{aligned}$$

be assumption variables. Then the proof term is

$$\begin{aligned} &\lambda\vec{p}\lambda u_1\lambda u_2\lambda f_3\lambda u_3\lambda f_4\lambda u_4.\mathsf{Ind}_{b,bs,(\varepsilon\,\mathbf{r}\,A),(F f_3 f_4 bs\,\mathbf{r}\,B)}\vec{p} f_3 f_4 \\ &\quad u_1(\lambda bs. u_2\, bs (F f_3 f_4 bs)) u_3 (\lambda b\lambda bs\lambda v_1. u_4\, b\, bs\, v_1 (F f_3 f_4 bs) v_2). \end{aligned}$$

Case $\tau(A) \neq \varepsilon = \tau(B)$. Then $[\![\mathsf{Ind}_{b,bs,A,B}]\!] = F$ and we obtain

$$\begin{aligned} &F\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\forall n A[b{:=}\mathsf{Leaf}\,n] \to \\ &\qquad (\forall bs. B \to A[b{:=}\mathsf{Branch}\,bs]) \to \\ &\qquad B[bs{:=}\mathsf{Empty}] \to \\ &\qquad (\forall b, bs. A \to B \to B[bs{:=}\mathsf{Tcons}\, b\, bs]) \to \\ &\qquad \forall b A \\ &= \forall^{\mathsf{nc}}\vec{p}.\forall f_1. f_1\,\mathbf{r}\,\forall n A[b{:=}\mathsf{Leaf}\,n] \to \\ &\qquad \forall f_2. f_2\,\mathbf{r}\,(\forall bs. B \to A[b{:=}\mathsf{Branch}\,bs]) \to \\ &\qquad \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to \\ &\qquad \varepsilon\,\mathbf{r}\,(\forall b, bs. A \to B \to B[bs{:=}\mathsf{Tcons}\, b\, bs]) \to \\ &\qquad \forall b(F f_1 f_2 b\,\mathbf{r}\,A) \\ &= \forall^{\mathsf{nc}}\vec{p}.\forall f_1.\forall n(f_1 n\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n]) \to \\ &\qquad \forall f_2.(\forall bs.\varepsilon\,\mathbf{r}\,B \to f_2\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs]) \to \\ &\qquad \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to \\ &\qquad (\forall b, bs, y_1. y_1\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B \to \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\, b\, bs]) \to \\ &\qquad \forall b(F f_1 f_2 b\,\mathbf{r}\,A). \end{aligned}$$

Hence we can define $\mu(\mathsf{Ind}_{b,bs,A,B})$ to be the expected inductive proof of this formula, which uses the induction axiom

$$\begin{aligned}
&\mathsf{Ind}_{b,bs,(F\,f_1\,f_2\,b\,\mathbf{r}\,A),(\varepsilon\,\mathbf{r}\,B)}\colon \forall^{\mathsf{nc}}\vec{p}\forall f_1, f_2.\\
&\quad \forall n(F\,f_1\,f_2\,b\,\mathbf{r}\,A)[b{:=}\mathsf{Leaf}\,n] \to\\
&\quad (\forall bs.\varepsilon\,\mathbf{r}\,B \to (F\,f_1\,f_2\,b\,\mathbf{r}\,A)[b{:=}\mathsf{Branch}\,bs]) \to\\
&\quad (\varepsilon\,\mathbf{r}\,B)[bs{:=}\mathsf{Empty}] \to\\
&\quad (\forall b, bs.F\,f_1\,f_2\,b\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B \to (\varepsilon\,\mathbf{r}\,B)[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\quad \forall b(F\,f_1\,f_2\,b\,\mathbf{r}\,A)
\end{aligned}$$

with F defined by the recursion equations

$$\begin{aligned}
F\,f_1\,f_2(\mathsf{Leaf}\,n) &= f_1 n,\\
F\,f_1\,f_2(\mathsf{Branch}\,bs) &= f_2\,bs.
\end{aligned}$$

By our identification of $\beta\eta\mathcal{R}$-equivalent terms this is the same as

$$\begin{aligned}
&\mathsf{Ind}_{b,bs,(\varepsilon\,\mathbf{r}\,A),(\varepsilon\,\mathbf{r}\,B)}\colon \forall^{\mathsf{nc}}\vec{p}\forall f_1, f_2.\\
&\quad \forall n(f_1 n\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n]) \to\\
&\quad (\forall bs.\varepsilon\,\mathbf{r}\,B \to f_2\,bs\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs]) \to\\
&\quad \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}] \to\\
&\quad (\forall b, bs.F\,f_1\,f_2\,b\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B \to \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\,b\,bs]) \to\\
&\quad \forall b(F\,f_1\,f_2\,b\,\mathbf{r}\,A)
\end{aligned}$$

Let

$$\begin{aligned}
&u_1\colon \forall n(f_1 n\,\mathbf{r}\,A[b{:=}\mathsf{Leaf}\,n])\\
&u_2\colon \forall bs.\varepsilon\,\mathbf{r}\,B \to f_2 bs\,\mathbf{r}\,A[b{:=}\mathsf{Branch}\,bs]\\
&u_3\colon \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Empty}]\\
&u_4\colon \forall b, bs, y_1.y_1\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B \to \varepsilon\,\mathbf{r}\,B[bs{:=}\mathsf{Tcons}\,b\,bs]
\end{aligned}$$

be assumption variables. Then the proof term is

$$\begin{aligned}
&\lambda\vec{p}\lambda f_1\lambda u_1\lambda f_2\lambda u_2\lambda u_3\lambda u_4\lambda b.\mathsf{Ind}_{b,bs,(F f_1 f_2 b\,\mathbf{r}\,A),(\varepsilon\,\mathbf{r}\,B)}\vec{p} f_1 f_2\\
&(\lambda n.u_1 n)(\lambda bs, v_1.u_2\,bs\,v_1)u_3(\lambda b, bs, v_1^{F f_1 f_2 b\,\mathbf{r}\,A}, v_2^{\varepsilon\,\mathbf{r}\,B}.u_4\,b\,bs(F f_1 f_2 b)v_1 v_2)b
\end{aligned}$$

4.3.3. Realizing a Cases Axiom. The cases axiom Cases_{x_j,A_j} or shortly Cases_j proves the universal closure (w.r.t. $\forall^{\mathsf{nc}}$) of the formula

$$D_{i_1} \to \cdots \to D_{i_q} \to \forall x_j^{\mu_j}\,A_j.$$

where $D_{i_1} \to \cdots \to D_{i_q}$ consists of all D_i concerning constructors for μ_j. We may assume $\tau(A_j) \neq \varepsilon$, for otherwise $[\![\mathsf{Cases}_j]\!] = \varepsilon$ and we can easily find a derivation $\mu(\mathsf{Cases}_j)$ of

$$\begin{aligned}
&\varepsilon\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{p}.D_{i_1} \to \cdots \to D_{i_q} \to \forall x_j^{\mu_j}\,A_j\big)\\
&= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,\big(D_{i_1} \to \cdots \to D_{i_q} \to \forall x_j^{\mu_j}\,A_j\big)\\
&= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,D_{i_1} \to \cdots \to \varepsilon\,\mathbf{r}\,D_{i_q} \to \forall x_j^{\mu_j}\varepsilon\,\mathbf{r}\,A_j.
\end{aligned}$$

Indeed, $\mu(\mathsf{Cases}_j)$ is just $\mathsf{Cases}_{x_j,\varepsilon\,\mathbf{r}\,A_j}$.

Recall

$$[\![\mathsf{Cases}_j]\!] = \lambda f_1 \ldots \lambda f_q \lambda x[\mathtt{if}\ x\ f_1\ \ldots\ f_q].$$

Hence we must find a derivation $\mu(\mathsf{Cases}_j)$ of

$$\begin{aligned}
&\lambda f_1 \dots \lambda f_q \lambda x[\mathtt{if}\ x\ f_1\ \dots\ f_q]\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{p}.Di_1 \to \dots \to Di_q \to \forall x_j^{\mu_j}\,A_j\big)\\
&= \forall^{\mathsf{nc}}\vec{p}.\lambda f_1 \dots \lambda f_q \lambda x[\mathtt{if}\ x\ f_1\ \dots\ f_q]\,\mathbf{r}\,\big(Di_1 \to \dots \to Di_q \to \forall x_j^{\mu_j}\,A_j\big)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f_1.f_1\,\mathbf{r}\,Di_1 \to \dots \to \forall f_q.f_q\,\mathbf{r}\,Di_q \to \forall x_j^{\mu_j}[\mathtt{if}\ x_j\ f_1\ \dots\ f_q]\,\mathbf{r}\,A_j.
\end{aligned}$$

Notice that for the step formula D_{i_p} the formula $f_p\,\mathbf{r}\,D_{i_p}$ is defined to be

$$\forall y_1^{\rho_1},\dots,y_m^{\rho_m},y_{m+1}^{\vec{\sigma}_1\to\mu_{j_1}},\dots,y_{m+n}^{\vec{\sigma}_n\to\mu_{j_n}}\,f_p\vec{y}\,\mathbf{r}\,A_j[x_j{:=}\mathsf{constr}_{i_p}^{\vec{\mu}}(\vec{y})].$$

We proceed informally. Assume $\vec{p}$, f_1, $u_1\colon f_1\,\mathbf{r}\,Di_1$, …, f_q, $u_q\colon f_q\,\mathbf{r}\,D_{i_q}$; our goal is $\forall x_j^{\mu_j}[\mathtt{if}\ x_j\ f_1\ \dots\ f_q]\,\mathbf{r}\,A_j$. For the proof we use the cases axiom w.r.t. $\forall x_j^{\mu_j}[\mathtt{if}\ x_j\ f_1\ \dots\ f_q]\,\mathbf{r}\,A_j$. Hence it suffices to prove the step formulas w.r.t. the latter cases axiom, i.e. all formulas

$$\begin{aligned}
D_{i_p}^{\mathbf{r}} :=& \forall y_1^{\rho_1},\dots,y_m^{\rho_m},y_{m+1}^{\vec{\sigma}_1\to\mu_{j_1}},\dots,y_{m+n}^{\vec{\sigma}_n\to\mu_{j_n}}.\\
&[\mathtt{if}\ (\mathsf{constr}_{i_p}^{\vec{\mu}}(\vec{y}))\ f_1\ \dots\ f_q]\,\mathbf{r}\,A_j[x_j{:=}\mathsf{constr}_{i_p}^{\vec{\mu}}(\vec{y})].
\end{aligned}$$

But this follows from $[\mathtt{if}\ (\mathsf{constr}_{i_p}^{\vec{\mu}}(\vec{y}))\ f_1\ \dots\ f_q] \mapsto f_p\vec{y}$. Hence the proof term is

$$\lambda\vec{p}\lambda f_1\lambda u_1\dots\lambda f_q\lambda u_q\mathsf{Cases}_{x_j,[\mathtt{if}\ x_j\ f_1\ \dots\ f_q]\,\mathbf{r}\,A_j}\vec{q}u_1\dots u_q.$$

with $\vec{q}$ the parameters of $\mathsf{Cases}_{x_j,[\mathtt{if}\ x_j\ f_1\ \dots\ f_q]\,\mathbf{r}\,A_j}$.

4.3.4. Existence Introduction and Elimination. Consider an existence introduction axiom

$$\exists^+_{x,A}\colon \forall^{\mathsf{nc}}\vec{p}\forall x.A \to \exists x^\rho A.$$

We must find a derivation $\mu(\exists^+_{x,A})$ of $[\![\exists^+_{x,A}]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}\forall x.A \to \exists x^\rho A$.

Case $\tau(A)=\varepsilon$. Then $[\![\exists^+_{x,A}]\!] = \lambda xx$ and we obtain

$$\begin{aligned}
(\lambda xx)\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}\forall x.A \to \exists xA &= \forall^{\mathsf{nc}}\vec{p}\forall x.x\,\mathbf{r}\,(A \to \exists xA)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to x\,\mathbf{r}\,\exists xA\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,A.
\end{aligned}$$

Hence we can define $\mu(\exists^+_{x,A}) = \lambda\vec{p}\lambda x\lambda uu$.

Case $\tau(A)\neq\varepsilon$. Then $[\![\exists^+_{x,A}]\!] = \lambda x\lambda f\langle x,f\rangle$ and we obtain

$$\begin{aligned}
&\lambda x\lambda f\langle x,f\rangle\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}\forall x.A \to \exists xA\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x,f.f\,\mathbf{r}\,A \to \langle x,f\rangle\,\mathbf{r}\,\exists xA\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x,f.f\,\mathbf{r}\,A \to f\,\mathbf{r}\,A
\end{aligned}$$

Hence we can define $\mu(\exists^+_{x,A}) = \lambda\vec{p}\lambda x\lambda f\lambda uu$.

Consider now the existence elimination axioms

$$\exists^-_{x,A,B}\colon \forall^{\mathsf{nc}}\vec{p}.\exists xA \to (\forall x.A \to B) \to B \quad \text{with } x \notin \mathsf{FV}(B).$$

We must find a derivation $\mu(\exists^-_{x,A,B})$ of

$$[\![\exists^-_{x,A,B}]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists xA \to (\forall x.A \to B) \to B.$$

Case $\tau(A) = \varepsilon = \tau(B)$. Then $[\![\exists^-_{x,A,B}]\!] = \varepsilon$ and we obtain

$$\begin{aligned}
&\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists xA \to (\forall x.A \to B) \to B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.x\,\mathbf{r}\,\exists xA \to \varepsilon\,\mathbf{r}\,((\forall x.A \to B) \to B)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,(\forall x.A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to \forall x\,\varepsilon\,\mathbf{r}\,(A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to (\forall x.\varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B) \to \varepsilon\,\mathbf{r}\,B.
\end{aligned}$$

Hence we can define $\mu(\exists^-_{x,A,B}) = \lambda\vec{p}\lambda x\lambda u\lambda v.vxu$.

Case $\tau(A) \neq \varepsilon = \tau(B)$. Then we again have $[\![\exists^-_{x,A,B}]\!] = \varepsilon$ and we obtain

$$\begin{aligned}
&\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists xA \to (\forall x.A \to B) \to B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.f\,\mathbf{r}\,\exists xA \to \varepsilon\,\mathbf{r}\,((\forall x.A \to B) \to B)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.(f1)\,\mathbf{r}\,A[x{:=}f0] \to \varepsilon\,\mathbf{r}\,(\forall x.A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.(f1)\,\mathbf{r}\,A[x{:=}f0] \to \forall x\,\varepsilon\,\mathbf{r}\,(A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.(f1)\,\mathbf{r}\,A[x{:=}f0] \to (\forall x\forall g.g\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B) \to \varepsilon\,\mathbf{r}\,B.
\end{aligned}$$

Hence we can define $\mu(\exists^-_{x,A,B}) = \lambda\vec{p}\lambda f\lambda u\lambda v.v(f0)(f1)u$.

Case $\tau(A) = \varepsilon \neq \tau(B)$. Then $[\![\exists^-_{x,A,B}]\!] = \lambda x\lambda z(zx)$ and we obtain

$$\begin{aligned}
&\lambda x\lambda z(zx)\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists xA \to (\forall x.A \to B) \to B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.x\,\mathbf{r}\,\exists xA \to \lambda z(zx)\,\mathbf{r}\,((\forall x.A \to B) \to B)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to \forall z.z\,\mathbf{r}\,(\forall x.A \to B) \to zx\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to \forall z.\forall x\,zx\,\mathbf{r}\,(A \to B) \to zx\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall x.\varepsilon\,\mathbf{r}\,A \to \forall z.(\forall x.\varepsilon\,\mathbf{r}\,A \to zx\,\mathbf{r}\,B) \to zx\,\mathbf{r}\,B.
\end{aligned}$$

Hence we can define $\mu(\exists^-_{x,A,B}) = \lambda\vec{p}\lambda x\lambda u\lambda z\lambda v.vxu$.

Case $\tau(A) \neq \varepsilon \neq \tau(B)$. Then $[\![\exists^-_{x,A,B}]\!] = \lambda f\lambda z(z(f0)(f1))$ and we obtain

$$\begin{aligned}
&\lambda f\lambda z(z(f0)(f1))\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists xA \to (\forall x.A \to B) \to B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.f\,\mathbf{r}\,\exists xA \to \lambda z(z(f0)(f1))\,\mathbf{r}\,((\forall x.A \to B) \to B)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.(f1)\,\mathbf{r}\,A[x{:=}f0] \to\\
&\qquad \forall z.z\,\mathbf{r}\,(\forall x.A \to B) \to z(f0)(f1)\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall f.(f1)\,\mathbf{r}\,A[x{:=}f0] \to\\
&\qquad \forall z.(\forall x,g.g\,\mathbf{r}\,A \to zxg\,\mathbf{r}\,B) \to z(f0)(f1)\,\mathbf{r}\,B.
\end{aligned}$$

Hence we can define $\mu(\exists^-_{x,A,B}) = \lambda\vec{p}\lambda f\lambda u\lambda z\lambda v.v(f0)(f1)u$.

The treatment of the introduction and elimination axioms for the quantifiers $\forall^{\mathsf{nc}}, \exists^{\mathsf{nc}}$ is somewhat different: we will need other instances of these axioms in our derivations. Consider an existence introduction axiom

$$(\exists^{\mathsf{nc}})^+_{x,A}\colon \forall^{\mathsf{nc}}\vec{p},x.A \to \exists^{\mathsf{nc}}x^\rho A.$$

We must find a derivation $\mu((\exists^{\mathsf{nc}})^+_{x,A})$ of $[\![(\exists^{\mathsf{nc}})^+_{x,A}]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p},x.A \to \exists^{\mathsf{nc}}x^\rho A$.

Case $\tau(A) = \varepsilon$. Then $[\![(\exists^{\mathsf{nc}})^+_{x,A}]\!] = \varepsilon$ and we obtain

$$\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p},x.A \to \exists^{\mathsf{nc}}xA = \forall^{\mathsf{nc}}\vec{p},x.\varepsilon\,\mathbf{r}\,(A \to \exists^{\mathsf{nc}}xA)$$

$$
\begin{aligned}
&= \forall^{\mathsf{nc}}\vec{p}, x.\varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,\exists^{\mathsf{nc}}xA\\
&= \forall^{\mathsf{nc}}\vec{p}, x.\varepsilon\,\mathbf{r}\,A \to \exists^{\mathsf{nc}}x.\varepsilon\,\mathbf{r}\,A.
\end{aligned}
$$

Hence we can define $\mu((\exists^{\mathsf{nc}})^+_{x,A}) = (\exists^{\mathsf{nc}})^+_{x,\varepsilon\,\mathbf{r}\,A}$.

Case $\tau(A) \neq \varepsilon$. Then $[\![(\exists^{\mathsf{nc}})^+_{x,A}]\!] = \lambda yy$ and we obtain

$$
\begin{aligned}
&\lambda yy\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}, x.A \to \exists^{\mathsf{nc}}xA\\
&= \forall^{\mathsf{nc}}\vec{p}, x.\lambda yy\,\mathbf{r}\,(A \to \exists^{\mathsf{nc}}xA)\\
&= \forall^{\mathsf{nc}}\vec{p}, x\forall y.y\,\mathbf{r}\,A \to y\,\mathbf{r}\,\exists^{\mathsf{nc}}xA\\
&= \forall^{\mathsf{nc}}\vec{p}, x\forall y.y\,\mathbf{r}\,A \to \exists^{\mathsf{nc}}x.y\,\mathbf{r}\,A
\end{aligned}
$$

Now observe that $\tau(y\,\mathbf{r}\,A) = \varepsilon$, hence we can equivalently replace $\forall y$ by $\forall^{\mathsf{nc}}y$. Therefore we can define $\mu((\exists^{\mathsf{nc}})^+_{x,A}) = \lambda^{\mathsf{nc}}\vec{p}, x, y.(\exists^{\mathsf{nc}})^+_{x,y\,\mathbf{r}\,A}\vec{q}x$ with $\vec{q}$ the free variables of $y\,\mathbf{r}\,A$.

Consider now the existence elimination axioms

$$
(\exists^{\mathsf{nc}})^-_{x,A,B}\colon \forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}xA \to (\forall^{\mathsf{nc}}x.A \to B) \to B \quad \text{with } x \notin \mathsf{FV}(B).
$$

We must find a derivation $\mu((\exists^{\mathsf{nc}})^-_{x,A,B})$ of

$$
[\![(\exists^{\mathsf{nc}})^-_{x,A,B}]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}xA \to (\forall^{\mathsf{nc}}x.A \to B) \to B.
$$

Case $\tau(A) = \varepsilon = \tau(B)$. Then $[\![(\exists^{\mathsf{nc}})^-_{x,A,B}]\!] = \varepsilon$ and we obtain

$$
\begin{aligned}
&\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}xA \to (\forall^{\mathsf{nc}}x.A \to B) \to B\\
&= \forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,\exists^{\mathsf{nc}}xA \to \varepsilon\,\mathbf{r}\,((\forall^{\mathsf{nc}}x.A \to B) \to B)\\
&= \forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,(\forall^{\mathsf{nc}}x.A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A \to \forall^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,(A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A \to (\forall^{\mathsf{nc}}x.\varepsilon\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B) \to \varepsilon\,\mathbf{r}\,B.
\end{aligned}
$$

Hence we can define $\mu((\exists^{\mathsf{nc}})^-_{x,A,B}) = (\exists^{\mathsf{nc}})^-_{x,\varepsilon\,\mathbf{r}\,A,\varepsilon\,\mathbf{r}\,B}$. [Todo: check the next three cases]

Case $\tau(A) \neq \varepsilon = \tau(B)$. Then again $[\![(\exists^{\mathsf{nc}})^-_{x,A,B}]\!] = \varepsilon$ and we obtain

$$
\begin{aligned}
&\varepsilon\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}xA \to (\forall^{\mathsf{nc}}x.A \to B) \to B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall y.y\,\mathbf{r}\,\exists^{\mathsf{nc}}xA \to \varepsilon\,\mathbf{r}\,((\forall^{\mathsf{nc}}x.A \to B) \to B)\\
&= \forall^{\mathsf{nc}}\vec{p}\forall y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,(\forall^{\mathsf{nc}}x.A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A \to \forall^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,(A \to B) \to \varepsilon\,\mathbf{r}\,B\\
&= \forall^{\mathsf{nc}}\vec{p}\forall y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A \to (\forall^{\mathsf{nc}}x\forall y.y\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B) \to \varepsilon\,\mathbf{r}\,B.
\end{aligned}
$$

Here again we can equivalently replace $\forall y$ by $\forall^{\mathsf{nc}}y$, and then use the general fact that $\vdash (\forall^{\mathsf{nc}}x.A \to B) \leftrightarrow (\exists^{\mathsf{nc}}xA \to B)$. So our claim is equivalent to

$$
\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A \to (\forall^{\mathsf{nc}}x.\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A \to \varepsilon\,\mathbf{r}\,B) \to \varepsilon\,\mathbf{r}\,B,
$$

which is the axiom $(\exists^{\mathsf{nc}})^-_{x,\exists^{\mathsf{nc}}y\,y\,\mathbf{r}\,A,\varepsilon\,\mathbf{r}\,B}$. To spell out the derivation term, let $\vec{p} = \vec{q_1}, \vec{q_2}$ with $\vec{q_1}$ the variables free in B, $\vec{q_2}$ those free in A, and $\vec{q_3}$ those

free in A with x removed. The derivation term is

$$\lambda^{\mathsf{nc}}\vec{p}\lambda y\lambda u^{\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A}\lambda v^{\forall^{\mathsf{nc}}x\forall y.y\,\mathbf{r}\,A\to\varepsilon\,\mathbf{r}\,B}.(\exists^{\mathsf{nc}})^-_{x,\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A,\varepsilon\,\mathbf{r}\,B}M_1M_2 \quad \text{with}$$

$$M_1^{\exists^{\mathsf{nc}}x\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A} := (\exists^{\mathsf{nc}})^-_{x,y\,\mathbf{r}\,A,\exists^{\mathsf{nc}}x,y\ y\,\mathbf{r}\,A}\vec{q_3}yu$$
$$(\lambda^{\mathsf{nc}}x\lambda y\lambda u_1^{y\,\mathbf{r}\,A}.(\exists^{\mathsf{nc}})^+_{x.\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A}\vec{q_2}((\exists^{\mathsf{nc}})^+_{y,y\,\mathbf{r}\,A}\vec{q_2}yu_1))$$

$$M_2^{\forall^{\mathsf{nc}}x.\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A\to\varepsilon\,\mathbf{r}\,B} := \lambda^{\mathsf{nc}}x\lambda u_2^{\exists^{\mathsf{nc}}y\ y\,\mathbf{r}\,A}.(\exists^{\mathsf{nc}})^-_{y,y\,\mathbf{r}\,A,\varepsilon\,\mathbf{r}\,B}\vec{p}u_2(\lambda^{\mathsf{nc}}y.vxy)$$

Case $\tau(A)=\varepsilon\neq\tau(B)$. Then $[\![(\exists^{\mathsf{nc}})^-_{x,A,B}]\!]=\lambda zz$ and we obtain

$$\begin{aligned}
&\lambda zz\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}xA\to(\forall^{\mathsf{nc}}x.A\to B)\to B\\
&=\forall^{\mathsf{nc}}\vec{p}.\varepsilon\,\mathbf{r}\,\exists^{\mathsf{nc}}xA\to\lambda zz\,\mathbf{r}\,((\forall^{\mathsf{nc}}x.A\to B)\to B)\\
&=\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A\to\forall z.z\,\mathbf{r}\,(\forall^{\mathsf{nc}}x.A\to B)\to z\,\mathbf{r}\,B\\
&=\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A\to\forall z.\forall^{\mathsf{nc}}x\,z\,\mathbf{r}\,(A\to B)\to z\,\mathbf{r}\,B\\
&=\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A\to\forall z.(\forall^{\mathsf{nc}}x.\varepsilon\,\mathbf{r}\,A\to z\,\mathbf{r}\,B)\to z\,\mathbf{r}\,B.
\end{aligned}$$

This is equivalent to

$$\forall^{\mathsf{nc}}\vec{p},z.\exists^{\mathsf{nc}}x\ \varepsilon\,\mathbf{r}\,A\to(\forall^{\mathsf{nc}}x.\varepsilon\,\mathbf{r}\,A\to z\,\mathbf{r}\,B)\to z\,\mathbf{r}\,B,$$

which is the axiom $(\exists^{\mathsf{nc}})^-_{x,\varepsilon\,\mathbf{r}\,A,z\,\mathbf{r}\,B}$.

Case $\tau(A)\neq\varepsilon\neq\tau(B)$. Then $[\![(\exists^{\mathsf{nc}})^-_{x,A,B}]\!]=\lambda y\lambda z.zy$ and we obtain

$$\begin{aligned}
&(\lambda y\lambda z.zy)\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}.\exists^{\mathsf{nc}}xA\to(\forall^{\mathsf{nc}}x.A\to B)\to B\\
&=\forall^{\mathsf{nc}}\vec{p}\forall y.y\,\mathbf{r}\,\exists^{\mathsf{nc}}xA\to\lambda z.zy\,\mathbf{r}\,((\forall^{\mathsf{nc}}x.A\to B)\to B)\\
&=\forall^{\mathsf{nc}}\vec{p}\forall y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A\to\forall z.z\,\mathbf{r}\,(\forall^{\mathsf{nc}}x.A\to B)\to zy\,\mathbf{r}\,B\\
&=\forall^{\mathsf{nc}}\vec{p}\forall y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A\to\forall z.(\forall^{\mathsf{nc}}x\forall y_1.y_1\,\mathbf{r}\,A\to zy_1\,\mathbf{r}\,B)\to zy\,\mathbf{r}\,B.
\end{aligned}$$

This follows from

$$\forall^{\mathsf{nc}}\vec{p},z,y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A\to(\forall y.\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A\to zy\,\mathbf{r}\,B)\to zy\,\mathbf{r}\,B,$$

which is the axiom $(\exists^{\mathsf{nc}})^-_{y,\exists^{\mathsf{nc}}x\ y\,\mathbf{r}\,A,zy\,\mathbf{r}\,B}$.

4.3.5. Compatibility. Consider a compatibility axiom

$$\mathsf{Eq\text{-}Compat}_{x_1,A}\colon\forall^{\mathsf{nc}}\vec{p}\forall x_1,x_2.x_1\approx x_2\to A\to A[x_1{:=}x_2].$$

We must find a derivation $\mu(\mathsf{Eq\text{-}Compat}_{x_1,A})$ of

$$[\![\mathsf{Eq\text{-}Compat}_{x_1,A}]\!]\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{p}\forall x_1,x_2.x_1\approx x_2\to A\to A[x_1{:=}x_2]\big).$$

Case $\tau(A)=\varepsilon$. Then $[\![\mathsf{Eq\text{-}Compat}_{x_1,A}]\!]=\varepsilon$ and we clearly can define $\mu(\mathsf{Eq\text{-}Compat}_{x_1,A}):=\mathsf{Eq\text{-}Compat}_{x_1,\varepsilon\,\mathbf{r}\,A}$.

Case $\tau(A)\neq\varepsilon$. Then $[\![\mathsf{Eq\text{-}Compat}_{x_1,A}]\!]=\lambda x_1\lambda x_2\lambda f\,f$ and we obtain

$$\begin{aligned}
&\lambda x_1\lambda x_2\lambda f\,f\,\mathbf{r}\,\forall^{\mathsf{nc}}\vec{p}\forall x_1,x_2.x_1\approx x_2\to A\to A[x_1{:=}x_2]\\
&=\forall^{\mathsf{nc}}\vec{p}\forall x_1,x_2.x_1\approx x_2\to\forall f.f\,\mathbf{r}\,A\to f\,\mathbf{r}\,A[x_1{:=}x_2].
\end{aligned}$$

Hence we can define

$$\mu(\mathsf{Eq\text{-}Compat}_{x_1,A}):=\lambda\vec{p}\lambda x_1\lambda x_2\lambda u^{x_1\approx x_2}\lambda f.\mathsf{Eq\text{-}Compat}_{x_1,(f\,\mathbf{r}\,A)}f\vec{p}x_1x_2u.$$

4.3.6. The Soundness Theorem. Recall that $x_u := \varepsilon$ if u^A is an assumption variable with a Harrop-formula A.

THEOREM (Soundness). *If M is a derivation of a formula B, then there is a derivation $\mu(M)$ of $[\![M]\!]\,\mathbf{r}\,B$ from $\{\, x_u\,\mathbf{r}\,C \mid u^C \in \mathsf{FA}(M)\,\}$.*

PROOF. Induction on M.

Case $u\colon A$. Then $\bar{u}\colon x_u\,\mathbf{r}\,A$. Let $\mu(u) := \bar{u}$.

Case $c\colon A$, c an axiom. These cases have been treated above.

Case $\lambda u^A M^B$. We must find a derivation $\mu(\lambda uM)$ of

$$[\![\lambda uM]\!]\,\mathbf{r}\,(A \to B).$$

Subcase $\tau(A) = \varepsilon$. Then $[\![\lambda uM]\!] = [\![M]\!]$, hence

$$[\![\lambda uM]\!]\,\mathbf{r}\,(A \to B) \;=\; \varepsilon\,\mathbf{r}\,A \;\to\; [\![M]\!]\,\mathbf{r}\,B.$$

By IH we can define $\mu(\lambda uM) := \lambda\bar{u}\mu(M)$ with $\bar{u}\colon \varepsilon\,\mathbf{r}\,A$.

Subcase $\tau(A) \neq \varepsilon = \tau(B)$. Then $[\![\lambda uM]\!] = \varepsilon$ and

$$[\![\lambda uM]\!]\,\mathbf{r}\,(A \to B) \;=\; \forall x.x\,\mathbf{r}\,A \;\to\; \varepsilon\,\mathbf{r}\,B,$$

and by IH we can define $\mu(\lambda uM) := \lambda x_u\lambda\bar{u}\mu(M)$ with $\bar{u}\colon x_u\,\mathbf{r}\,A$.

Subcase $\tau(A) \neq \varepsilon \neq \tau(B)$. Then

$$[\![\lambda uM]\!]\,\mathbf{r}\,(A \to B) \;=\; \forall x.x\,\mathbf{r}\,A \;\to\; [\![\lambda uM]\!]x\,\mathbf{r}\,B$$

Because of $[\![\lambda uM]\!] = \lambda x_u[\![M]\!]$ and since we identify terms with the same β-normal form, again by IH we can define $\mu(\lambda uM) := \lambda x_u\lambda\bar{u}\mu(M)$.

Case $M^{A\to B}N^A$. We must find a derivation $\mu(MN)$ of $[\![MN]\!]\,\mathbf{r}\,B$.

Subcase $\tau(A) = \varepsilon$. Then $[\![MN]\!] = [\![M]\!]$. By IH we have derivations $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,(A \to B) \;=\; \varepsilon\,\mathbf{r}\,A \;\to\; [\![M]\!]\,\mathbf{r}\,B$$

and $\mu(N)$ of $\varepsilon\,\mathbf{r}\,A$; hence we can define $\mu(MN) := \mu(M)\mu(N)$.

Subcase $\tau(A) \neq \varepsilon = \tau(B)$. Then $[\![MN]\!] = \varepsilon$. By IH we have derivations $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,(A \to B) \;=\; \forall x.x\,\mathbf{r}\,A \;\to\; \varepsilon\,\mathbf{r}\,B$$

and $\mu(N)$ of $[\![N]\!]\,\mathbf{r}\,A$; hence we can define $\mu(MN) := \mu(M)[\![N]\!]\mu(N)$.

Subcase $\tau(A) \neq \varepsilon \neq \tau(B)$. Then $[\![MN]\!] = [\![M]\!][\![N]\!]$. By induction hypothesis we have derivations $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,(A \to B) \;=\; \forall x.x\,\mathbf{r}\,A \;\to\; [\![M]\!]x\,\mathbf{r}\,B$$

and $\mu(N)$ of $[\![N]\!]\,\mathbf{r}\,A$; hence we can define $\mu(MN) := \mu(M)[\![N]\!]\mu(N)$.

Case $\langle M_0^{A_0}, M_1^{A_1}\rangle$. We must find a derivation $\mu(\langle M_0, M_1\rangle)$ of

$$[\![\langle M_0, M_1\rangle]\!]\,\mathbf{r}\,(A_0 \wedge A_1).$$

Subcase $\tau(A_0) = \varepsilon = \tau(A_1)$. Then $[\![\langle M_0, M_1\rangle]\!] = \varepsilon$, hence

$$[\![\langle M_0, M_1\rangle]\!]\,\mathbf{r}\,(A_0 \wedge A_1) \;=\; \varepsilon\,\mathbf{r}\,A_0 \;\wedge\; \varepsilon\,\mathbf{r}\,A_1$$

and by IH we can define $\mu(\langle M_0, M_1\rangle) := \langle\mu(M_0), \mu(M_1)\rangle$.

Subcase $\tau(A_0) = \varepsilon \neq \tau(A_1)$. Then $[\![\langle M_0, M_1\rangle]\!] = [\![M_1]\!]$, hence

$$[\![\langle M_0, M_1\rangle]\!]\,\mathbf{r}\,(A_0 \wedge A_1) \;=\; \varepsilon\,\mathbf{r}\,A_0 \;\wedge\; [\![M_1]\!]\,\mathbf{r}\,A_1$$

and by IH we can define $\mu(\langle M_0, M_1\rangle) := \langle\mu(M_0), \mu(M_1)\rangle$.

Subcase $\tau(A_0) \neq \varepsilon = \tau(A_1)$. Similar.

Subcase $\tau(A_0) \neq \varepsilon \neq \tau(A_1)$. Then $[\![\langle M_0, M_1\rangle]\!] = \langle [\![M_0]\!], [\![M_1]\!]\rangle$, hence

$$[\![\langle M_0, M_1\rangle]\!]\,\mathbf{r}\,(A_0 \wedge A_1) \;=\; [\![M_0]\!]\,\mathbf{r}\,A_0 \;\wedge\; [\![M_1]\!]\,\mathbf{r}\,A_1$$

and by IH we can define $\mu(\langle M_0, M_1\rangle) := \langle \mu(M_0), \mu(M_1)\rangle$.

Case $(M^{A_0 \wedge A_1} 0)$. We must find a derivation $\mu(M0)$ of

$$[\![M0]\!]\,\mathbf{r}\,A_0.$$

Subcase $\tau(A_1) = \varepsilon$. Then $[\![M0]\!] = [\![M]\!]$. By IH we have a derivation $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,(A_0 \wedge A_1) \;=\; [\![M]\!]\,\mathbf{r}\,A_0 \;\wedge\; \varepsilon\,\mathbf{r}\,A_1.$$

hence we can define $\mu(M0) := \mu(M)0$.

Subcase $\tau(A_0) = \varepsilon \neq \tau(A_1)$. Then $[\![M0]\!] = \varepsilon$. By IH we have a derivation $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,(A_0 \wedge A_1) \;=\; \varepsilon\,\mathbf{r}\,A_0 \;\wedge\; [\![M]\!]\,\mathbf{r}\,A_1.$$

hence we can define $\mu(M0) := \mu(M)0$).

Subcase $\tau(A_0) \neq \varepsilon \neq \tau(A_1)$. Similar; we can define $\mu(\langle M_0, M_1\rangle) := \langle \mu(M_0), \mu(M_1)\rangle$.

Case $M^{A_0 \wedge A_1} 1$. Similar.

Case $\lambda z M^A$. We must find a derivation $\mu(\lambda z M)$ of $[\![\lambda z M]\!]\,\mathbf{r}\,\forall z A$. By definition $[\![\lambda z M]\!] = \lambda z [\![M]\!]$.

Subcase $\tau(A) = \varepsilon$. Then

$$\lambda z [\![M]\!]\,\mathbf{r}\,\forall z A \;=\; \forall z.\varepsilon\,\mathbf{r}\,A$$

and by IH we can define $\mu(\lambda z M) := \lambda z \mu(M)$. The variable condition is satisfied, since $\lambda z M^A$ is a derivation term, and hence z does not occur free in any assumption variable $u\colon B$ free in M^A, hence also does not occur free in the free assumption $\bar{u}\colon x_u\,\mathbf{r}\,B$.

Subcase $\tau(A) \neq \varepsilon$. Then

$$\lambda z [\![M]\!]\,\mathbf{r}\,\forall z A \;=\; \forall z.(\lambda z [\![M]\!]) z\,\mathbf{r}\,A.$$

Since we identify terms with the same β-normal form, by IH we again can define $\mu(\lambda z M) := \lambda z \mu(M)$. As before one can see that the variable condition is satisfied.

Case $M^{\forall z A} t$. We must find a derivation $\mu(Mt)$ of $[\![Mt]\!]\,\mathbf{r}\,A[z{:=}t]$. By definition we have $[\![Mt]\!] = [\![M]\!]t$.

Subcase $\tau(A) = \varepsilon$. By IH we have a derivation of

$$[\![M]\!]\,\mathbf{r}\,\forall z A \;=\; \forall z.\varepsilon\,\mathbf{r}\,A$$

hence we can define $\mu(Mt) := \mu(M)t$.

Subcase $\tau(A) \neq \varepsilon$. By IH we have a derivation of

$$[\![M]\!]\,\mathbf{r}\,\forall z A \;=\; \forall z.[\![M]\!] z\,\mathbf{r}\,A,$$

hence we again can define $\mu(Mt) := \mu(M)t$.

Case $(\lambda z M)^{\forall^{\mathsf{nc}} z A}$. We must find a derivation $\mu(\lambda z M)$ of $[\![\lambda z M]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}} z A$. By definition $[\![(\lambda z M)^{\forall^{\mathsf{nc}} z A}]\!] = [\![M]\!]$.

Subcase $\tau(A) = \varepsilon$. Then

$$[\![M]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}} z A \;=\; \forall z.\varepsilon\,\mathbf{r}\,A$$

and by IH we can define $\mu((\lambda z M)^{\forall^{\mathsf{nc}} z A}) := \lambda z \mu(M)$. The variable condition is satisfied, since $(\lambda z M)^{\forall^{\mathsf{nc}} z A}$ is a derivation term, and hence z does not

occur free in any assumption variable $u\colon B$ free in M^A, hence also does not occur free in the free assumption $\bar{u}\colon x_u\,\mathbf{r}\,B$.

Subcase $\tau(A) \neq \varepsilon$. Then

$$[\![M]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}zA \;=\; \forall z.[\![M]\!]\,\mathbf{r}\,A.$$

By IH we again can define $\mu((\lambda zM)^{\forall^{\mathsf{nc}}zA}) := \lambda z\mu(M)$. As before one can see that the variable condition is satisfied.

Case $M^{\forall^{\mathsf{nc}}zA}t$. We must find a derivation $\mu(Mt)$ of $[\![Mt]\!]\,\mathbf{r}\,A[z{:=}t]$. By definition we have $[\![Mt]\!] = [\![M]\!]$.

Subcase $\tau(A) = \varepsilon$. By IH we have a derivation $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}zA \;=\; \forall z.\varepsilon\,\mathbf{r}\,A$$

hence we can define $\mu(Mt) := \mu(M)t$.

Subcase $\tau(A) \neq \varepsilon$. By IH we have a derivation $\mu(M)$ of

$$[\![M]\!]\,\mathbf{r}\,\forall^{\mathsf{nc}}zA \;=\; \forall z.[\![M]\!]\,\mathbf{r}\,A.$$

Define $\mu(Mt) := \mu(M)t$, since we can assume that $z \notin \mathsf{FV}([\![M]\!])$. □

If B is $\exists$-free, then $\varepsilon\,\mathbf{r}\,B = B$. Hence for $\forall x^\rho \exists y^\sigma B$ with $\exists$-free B we have $\tau(\forall x \exists y B) = \rho \to \sigma$ and

$$t\,\mathbf{r}\,\forall x\exists y B = \forall x B[y{:=}tx].$$

Then as a corollary to the soundness theorem we obtain the extraction theorem

Corollary (Extraction). *From a derivation* $M\colon \forall x^\rho \exists y^\sigma B$ *with* B $\exists$*-free from* $\exists$*-free assumptions* Γ *one can extract a closed term* $[\![M]\!]^{\rho\to\sigma}$ *such that the formula* $\forall x B[y{:=}[\![M]\!]x]$ *is provable from* Γ. □

Here Γ should be viewed as lemmata, i.e. true formulas (proved separately, to keep M short). The theorem says that the extracted program is independent of how this shortcut is achieved.

4.4. Case Studies

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

$$\begin{aligned}\alpha_0 &:= 0\\ \alpha_1 &:= 1\\ \alpha_n &:= \alpha_{n-2} + \alpha_{n-1} \quad \text{for } n \geq 2\end{aligned}$$

Here is a naive Scheme-program to compute them

```
(define (f n)
  (if (= n 0)
      0
      (if (= 1 n)
          1
          (+ (f (- n 2)) (f (- n 1))))))
```

Clearly this program is very inefficient, since it leads to many recomputations. Our goal here is to demonstrate that deriving a program from a constructive proof of the existence of Fibonacci numbers avoids the immediate source of the inefficiency.

We first build a constructive existence proof for the Fibonacci numbers.

```
(set-goal
  (pf "G 0 0 -> G 1 1 ->
       (all n,k,l.G n k -> G(n+1)l -> G(n+2)(k+l)) ->
       all n ex k,l. G n k & G(n+1)l"))
(assume "Init-Zero" "Init-One" "Step")
(ind)

; Base
(ex-intro (pt "0"))
(ex-intro (pt "1"))
(prop)

; Step
(assume "n" "IH")
(by-assume-with "IH" "k" "IH-k")
(by-assume-with "IH-k" "l" "IH-l")
(ex-intro (pt "l"))
(ex-intro (pt "k+l"))
(search)
```

The extracted program can now be obtained as follows.

```
(define Fib-neterm
  (nt (proof-to-extracted-term
    (theorem-name-to-proof "Fib"))))
```

The extracted term is obtained by

```
(term-to-string Fib-neterm)
```

which yields

```
(Rec nat=>nat@@nat)(0@1)([n1,p2]right p2@left p2+right p2)
```

This clearly is a linear algorithm. To run it, type

```
(pp (nt (make-term-in-app-form Fib-neterm (pt "13"))))
```

which yields

```
233@377
```

We can also use an "external" extraction, yielding Scheme code:

```
(term-to-expr Fib-neterm)
```

produces

```
((natrec (cons 0 1))
 (lambda (n1)
   (lambda (p2) (cons (cdr p2) (+ (car p2) (cdr p2))))))
```

To run this code, we need to give a Scheme-definition of `natrec`, that is recursion on the natural numbers:

```
(define (natrec init)
  (lambda (step)
    (lambda (n)
      (if (= 0 n)
          init
          ((step n) (((natrec init) step) (- n 1)))))))
```

This again is a linear algorithm. We will see later that a classical proof yields a linear algorithm as well, which however uses functions instead of pairs.

REMARK. There are other algorithms to compute the Fibonacci numbers α_n, which run in logarithmic time. By definition we have

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \alpha_{n-2} \\ \alpha_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha_{n-1} \\ \alpha_{n-2} + \alpha_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha_{n-1} \\ \alpha_n \end{pmatrix},$$

hence with $A = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right) \in \mathbb{R}^{2\times 2}$

$$A^n \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_n \\ \alpha_{n+1} \end{pmatrix}.$$

So an easy way to compute the Fibonacci numbers is by computing the powers of A. This can be done in time $O(\log(n))$, since

$$A^{2n} = (A^n)^2$$
$$A^{2n+1} = A^{2n} \cdot A$$

It is possible to obtain this algorithm as computational content of a proof: Use u, v to denote vectors in $\mathbb{Z}^2$ and X, Y to denote matrices in $\mathbb{Z}^{2\times 2}$. Let $G(n, u)$ mean that u is the vector of the n-th and $(n+1)$-th Fibonacci number. G can be axiomatized by

$$G(1, \begin{pmatrix} 1 \\ 1 \end{pmatrix}), \quad G(n, \begin{pmatrix} \alpha \\ \beta \end{pmatrix}) \to G(n+1, \begin{pmatrix} \beta \\ \alpha + \beta \end{pmatrix}).$$

Then prove by induction on the positive (binary) numbers n

$$\forall n \exists X \forall m, u. G(m, u) \to G(m + n, Xu).$$

Clearly this X must be A^n.

Using some linear algebra, one can even give an explicit formula for α_n. To this end we first diagonalize the matrix A. The general recipe runs as follows. Form $B := A^t$, hence $B = A$ in our case. The eigenvalues can be computed as the zeros of the characteristic polynomial $p_A = |A - tE| = -t(1-t) - 1 = t^2 - t - 1$. So the eigenvalues are $\lambda_{1,2} = \frac{1\pm\sqrt{5}}{2}$. We now compute eigenvectors for these eigenvalues.

For $\lambda_1 = \frac{1+\sqrt{5}}{2}$. Let $x = \left(\begin{smallmatrix} \xi_1 \\ \xi_2 \end{smallmatrix}\right)$, and solve the linear equation system

$$\begin{pmatrix} -\lambda_1 & 1 \\ 1 & 1 - \lambda_1 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

So $-\lambda_1\xi_1 + \xi_2 = 0$. For $\xi_1 = 1$ we obtain $\xi_2 = \lambda_1$, so $x_1 = \left(\begin{smallmatrix} 1 \\ \lambda_1 \end{smallmatrix}\right)$ is an eigenvector. For $\lambda_2 = \frac{1-\sqrt{5}}{2}$ we similarly obtain $x_2 = \left(\begin{smallmatrix} 1 \\ \lambda_2 \end{smallmatrix}\right)$ as an eigenvector.

For the dimensions of the eigenspaces and the multiplicities of the eigenvalues we obtain

$$\dim(\mathsf{Eig}(f_B, \lambda_1)) = 1 = \mu(p_B, \lambda_1)$$
$$\dim(\mathsf{Eig}(f_B, \lambda_2)) = 1 = \mu(p_B, \lambda_2),$$

so the matrix B is diagonalizable, and $\binom{1}{\lambda_1}, \binom{1}{\lambda_2}$ is a basis of $\mathbb{R}^2$ consisting of eigenvectors of f_B. Let

$$T := \begin{pmatrix} 1 & 1 \\ \lambda_1 & \lambda_2 \end{pmatrix}, \qquad S := T^t = \begin{pmatrix} 1 & \lambda_1 \\ 1 & \lambda_2 \end{pmatrix}.$$

Then the general theory yields

$$SAS^{-1} = D := \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \qquad S^{-1} = \frac{1}{\lambda_2 - \lambda_1} \begin{pmatrix} \lambda_2 & -\lambda_1 \\ -1 & 1 \end{pmatrix}.$$

We can now give an explicit formula for A^n and hence also for the Fibonacci numbers.

$$\begin{aligned}
A^n &= (S^{-1}DS)^n \\
&= \underbrace{(S^{-1}DS) \cdot \ldots \cdot (S^{-1}DS)}_{n \text{ times}} \\
&= S^{-1}D^nS \\
&= S^{-1} \begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix} S \\
&= S^{-1} \begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix} \begin{pmatrix} 1 & \lambda_1 \\ 1 & \lambda_2 \end{pmatrix} \\
&= S^{-1} \begin{pmatrix} \lambda_1^n & \lambda_1^{n+1} \\ \lambda_2^n & \lambda_2^{n+1} \end{pmatrix} \\
&= \frac{1}{\lambda_2 - \lambda_1} \begin{pmatrix} \lambda_2 & -\lambda_1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1^n & \lambda_1^{n+1} \\ \lambda_2^n & \lambda_2^{n+1} \end{pmatrix} \\
&= -\frac{1}{\sqrt{5}} \begin{pmatrix} \lambda_1^n\lambda_2 - \lambda_2^n\lambda_1 & \lambda_2\lambda_1^{n+1} - \lambda_1\lambda_2^{n+1} \\ -\lambda_1^n + \lambda_2^n & -\lambda_1^{n+1} + \lambda_2^{n+1} \end{pmatrix} \\
&= -\frac{1}{\sqrt{5}} \begin{pmatrix} -\lambda_1^{n-1} + \lambda_2^{n-1} & -\lambda_1^n + \lambda_2^n \\ -\lambda_1^n + \lambda_2^n & -\lambda_1^{n+1} + \lambda_2^{n+1} \end{pmatrix} \qquad \text{since } \lambda_1\lambda_2 = -1 \\
&= \frac{1}{\sqrt{5}} \begin{pmatrix} \beta_{n-1} & \beta_n \\ \beta_n & \beta_{n+1} \end{pmatrix} \qquad \text{with } \beta_n := \lambda_1^n - \lambda_2^n.
\end{aligned}$$

Hence by the above

$$\begin{pmatrix} \alpha_n \\ \alpha_{n+1} \end{pmatrix} = \frac{1}{\sqrt{5}} \begin{pmatrix} \beta_{n-1} & \beta_n \\ \beta_n & \beta_{n+1} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

In particular we then have for α_n

$$\alpha_n = \frac{1}{\sqrt{5}}\beta_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right].$$

Other examples for program extraction from constructive proofs are abundant in the literature. Major case studies done in our group include the

development of the Warshall algorithm in [**11**], and of the Dijkstra algorithm in [**4**].

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

$k \in x$	k occurs in the path x,
$\mathsf{Rf}(x)$	x is a repitition free path,
$P_i(x, j, k)$	x is an R-path from j to k whose inner elements are $< i$,
$x \mid y$	concatenation,

where the concatenation function $x \mid y$ is defined as follows. If x and y are paths with the same end and initial point, then $x \mid y$ is the path obtained by concatenating the two, where the same end and initial point is used only once. If end and initial points are different, the result is the empty list ϵ.

We list the used properties ("Lemmata") without computational content.

(11) $P_i(x, j, k) \to P_{i+1}(x, j, k)$

(12) $P_0(x, j, k) \to j \neq k \to R(j, k)$

(13) $P_i(x, j, i) \to P_i(y, i, k) \to P_{i+1}(x \mid y, j, k)$

(14) $P_i(x, j, i) \to \mathsf{Rf}(x) \to P_i(y, i, k) \to \mathsf{Rf}(y) \to \forall z \neg P_i(z, j, k) \to \mathsf{Rf}(x \mid y)$

(15) $P_{i+1}(x, j, k) \to \neg P_i(x, j, k) \to \exists^{\mathsf{cl}} y\, P_i(y, j, i)$

(16) $P_{i+1}(x, j, k) \to \neg P_i(x, j, k) \to \exists^{\mathsf{cl}} z\, P_i(z, i, k)$

The last two propositions contain the main idea of the proof: if there is a path from j to k whose inner elements are smaller than $i{+}1$, but not smaller than i, then there exist (classical existence!) paths from j to i and from i to k with inner elements smaller than i. Proposition (14) can be seen as follows: under the given hypothesises $x \mid y$ could only contain a repetition if x and y have a common inner element. But this would contradict $\forall z \neg P_i(z, j, k)$.

We now give a constructive proof of the present $\forall\exists$-Proposition. We first formulate the claim, that for our relation R (suppressed in the notation) and given i, j, k there is a path x with the following properties.

- If x is the empty list ϵ, then there is no R-path from j to k with inner elements $< i$, and
- if x is not the empty list ϵ, then x connects the nodes j and k, has inner elements $< i$ and is repitition free.

Proposition (Warshall).

$$\forall i, j, k \exists x. \big(x = \epsilon \to \forall y \neg P_i(y, j, k)\big) \wedge \big(x \neq \epsilon \to P_i(x, j, k) \wedge \mathsf{Rf}(x)\big).$$

Proof. Induction on i. *Base* $i = 0$. Given j, k.

Case $j = k$. Choose $x := j{:}$. Then $P_0(j{:}, j, k)$ because of $j = k$ and $\mathsf{Rf}(j{:})$ by definition.

Case $j \neq k$.

UCase $R(j, k)$. Choose $x := j :: k{:}$. Then we have $P_0(j :: k{:}, j, k)$ because of $R(j, k)$ and $\mathsf{Rf}(j :: k{:})$ because of $j \neq k$.

UCase $\neg R(j,k)$. Choose $x := \epsilon$. We must show $\forall y \neg P_0(y,j,k)$. Let y be given, and assume $P_0(y,j,k)$. Then $R(j,k)$ by (12), a contradiction.

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

Case $x_i = \epsilon$. Then $\forall y \neg P_i(y,j,k)$. By IH we have $x_{i,j,i}$ and $x_{i,i,k}$.

SubCase $x_{i,j,i} = \epsilon$. Then $\forall y \neg P_i(y,j,i)$. Let $x_{i+1} = \epsilon$. Given y. Assume $P_{i+1}(y,j,k)$. by (15) (because of $\neg P_i(y,j,k)$) we have $\exists^{\mathsf{cl}} y\ P_i(y,j,i)$. Contradiction.

SubCase $x_{i,j,i} \neq \epsilon$.

SubSubCase $x_{i,i,k} = \epsilon$. Then $\forall y \neg P_i(y,i,k)$. Let $x_{i+1} = \epsilon$. Given y. Assume $P_{i+1}(y,j,k)$. By (16) (because of $\neg P_i(y,j,k)$) we have $\exists^{\mathsf{cl}} z\ P_i(z,i,k)$. Contradiction.

SubSubCase $x_{i,i,k} \neq \epsilon$. Let $x_{i+1} := x_{i,j,i} \mid x_{i,i,k}$. Then $P_{i+1}(x_{i+1},j,k)$ by (13) and $\mathsf{Rf}(x_{i+1})$ by (14), since $\forall y \neg P_i(y,j,k)$ follows from $x_i = \epsilon$.

Case $x_i \neq \epsilon$. Choose $x_{i+1} := x_i$. Then $\mathsf{Rf}(x_{i+1})$ and $P_i(x_{i+1},j,k)$, hence $P_{i+1}(x_{i+1},j,k)$ by (11). □

From a formal proof of this proposition we can extract the following program: (original output of MINLOG, with renaming of bound variables, and indentation added)

```
[r](Rec 1 nat=>nat=>boole nat=>nat=>nat=>list nat)r
([j,k][if (j=k) ([x6,x7]x6) ([x6,x7]x7)]j:
 ([if (r j k) ([x6,x7]x6) ([x6,x7]x7)](j::k:)(Nil nat)))
([i,f,j,k]
 [if (f j k=(Nil nat)) ([x8,x9]x8) ([x8,x9]x9)]
 ([if (f j i=(Nil nat)) ([x8,x9]x8) ([x8,x9]x9)]
  (Nil nat)
  ([if (f i k=(Nil nat)) ([x8,x9]x8) ([x8,x9]x9)]
   (Nil nat)(f j i|f i k)))
 (f j k))
```

To make this program more readable, we give the (primitive) recursion equations for the defined function f. For given i,j,k they either yield a path from j to k with innee elements $< i$ or else the empty path ϵ.

$$
\begin{array}{rcl}
f(0,j,k) & := & \textbf{if}\ \ j=k\ \ \textbf{then}\ \ j{:}\ \ \textbf{else} \\
 & & \quad \textbf{if}\ \ Rjk\ \ \textbf{then}\ \ j :: k{:}\ \ \textbf{else}\ \ \epsilon\ \ \textbf{fi fi} \\
f(i+1,j,k) & := & \textbf{if}\ \ f(i,j,k)=\epsilon\ \ \textbf{then} \\
 & & \quad \textbf{if}\ \ f(i,j,i)=\epsilon\ \ \textbf{then} \\
 & & \qquad \epsilon\ \ \textbf{else} \\
 & & \qquad \textbf{if}\ \ f(i,i,k)=\epsilon\ \ \textbf{then} \\
 & & \qquad\quad \epsilon\ \ \textbf{else} \\
 & & \qquad\quad f(i,j,i) \mid f(i,i,k)\ \ \textbf{fi fi else} \\
 & & \quad f(i,j,k)\ \ \textbf{fi}
\end{array}
$$

Note that the algorithm in exactly this form is exponential, since in the step case $f(i+1,j,k)$ it contains the three recursive calls $f(i,j,k)$, $f(i,j,i)$ und $f(i,i,k)$. With a well known technique we can transform it into a polynomial algorithm: consider f as a unary function computing a $n \times n$-matrix (i.e. *not* as a function of three arguments).

4.4.3. An Informal Proof for the Dijkstra Algorithm. We want to show how the Dijkstra algorithm can be obtained from a proof of an appropriate existence theorem. To this end we enrich the Dijkstra algorithm such that it not only yields shortest distances, but also corresponding paths. This seems to be an useful information anyway.

Notice that a common proof for the correctness of the Dijkstra algorithm cannot be used for this purpose, since it uses the minimum principle (it starts with choosing a shortest path $u_1 = v_0, v_1, \ldots, v_m, u_{i+1}$ from v_0 to u_{i+1}) and hence is not constructive.

So let a weighted graph $\Gamma = (V, E, W)$ be given, where $W\colon E \to \mathbb{N}$. For simplicity we assume $V = \{0, 1, \ldots, N-1\}$, where 0 is the distinguished node. Assume $W(0,0) = 0$ and – if u, v are not both 0 – $W(u,v) := \infty$ if $\{u, v\} \notin E$.

We show: for every node a there is a "shortest distance" $s \in \mathbb{N}$ from the distinguished node v_0 to a. Writing

$$S_f(m) := \sum_{i<m} W(f(i), f(i+1)) \quad \text{(distance sum)},$$

this means

(a) There is a sequence g of $n+1$ nodes starting with the distinguished node 0 and ending with the node a, whose distance sum $S_g(n)$ is s.
(b) For every sequence f of m nodes, which also starts with the distinguished node 0 and ends with a, the distance sum $S_f(m)$ is at least as big as s.

In other words:

THEOREM. *For every $a \in V$ there is an $s \in \mathbb{N}$ such that*

(a) $(\exists g, n{<}N).g(0) = 0 \wedge g(n) = a \wedge s = S_g(n)$
(b) $\forall f, m.f(0) = 0 \to f(m) = a \to s \le S_f(m)$

For the proof we shall construct functions $\mathsf{nxt}\colon V \to V$ and $d\colon V \to \mathbb{N} \cup \{\infty\}$ with the following intended meaning:

$\mathsf{nxt}(a)$	successor on a shortest path to 0, or 0,
$d(a)$	length of this path.

For these we will show the following properties, for all $a, b \in V$:

(17) $d(0) = 0$

(18) $a \neq 0 \to \mathsf{nxt}(a) \neq a$

(19) $d(\mathsf{nxt}(a)) < \infty$

(20) $d(a) = d(\mathsf{nxt}(a)) + W(\mathsf{nxt}(a), a)$

(21) $d(a) \le d(b) + W(b, a)$

We first show that from (17)–(21) the above theorem follows.

PROOF. Fix $s := d(a)$.
(a). Let

$$h(0) := a, \quad h(i+1) := \begin{cases} \mathsf{nxt}(h(i)) & \text{if } h(i) \neq 0 \\ 0 & \text{sonst.} \end{cases}$$

In case $h(i) \neq 0$ we have

$$\begin{aligned} d(h(i+1)) &= d(\mathsf{nxt}(h(i))) \\ &< d(\mathsf{nxt}(h(i))) + W(\mathsf{nxt}(h(i)), h(i)) && \text{because of (18) and (19)} \\ &= d(h(i)). && \text{because of (20).} \end{aligned}$$

Hence there is a least $n < N$ such that $h(n) = 0$. Let $g(i) := h(n-i)$, so $g(0) = h(n) = 0$ and $g(n) = h(0) = a$. We show

$$\sum_{k<j} W(g(k), g(k+1)) = d(g(j)),$$

by induction on j. Basis $j = 0$. Clear from $g(0) = 0$. Step $j \mapsto j+1$. Because of $j < n$ we have $i := n - j - 1 < n$, hence

$$\begin{aligned} h(i+1) &= \mathsf{nxt}(h(i)) \\ h(n-(n-i-1)) &= \mathsf{nxt}(h(n-(n-i))) \\ g(n-i-1) &= \mathsf{nxt}(g(n-i)) \\ g(j) &= \mathsf{nxt}(g(j+1)). \end{aligned}$$

Therefore

$$\begin{aligned} &\sum_{k<j+1} W(g(k), g(k+1)) \\ &= \sum_{k<j} W(g(k), g(k+1)) + W(g(j), g(j+1)) \\ &= d(g(j)) + W(g(j), g(j+1)) && \text{by IH} \\ &= d(\mathsf{nxt}(g(j+1))) + W(\mathsf{nxt}(g(j+1)), g(j+1)) && \text{see above} \\ &= d(g(j+1)) && \text{by (20).} \end{aligned}$$

(b). Let $f\colon \mathbb{N} \to V$ be given with $f(0) = 0$. We show $f(m) = a \to d(a) \leq S_f(m)$ by induction on m. Basis $m = 0$. Then $a = f(0) = 0$, hence $d(a) = 0$ by (17). Step $m \mapsto m+1$. Assume $a = f(m+1)$. Then $s = d(a) = d(f(m+1))$ and

$$\begin{aligned} d(f(m+1)) &\leq d(f(m)) + W(f(m), f(m+1)) && \text{by (21)} \\ &\leq \sum_{i<m} W(f(i), f(i+1)) + W(f(m), f(m+1)) && \text{by IH} \\ &= \sum_{i<m+1} W(f(i), f(i+1)) \\ &= S_f(m+1). \end{aligned}$$

This completes the proof. □

To construct the functions $\mathsf{nxt}\colon V \to V$ and $d\colon V \to \mathbb{N} \cup \{\infty\}$ we define, for every $n < N$

- a set $C_n \subseteq V$, the set of *computed* nodes of level n,
- a function $\mathsf{nxt}_n\colon V \to V$ assigning to every node the *next* node on a shortest path from a to 0, as far as this is known at level n,

- a function $d_n\colon V \to \mathbb{N} \cup \{\infty\}$ assigning to every node its *distance* from 0 along the path to 0 determined by the nxt-function, again as far as this is known at level n,
- p_n, the node "picked" at level n.

Let

$$C_0 := \{0\}, \qquad \mathsf{nxt}_0(a) := 0, \qquad d_0(a) := W(a, 0).$$

Assume that at level n we have picked p_n such that

$$p_n < N \wedge p_n \notin C_n \wedge \forall a.d_n(a) < d_n(p_n) \to a \in C_n. \tag{22}$$

Let

$$C_{n+1} := C_n \cup \{p_n\},$$

$$\mathsf{nxt}_{n+1}(a) := \begin{cases} p_n & \text{if } d_n(p_n) + W(p_n, a) < d_n(a), \\ \mathsf{nxt}_n(a) & \text{otherwise,} \end{cases}$$

$$d_{n+1}(a) := \min\{d_n(p_n) + W(p_n, a), d_n(a)\}.$$

Observe that the algorithm is relative to another simple search algorithm pick, which assigns to a given set $C \subsetneq V$ and distance function $d\colon V \to \mathbb{N} \cup \{\infty\}$ a $\mathsf{pick}(C, d) \in V \setminus C$ that among all $a \in V \setminus C$ has minimal distance $d(a)$.

We will prove for every $n < N$:

$$\#(C_n) = n + 1, \tag{23}$$

$$d_n(0) = 0, \tag{24}$$

$$a \neq 0 \to \mathsf{nxt}_n(a) \neq a, \tag{25}$$

$$\mathsf{nxt}_n(a) \in C_n, \tag{26}$$

$$d_n(\mathsf{nxt}_n(a)) < \infty, \tag{27}$$

$$d_n(a) = d_n(\mathsf{nxt}_n(a)) + W(\mathsf{nxt}_n(a), a), \tag{28}$$

$$b \in C_n \to d_n(a) \leq d_n(b) + W(b, a), \tag{29}$$

$$b \in C_n \to d_n(a) < d_n(b) \to a \in C_n. \tag{30}$$

From (23) it follows that $C_{N-1} = V$, and (24)–(29) then express that $d := d_{N-1}$ and $\mathsf{nxt} := \mathsf{nxt}_{N-1}$ satisfy the conditions (17)–(21).

It remains to show that (23)–(30) hold. We prove this by induction on n. The base case $n = 0$ is clear. For the step assume that (23)–(30) hold for n, and that p_n is chosen according to (22), hence

$$p_n < N, \quad p_n \notin C_n, \quad \forall a.d_n(a) < d_n(p_n) \to a \in C_n.$$

(23) for $n + 1$ follows from $p_n < N$ and $p_n \notin C_n$. (24) for $n + 1$ is clear, and (25) for $n + 1$ follows from the definition with IH(25). Also (26) follows immediately from the definitions. – We now first show

$$d_{n+1}(a) = d_n(a) \quad \text{for all } a \in C_n. \tag{31}$$

Let $a \in C_n$. Because of $p_n \notin C_n$ we have $d_n(a) \leq d_n(p_n)$ by IH(30), hence $d_{n+1}(a) = d_n(a)$ by definition.

Proof of $d_{n+1}(\mathsf{nxt}_{n+1}(a)) < \infty$. In case $d_n(p_n) + W(p_n, a) < d_n(a)$ we have $d_{n+1}(a) = d_n(p_n) + W(p_n, a) < \infty$, hence

$$d_{n+1}(\mathsf{nxt}_{n+1}(a)) = d_{n+1}(p_n) = d_n(p_n) < \infty.$$

On the other hand $\mathsf{nxt}_{n+1}(a) = \mathsf{nxt}_n(a)$, hence by $\mathsf{nxt}_n(a) \in C_n$ (because of (31)) also $d_{n+1}(\mathsf{nxt}_{n+1}(a)) = d_n(\mathsf{nxt}_n(a)) < \infty$.

Proof of $\forall a\, d_{n+1}(a) = d_{n+1}(\mathsf{nxt}_{n+1}(a)) + W(\mathsf{nxt}_{n+1}(a), a))$. *Case* 1: $d_n(p_n) + W(p_n, a) < d_n(a)$. Observe that $d_{n+1}(p_n) = d_n(p_n)$, for $d_{n+1}(p_n) = \min\{d_n(p_n) + W(p_n, p_n), d_n(p_n)\}$. Hence

$$\begin{aligned} & d_{n+1}(a) \\ &= d_{n+1}(p_n) + W(p_n, a) && \text{by definition and remark} \\ &= d_{n+1}(\mathsf{nxt}_{n+1}(a)) + W(\mathsf{nxt}_{n+1}(a), a) && \text{by definition of } \mathsf{nxt}_{n+1}(a). \end{aligned}$$

Case 2: Otherwise. Then by definition $d_{n+1}(a) = d_n(a)$ and $\mathsf{nxt}_{n+1}(a) = \mathsf{nxt}_n(a)$. The claim follows from the IH and (31) because of $\mathsf{nxt}_n(a) \in C_n$.

Proof of $\forall a, b.b \in C_{n+1} \to d_{n+1}(a) \le d_{n+1}(b) + W(b, a)$. Let $b \in C_{n+1}$. *Case* $b \in C_n$. Then

$$\begin{aligned} d_{n+1}(a) &\le d_n(a) && \text{by definition} \\ &\le d_n(b) + W(b, a) && \text{by IH} \\ &= d_{n+1}(b) + W(b, a) && \text{by (31).} \end{aligned}$$

Case 2: $b = p_n$. Observe again that $d_{n+1}(p_n) = d_n(p_n)$, since $d_{n+1}(p_n) = \min\{d_n(p_n) + W(p_n, p_n), d_n(p_n)\}$. Then

$$\begin{aligned} d_{n+1}(a) &\le d_n(p_n) + W(p_n, a) && \text{by definition} \\ &= d_{n+1}(p_n) + W(p_n, a) && \text{by the remark} \\ &= d_{n+1}(b) + W(b, a). \end{aligned}$$

Proof of $\forall a, b.b \in C_{n+1} \to d_{n+1}(a) < d_{n+1}(b) \to a \in C_{n+1}$. Assume $b \in C_{n+1}$ and $d_{n+1}(a) < d_{n+1}(b)$. *Case* $d_n(p_n) + W(p_n, a) < d_n(a)$, i.e. $d_{n+1}(a) = d_n(p_n) + W(p_n, a)$, hence

$$\begin{aligned} d_n(p_n) &\le d_n(p_n) + W(p_n, a) \\ &= d_{n+1}(a) && \text{by definition} \\ &< d_{n+1}(b) && \text{by assumption} \\ &\le d_n(b) && \text{by definition.} \end{aligned}$$

Subcase $b \in C_n$. Then from the IH and the inequality above we obtain $p_n \in C_n$, hence a contradiction. *Subcase* $b = p_n$. The above inequality immediately entails the contradiction $d_n(p_n) < d_n(p_n)$.

Case otherwise, i.e. $d_{n+1}(a) = d_n(a)$, hence

$$\begin{aligned} d_n(a) = d_{n+1}(a) && \text{by definition} \\ < d_{n+1}(b) && \text{by assumption} \\ \le d_n(b) && \text{by definition.} \end{aligned}$$

Subcase $b \in C_n$. From the IH we obtain $a \in C_n$. *Subcase* $b = p_n$. The above inequality yields $a \in C_n$, by the choice of p_n.

4.5. Notes

We have made extensive use of [**11**]. The idea of using universal quantifiers without computational content is taken from Berger's [**5**]. Dan Hernest pointed out an oversight in my original definition of realizability for $\exists^{\mathsf{nc}}$.

CHAPTER 5

Inductive Definitions

As we have seen, type variables allow for a general treatment of inductively generated types $\mu\vec{\alpha}\,\vec{\kappa}$. Similarly, we can use predicate variables to inductively generate predicates $\mu\vec{X}\,\vec{K}$.

More precisely, we allow the formation of inductively generated predicates $\mu\vec{X}\,\vec{K}$, where $\vec{X} = (X_j)_{j=1,\dots,N}$ is a list of distinct predicate variables, and $\vec{K} = (K_i)_{i=1,\dots,k}$ is a list of constructor formulas (or "clauses") containing $X_1, \dots, X_N$ in strictly positive positions only.

In this chapter, totality plays no role. For the sake of readability we therefore use object variables $x, y, \dots$ without a hat ˆ to range over arbitrary objects of the underlying domain.

5.1. Axioms

5.1.1. Introduction and Elimination Axioms.

DEFINITION. Let $\vec{X} = (X_j)_{j=1,\dots,N}$ be a list of distinct predicate variables. *Formulas* $A, B, C, D \in \mathsf{F}$, *predicates* $P, Q, I \in \mathsf{P}$ and *constructor formulas* $K \in \mathsf{KF}(\vec{X})$ are defined inductively as follows.

$$\frac{\vec{A}, \vec{B}_1, \dots, \vec{B}_n \in \mathsf{F} \quad (n \geq 0)}{\forall^{\mathsf{nc}}\vec{x'}\forall\vec{x}.\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to X_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to X_j(\vec{t}) \in \mathsf{KF}(\vec{X})}$$

$$\frac{K_1, \dots, K_n \in \mathsf{KF}(\vec{X}) \quad (n \geq 1)}{(\mu\vec{X}\,(K_1, \dots, K_n))_j \in \mathsf{P}} \qquad \frac{P \in \mathsf{P}}{P(\vec{r}) \in \mathsf{F}} \qquad \frac{C \in \mathsf{F}}{\{\,\vec{x} \mid C\,\} \in \mathsf{P}}$$

$$\frac{A, B \in \mathsf{F}}{A \to B \in \mathsf{F}} \qquad \frac{A, B \in \mathsf{F}}{A \wedge B \in \mathsf{F}} \qquad \frac{A \in \mathsf{F}}{\forall x^\rho A \in \mathsf{F}} \qquad \frac{A \in \mathsf{F}}{\forall^{\mathsf{nc}} x^\rho A \in \mathsf{F}}$$

A predicate of the form $\{\,\vec{x} \mid C\,\}$ is called a *comprehension term*. We identify $\{\,\vec{x} \mid C\,\}(\vec{r})$ with $C[\vec{x}{:=}\vec{r}]$.

We shall use I for predicates of the form $(\mu\vec{X}\,(K_1, \dots, K_k))_j$ only, and for predicates $\vec{P}$ and a constructor formula

$$K_i = \forall^{\mathsf{nc}}\vec{x'}_i\forall\vec{x}_i.\vec{A}_i \to \big(\forall^{\mathsf{nc}}\vec{y'}_{i\nu}\forall\vec{y}_{i\nu}.\vec{B}_{i\nu} \to X_{j_{i\nu}}(\vec{s}_{i\nu})\big)_{\nu=1,\dots,n_i} \to X_{j_i}(\vec{t}_i)$$

with $1 \leq i \leq k$ we shall write

$$K_i[\vec{P}] = \forall^{\mathsf{nc}}\vec{x'}_i\forall\vec{x}_i.\vec{A}_i \to \big(\forall^{\mathsf{nc}}\vec{y'}_{i\nu}\forall\vec{y}_{i\nu}.\vec{B}_{i\nu} \to P_{j_{i\nu}}(\vec{s}_{i\nu})\big)_{\nu=1,\dots,n_i} \to P_{j_i}(\vec{t}_i).$$

Now let $\vec{I} := \mu\vec{X}\,\vec{K}$ be inductively defined predicates. According to their intended meaning, we postulate the *introduction axioms* $\vec{K}[\vec{I}]$, i.e. all axioms

(32)

$$K_i[\vec{I}] = \forall^{\mathsf{nc}}\vec{x'}_i\forall\vec{x}_i.\vec{A}_i \to \big(\forall^{\mathsf{nc}}\vec{y'}_{i\nu}\forall\vec{y}_{i\nu}.\vec{B}_{i\nu} \to I_{j_{i\nu}}(\vec{s}_{i\nu})\big)_{\nu=1,\dots,n_i} \to I_{j_i}(\vec{t}_i)$$

for $i = 1, \ldots, k$, and for $j = 1, \ldots, N$ the *elimination axioms*

$$\forall^{\mathsf{nc}}\vec{x}_j.K_1[\vec{P}] \to \cdots \to K_k[\vec{P}] \to I_j(\vec{x}_j) \to P_j(\vec{x}_j).$$

5.1.2. Introduction and Elimination Axioms. However, in applications one often wants to use a strengthened form of the elimination axioms. For their formulation it is useful to introduce the notation $K[\vec{Q}, \vec{P}]$ for

$$\begin{aligned}\forall^{\mathsf{nc}}\vec{x'}_j\forall\vec{x}_j.\vec{A} \to &\big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to Q_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\ldots,n} \to \\ &\big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to P_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\ldots,n} \to P_j(\vec{t}).\end{aligned}$$

Then the *strengthened elimination axioms* are

$$\forall^{\mathsf{nc}}\vec{x}_j.K_1[\vec{I}, \vec{P}] \to \cdots \to K_k[\vec{I}, \vec{P}] \to I_j(\vec{x}_j) \to P_j(\vec{x}_j). \tag{33}$$

They are indeed stronger (or better: easier to use), because each premise $K_i[\vec{I}, \vec{P}]$ is weaker than $K_i[\vec{P}]$ (because $K_i[\vec{I}, \vec{P}]$ has more premises than $K_i[\vec{P}]$). However, there is no essential difference, because from the (ordinary) elimination axiom

$$\forall^{\mathsf{nc}}\vec{x}_j.K_1[\vec{I} \wedge \vec{P}] \to \cdots \to K_k[\vec{I} \wedge \vec{P}] \to I_j(\vec{x}_j) \to I_j(\vec{x}_j) \wedge P_j(\vec{x}_j)$$

(with $\vec{I} \wedge \vec{P}$ denoting the list of predicates $\{\, \vec{x}_j \mid I_j(\vec{x}_j) \wedge P_j(\vec{x}_j) \,\}$) we can derive the strengthened one (33). To see this, assume $\vec{x}_j$, $K_1[\vec{I}, \vec{P}] \ldots K_k[\vec{I}, \vec{P}]$ and $I_j(\vec{x}_j)$. We must show $P_j(\vec{x}_j)$. To this end we use the ordinary elimination axiom above. Hence it suffices to prove each $K_i[\vec{I} \wedge \vec{P}]$. So assume $\vec{x'}$, $\vec{x}$, $\vec{A}$ and $\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to I_{j_\nu}(\vec{s}_\nu) \wedge P_{j_\nu}(\vec{s}_\nu)$ for $\nu = 1, \ldots, n$. We must show $I_j(\vec{t}) \wedge P_j(\vec{t})$. Now $I_j(\vec{t})$ follows from the introduction axioms, and $P_j(\vec{t})$ follows from $K_i[\vec{I}, \vec{P}]$. □

We shall from now on normally use the strengthened elimination axioms, for they are more useful in practice.

As is to be expected from the terminology, we have conversion rules, converting a (strengthened) elimination immediately following an introduction:

$$\begin{aligned}&\mathsf{Elim}_j\vec{p}\vec{q}\vec{t}_i[\vec{x}_i{:=}\vec{r}_i](M_\kappa^{K_\kappa[\vec{I}[\vec{p}],\vec{P}[\vec{p},\vec{q}]]})_{\kappa=1,\ldots,k}(\mathsf{Intro}_i\vec{p}\vec{r'}_i\vec{r}_i\vec{N}_i^P\vec{N}_i^R) \mapsto \\ &M_i\vec{r'}_i\vec{r}_i\vec{N}_i^P\vec{N}_i^R \\ &\quad\big(\lambda\vec{y}_{i\nu}\lambda\vec{u}_{i\nu}.\mathsf{Elim}_{j_{i\nu}}\vec{p}\vec{q}\vec{s}_{i\nu}[\vec{x}_i{:=}\vec{r}_i](M_\kappa)_{\kappa=1,\ldots,k}(N_{i\nu}^R\vec{y}_{i\nu}\vec{u}_{i\nu})\big)_{\nu=1,\ldots,n_i},\end{aligned}$$

for $j = 1, \ldots, N$. Here $\vec{p}$ are to be substituted for the free variables of $\vec{I}$, and $\vec{q}$ for those of $\vec{P}$ which are not in $\vec{I}$. Moreover, the $\vec{N}_i^P$ are the parameter arguments for the i-th introduction axiom $K_i[\vec{I}]$ (i.e., proofs of its premises $\vec{A}_i$), and $\vec{N}_i^R$ are the recursive arguments (i.e., proofs of its further premises $\forall^{\mathsf{nc}}\vec{y'}_{i\nu}\forall\vec{y}_{i\nu}.\vec{B}_{i\nu} \to I_{j_{i\nu}}(\vec{s}_{i\nu})$).

5.1.3. Examples.

5.1.3.1. *The Even Numbers.* We begin with a very simple example, an inductive definition of the set Even of the even numbers. As underlying algebra we take the one generated from 0 by one unary constructor S, i.e. the natural numbers. The introduction axioms are

$$\mathsf{Even}(0)$$
$$\forall^{\mathsf{nc}} n.\mathsf{Even}(n) \to \mathsf{Even}(\mathsf{S}(\mathsf{S}(n)))$$

and the (strengthened) elimination axiom is

$$\forall^{\mathsf{nc}} n.P(0) \to (\forall^{\mathsf{nc}} n.\mathsf{Even}(n) \to P(n) \to P(\mathsf{S}(\mathsf{S}(n)))) \to \mathsf{Even}(n) \to P(n).$$

As an example of the use of the elimination axiom we prove

$$\forall^{\mathsf{nc}} n.\mathsf{Even}(\mathsf{S}(\mathsf{S}(n))) \to \mathsf{Even}(n).$$

Let P be the comprehension term $\{\, n \mid \forall^{\mathsf{nc}} m.n = \mathsf{S}(\mathsf{S}(m)) \to \mathsf{Even}(m) \,\}$. Then $P(0)$ holds (by ex-falso), and for an arbitrary n with $\mathsf{Even}(n)$ we have $P(\mathsf{S}(\mathsf{S}(n)))$. Then $\forall^{\mathsf{nc}} n.\mathsf{Even}(n) \to P(n)$ by the elimination axiom. Specializing this to $\mathsf{S}(\mathsf{S}(n))$ and unfolding P gives

$$\mathsf{Even}(\mathsf{S}(\mathsf{S}(n))) \to \forall^{\mathsf{nc}} m.\mathsf{S}(\mathsf{S}(n)) = \mathsf{S}(\mathsf{S}(m)) \to \mathsf{Even}(m),$$

hence the claim.

5.1.3.2. *The Even and the Odd Numbers.* As an easy example of a simultaneous inductive definition we take the sets Ev and Od of the even and odd numbers. Again we choose the natural numbers as the underlying algebra. The introduction axioms are

$$\mathsf{Ev}(0) \qquad\qquad \mathsf{Od}(1)$$
$$\forall^{\mathsf{nc}} n.\mathsf{Od}(n) \to \mathsf{Ev}(\mathsf{S}(n)) \qquad \forall^{\mathsf{nc}} n.\mathsf{Ev}(n) \to \mathsf{Od}(\mathsf{S}(n))$$

and the (strengthened) elimination axioms are

$$\forall^{\mathsf{nc}} n.P_0(0) \to (\forall^{\mathsf{nc}} n.\mathsf{Od}(n) \to P_1(n) \to P_0(\mathsf{S}(n))) \to$$
$$\quad P_1(1) \to (\forall^{\mathsf{nc}} n.\mathsf{Ev}(n) \to P_0(n) \to P_1(\mathsf{S}(n))) \to \mathsf{Ev}(n) \to P_0(n),$$

$$\forall^{\mathsf{nc}} n.P_0(0) \to (\forall^{\mathsf{nc}} n.\mathsf{Od}(n) \to P_1(n) \to P_0(\mathsf{S}(n))) \to$$
$$\quad P_1(1) \to (\forall^{\mathsf{nc}} n.\mathsf{Ev}(n) \to P_0(n) \to P_1(\mathsf{S}(n))) \to \mathsf{Od}(n) \to P_1(n).$$

5.1.3.3. *Trees and Tree Lists.* As another example of a simultaneous inductive definition we consider trees and tree lists. In Section 2.1 we viewed trees and tree lists as an example of simultaneously defined algebras. Alternatively, we can choose as a somewhat more general underlying algebra the one generated from 0 and Nil by one binary constructor cons, and then single out trees and tree lists by a simultaneous inductive definition. Let a, b range over elements of the underlying algebra. The introduction axioms are

$$\mathsf{Tree}(0) \qquad\qquad \mathsf{Tlist}(\mathsf{Nil})$$
$$\forall^{\mathsf{nc}} a.\mathsf{Tlist}(a) \to \mathsf{Tree}(a) \qquad \forall^{\mathsf{nc}} a, b.\mathsf{Tree}(a) \to \mathsf{Tlist}(b) \to \mathsf{Tlist}(\mathsf{cons}(a, b))$$

and the (strengthened) elimination axioms are

$$\forall^{\mathsf{nc}}a.P_0(0) \to \big(\forall^{\mathsf{nc}}a.\mathsf{Tlist}(a) \to P_1(a) \to P_0(a)\big) \to$$
$$P_1(\mathsf{Nil}) \to \big(\forall^{\mathsf{nc}}a,b.\mathsf{Tree}(a) \to \mathsf{Tlist}(b) \to P_0(a) \to P_1(b) \to P_1(\mathsf{cons}(a,b))\big) \to$$
$$\mathsf{Tree}(a) \to P_0(a),$$

$$\forall^{\mathsf{nc}}a.P_0(0) \to \big(\forall^{\mathsf{nc}}a.\mathsf{Tlist}(a) \to P_1(a) \to P_0(a)\big) \to$$
$$P_1(\mathsf{Nil}) \to \big(\forall^{\mathsf{nc}}a,b.\mathsf{Tree}(a) \to \mathsf{Tlist}(b) \to P_0(a) \to P_1(b) \to P_1(\mathsf{cons}(a,b))\big) \to$$
$$\mathsf{Tlist}(a) \to P_1(a).$$

5.1.3.4. *The Accessible Part of an Ordering.* We assume that we have a binary relation $\prec$. Its *accessible part* is inductively defined as follows. The introduction axiom is

$$\mathsf{Acc}_1^+ : \forall^{\mathsf{nc}}x.(\forall y.y \prec x \to \mathsf{Acc}(y)) \to \mathsf{Acc}(x),$$

and the (strengthened) elimination axiom Acc^- is

$$\forall^{\mathsf{nc}}x.\big(\forall^{\mathsf{nc}}x.(\forall y.y \prec x \to \mathsf{Acc}(y)) \to (\forall y.y \prec x \to P(y)) \to P(x)\big) \to$$
$$\mathsf{Acc}(x) \to P(x).$$

Notice that the ordinary elimination axiom in this case is

$$\forall^{\mathsf{nc}}x.\big(\forall^{\mathsf{nc}}x.(\forall y.y \prec x \to P(y)) \to P(x)\big) \to \mathsf{Acc}(x) \to P(x).$$

Conversion:

$$\mathsf{Acc}^-\vec{q}rM_1^{\forall^{\mathsf{nc}}x.(\forall y.y\prec x\to\mathsf{Acc}(y))\to(\forall y.y\prec x\to P(y))\to P(x)}\big(\mathsf{Acc}_1^+rN^R\big) \mapsto$$
$$M_1rN^R\lambda y\lambda u^{y\prec r}.\mathsf{Acc}^-\vec{q}yM_1(N^Ryu).$$

5.1.3.5. *The Bar predicate.* Call a sequence $w_0, \ldots, w_{n-1}$ of words *good* if there are indices $i < j < n$ and an embedding f of w_i into w_j. The introduction axioms are

$$\forall^{\mathsf{nc}}ws.\mathsf{Good}(ws) \to \mathsf{Bar}(ws),$$
$$\forall^{\mathsf{nc}}ws.\forall w\mathsf{Bar}(ws * w) \to \mathsf{Bar}(ws).$$

and the (strengthened) elimination axiom is

$$\forall^{\mathsf{nc}}ws.\big(\forall^{\mathsf{nc}}ws.\mathsf{Good}(ws) \to P(ws)\big) \to$$
$$\big(\forall^{\mathsf{nc}}ws.\forall w\mathsf{Bar}(ws * w) \to \forall wP(ws * w) \to P(ws)\big) \to$$
$$\mathsf{Bar}(ws) \to P(ws).$$

5.1.3.6. *The Transitive Closure of a Relation $\prec$.* The introduction axioms are

$$\forall^{\mathsf{nc}}x,y.x \prec y \to \mathsf{TrCl}(x,y),$$
$$\forall^{\mathsf{nc}}x,y,z.x \prec y \to \mathsf{TrCl}(y,z) \to \mathsf{TrCl}(x,z).$$

and the (strengthened) elimination axiom is

$$\forall^{\mathsf{nc}}x,y.\big(\forall^{\mathsf{nc}}x,y.x \prec y \to P(x,y)\big) \to$$
$$\big(\forall^{\mathsf{nc}}x,y,z.x \prec y \to \mathsf{TrCl}(y,z) \to P(y,z) \to P(x,z)\big) \to$$
$$\mathsf{TrCl}(x,y) \to P(x,y).$$

5.1.3.7. *The Strongly Normalizable λ-Terms.* Introduction axioms:

$$\mathsf{SN}(\mathsf{Var}(0))$$
$$\forall^{\mathsf{nc}} M.\big(\forall N.(M \rightsquigarrow N) \to \mathsf{SN}(N)\big) \to \mathsf{SN}(M),$$

Here $\mathsf{Var}(n)$ denotes the n-th (untyped) variable, and $M \rightsquigarrow N$ expresses that M reduces to N in one step.

5.1.3.8. *The Strongly Computable λ-Terms.* The set SC^{μ_j} of strongly computable λ-terms of type μ_j (cf. Subsection 2.2.4). Here SC_1 are previously defined SC-predicates, for previous types. Introduction axioms:

$$\mathsf{SC}(\mathsf{Var}(0))$$

$$\begin{aligned}\forall^{\mathsf{nc}} M.\big(&\forall N^P, N^R.M = c(N^P, N^R) \to \mathsf{SC}_1(N^P)\big) \to \\ &\big(\forall N^P, N^R, K.M = c(N^P, N^R) \to \mathsf{SC}_1(K) \to \mathsf{SC}(N^R K)\big) \to \\ &\big(\forall N.(M \rightsquigarrow N) \to \mathsf{SC}(N)\big) \to \\ &\quad \mathsf{SC}(M).\end{aligned}$$

As an example of predicates defined by simultaneous induction we consider $\mathsf{SC}^{\mathsf{tree}}$ and $\mathsf{SC}^{\mathsf{tlist}}$, defining the strongly computable terms of these types. The algebras tree and tlist have been simultaneously defined in with constructors

$$\mathsf{Leaf}^{\mathbf{N}\to\mathsf{tree}}, \mathsf{Branch}^{\mathsf{tlist}\to\mathsf{tree}}, \mathsf{Empty}^{\mathsf{tlist}}, \mathsf{Tcons}^{\mathsf{tree}\to\mathsf{tlist}\to\mathsf{tlist}}.$$

Following the general pattern above, the clauses for $\mathsf{SC}^{\mathsf{tree}}$ and $\mathsf{SC}^{\mathsf{tlist}}$ are

$$\mathsf{SC}^{\mathsf{tree}}(\mathsf{Leaf}(0)),$$

$$\begin{aligned}\forall^{\mathsf{nc}} M.\big(&\forall N^{\mathbf{N}}.M = \mathsf{Leaf}(N) \to \mathsf{SC}^{\mathbf{N}}(N)\big) \to \\ &\big(\forall N^{\mathsf{tlist}}.M = \mathsf{Branch}(N) \to \mathsf{SC}^{\mathsf{tlist}}(N)\big) \to \\ &\big(\forall N^{\mathsf{tree}}.(M \rightsquigarrow N) \to \mathsf{SC}^{\mathsf{tree}}(N)\big) \to \\ &\quad \mathsf{SC}^{\mathsf{tree}}(M),\end{aligned}$$

$$\mathsf{SC}^{\mathsf{tlist}}(\mathsf{Empty}),$$

$$\begin{aligned}\forall^{\mathsf{nc}} M.\big(&\forall N_1^{\mathsf{tree}}, N_2^{\mathsf{tlist}}.M = \mathsf{Tcons}(N_1, N_2) \to \mathsf{SC}^{\mathsf{tree}}(N_1)\big) \to \\ &\big(\forall N_1^{\mathsf{tree}}, N_2^{\mathsf{tlist}}.M = \mathsf{Tcons}(N_1, N_2) \to \mathsf{SC}^{\mathsf{tlist}}(N_2)\big) \to \\ &\big(\forall N^{\mathsf{tlist}}.(M \rightsquigarrow N) \to \mathsf{SC}^{\mathsf{tlist}}(N)\big) \to \\ &\quad \mathsf{SC}^{\mathsf{tlist}}(M).\end{aligned}$$

The following inductive definitions of disjunction, equality, the tensor, the existential quantifier and falsity have already been considered by Martin-Löf [**26**] and are used in the proof systems Coq [**32**] and ALF [**13**]; see also Berger's draft [**6**].

5.1.3.9. *Disjunction.* $A_1 \vee A_2 := \mu X(A_1 \to X, A_2 \to X)$. The introduction axioms are

$$\vee_0^+ \colon A_1 \to A_1 \vee A_2$$
$$\vee_1^+ \colon A_2 \to A_1 \vee A_2$$

and the elimination axiom is

$$\vee^- \colon (A_1 \to C) \to (A_2 \to C) \to A_1 \vee A_2 \to C.$$

Conversion rules:

$$\vee^- M_i^{A_1 \to C} M_2^{A_2 \to C} \big(\vee_0^+ N^{A_1}\big) \mapsto M_1 N,$$
$$\vee^- M_i^{A_1 \to C} M_2^{A_2 \to C} \big(\vee_1^+ N^{A_2}\big) \mapsto M_2 N,$$

and in case $A_1 \vee A_2$ has free variables to be substituted by $\vec{p}$ and the remaining free variables of C are to be substituted by $\vec{q}$

$$\vee^- \vec{p}\vec{q} M_i^{A_1[\vec{p}] \to C[\vec{p},\vec{q}]} M_2^{A_2[\vec{p}] \to C[\vec{p},\vec{q}]} \big(\vee_0^+ \vec{p} N^{A_1[\vec{p}]}\big) \mapsto M_1 N,$$
$$\vee^- \vec{p}\vec{q} M_i^{A_1[\vec{p}] \to C[\vec{p},\vec{q}]} M_2^{A_2[\vec{p}] \to C[\vec{p},\vec{q}]} \big(\vee_1^+ \vec{p} N^{A_2[\vec{p}]}\big) \mapsto M_2 N.$$

5.1.3.10. *Equality.* $\mathsf{Eq} := \mu X \forall^{\mathsf{nc}} x.X(x,x)$. The introduction axiom is

$$\mathsf{Eq}^+ \colon \forall^{\mathsf{nc}} x \mathsf{Eq}(x,x)$$

and the elimination axiom

$$\mathsf{Eq}^- \colon \forall^{\mathsf{nc}} x, y. \forall^{\mathsf{nc}} x R(x,x) \to \mathsf{Eq}(x,y) \to R(x,y).$$

The conversion rule is

$$\mathsf{Eq}^- \vec{q} r r M^{\forall^{\mathsf{nc}} x R[\vec{q}](x,x)} (\mathsf{Eq}^+ r) \mapsto Mr.$$

5.1.3.11. *Tensor.* $A_1 \otimes A_2 := \mu X(A_1 \to A_2 \to X)$. The introduction axiom is

$$\otimes^+ \colon A_1 \to A_2 \to A_1 \otimes A_2$$

and the elimination axiom is

$$\otimes^- \colon (A_1 \to A_2 \to C) \to A_1 \otimes A_2 \to C.$$

Conversion (assuming that there are no free variables in A_1, A_2 and C):

$$\otimes^- M^{A_1 \to A_2 \to C} (\otimes^+ N_1^{A_1} N_2^{A_2}) \mapsto M N_1 N_2.$$

5.1.3.12. *Existential Quantifier.* $\exists Q^{(\alpha)} := \mu X \forall x^\alpha . Q(x) \to X$. The introduction axiom is

$$\exists^+ \colon \forall y^\rho . A \to \exists y^\rho A$$

where $\exists y^\rho A$ abbreviates $\exists\{\, y^\rho \mid A \,\}$, and the elimination axiom is

$$\exists^- \colon (\forall y^\rho . A \to C) \to \exists y^\rho A \to C.$$

Conversion:

$$\exists^- \vec{p}\vec{q} M^{\forall^{\mathsf{nc}} y^\rho . A[\vec{p}] \to C[\vec{p},\vec{q}]} \big(\exists^+ \vec{p} t^\rho N^{A[\vec{p}][y:=t]}\big) \mapsto MtN.$$

5.1.3.13. *Falsity.* This example is somewhat extreme, since the list $\vec{K}$ in the general form $\mu X\, \vec{K}$ is empty here. There is no introduction axiom, and hence no conversion as well; the elimination axiom is

$$\bot^- \colon \bot \to A.$$

5.1.4. Cases. As in Section 2.4.4 there is an important variant of the elimination axiom, where for a certain P_j no induction hypotheses are used. To understand what this means, consider a P_j-clause

$$\forall^{\mathsf{nc}}\vec{x}'\forall\vec{x}.\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y}'_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to X_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to X_j(\vec{t}).$$

In the (strengthened) elimination axiom it gives rise to a premise $K[\vec{I},\vec{P}]$

$$\begin{aligned}\forall^{\mathsf{nc}}\vec{x}'\forall\vec{x}.\vec{A} \to &\big(\forall^{\mathsf{nc}}\vec{y}'_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to I_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to\\ &\big(\forall^{\mathsf{nc}}\vec{y}'_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to P_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to P_j(\vec{t}).\end{aligned}$$

For the cases axiom we instead take $K^{\mathcal{C}}[\vec{I},\vec{P}]$:

$$\forall^{\mathsf{nc}}\vec{x}'\forall\vec{x}.\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y}'_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to I_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to P_j(\vec{t}).$$

The *cases axiom* then is

$$\mathsf{Cases}_{I_j(\vec{x})\to P_j(\vec{x})}\colon \forall^{\mathsf{nc}}\vec{x}.K^{\mathcal{C}}_{i_1}[\vec{I},\vec{P}] \to \cdots \to K^{\mathcal{C}}_{i_k}[\vec{I},\vec{P}] \to I_j(\vec{x}) \to P_j(\vec{x}),$$

where $K^{\mathcal{C}}_{i_1}[\vec{I},\vec{P}],\dots,K^{\mathcal{C}}_{i_k}[\vec{I},\vec{P}]$ consists of all clauses $K^{\mathcal{C}}_i[\vec{I},\vec{P}]$ concerning P_j. So this axiom amounts to distinguish cases on all clauses generating I_j.

Examples are

$$\mathsf{Cases}_{\mathsf{Even}(n)\to A}\colon \forall^{\mathsf{nc}}n.A[n{:=}0] \to \forall^{\mathsf{nc}}n.A[n{:=}\mathsf{S}(\mathsf{S}(n))] \to \mathsf{Even}(n) \to A$$

and for the simultaneous inductive definition of trees and tree lists above

$$\begin{aligned}\mathsf{Cases}_{\mathsf{Tlist}(b)\to A}\colon \forall^{\mathsf{nc}}b.A[b{:=}\mathsf{Nil}] \to&\\ \big(\forall^{\mathsf{nc}}a,b.\mathsf{Tree}(a) \to \mathsf{Tlist}(b) \to A[b{:=}\mathsf{cons}(a,b)]\big) \to&\\ \mathsf{Tlist}(b) \to A.&\end{aligned}$$

5.2. Computational Content

The intended meaning of an inductively defined predicate constant I is quite clear. However, we first have to make up our mind as to whether it is to have computational content. We can decide that we do not want computational content, but only when all clauses K are "invariant" (i.e. have the property that $r\,\mathbf{r}\,K$ is the same as K). Then we can only apply the elimination axiom to invariant formulas as well. Both conditions are needed for the Soundness Theorem.

Suppose I is to have computational content. Then the new predicate $I^{\mathbf{r}}$ to be used in the definition of realizability should have an arity $(\mu,\vec{\rho})$, where the first argument of type μ represents a generation tree, witnessing how the other arguments were put into the inductively defined predicate.

Clearly this type μ is an algebra type in the sense of Chapter 2: it is one component of the types $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ generated from constructor types $\kappa_i := \tau(K_i)$ for all constructor formulas $K_1,\dots K_k$ from $\vec{I} = \mu\vec{X}\,(K_1,\dots,K_k)$. In case there is no nullary constructor type for μ, we add one with the name $\mathsf{Dummy}\mu$.

Notice that for k predicate variables in the clauses of I, we have 2^k possibilities for the type μ, depending on which of these predicate variables are to have Harrop degree 0. For example, the inductive definition of the existential quantifier in 5.1.3.12 needs two versions, depending on whether we use a predicate variable $\hat{Q}$ of Harrop degree 0 or Q of Harrop degree 1.

So we extend for inductively defined predicate constants the definition of the computational content from Section 4.2 by

$$\tau(I(\vec{s})) := \begin{cases} \mu & \text{if } I \text{ has computational content} \\ \varepsilon & \text{otherwise} \end{cases}$$

and the definition of realizability from Section 4.3 by

$$r\,\mathbf{r}\,I(\vec{s}) \;=\; \begin{cases} I^{\mathbf{r}}(r,\vec{s}) & \text{if } I \text{ has computational content} \\ I(\vec{s}) & \text{otherwise.} \end{cases}$$

From the intended meaning of $\vec{I}^{\mathbf{r}}$ it is rather obvious how these predicates should de defined inductively. For every constructor formula

$$K = \forall^{\mathsf{nc}}\vec{x'}\forall\vec{x}.\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu.\vec{B}_\nu \to X_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to X_j(\vec{t})$$

of the original inductive definition of $\vec{I}$ we build the new constructor formula $K^{\mathbf{r}}$ as

$$\forall^{\mathsf{nc}}\vec{x'}\forall\vec{x},\vec{u},\vec{f}.\vec{u}\,\mathbf{r}\,\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu,\vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to Y_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to Y_j(c\vec{x}\vec{u}\vec{f},\vec{t}).$$

Here c is the constructor of the algebra $\vec{\mu} = \mu\vec{\alpha}\,\vec{\kappa}$ generated from our constructor types $\kappa_i := \tau(K_i)$ (i.e. for K_i we have $c := \mathsf{constr}_i$). Recall that

$$\tau(K) = \vec{\rho} \to \tau(\vec{A}) \to \big(\vec{\sigma}_\nu \to \tau(\vec{B}_\nu) \to \alpha_{j_\nu}\big)_{\nu=1,\dots,n} \to \alpha_j \in \mathsf{KT}(\vec{\alpha}),$$

and c is the corresponding constructor. Then $\vec{I}^{\mathbf{r}} := \mu\vec{Y}\,(\vec{K}^{\mathbf{r}})$, an inductive definition with no computational content. The *introduction axioms* for $I^{\mathbf{r}}_j$ are $K^{\mathbf{r}}[\vec{I}^{\mathbf{r}}]$:

$$\forall^{\mathsf{nc}}\vec{x'}\forall\vec{x}\forall\vec{u},\vec{f}.\vec{u}\,\mathbf{r}\,\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu,\vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to I^{\mathbf{r}}_j(c\vec{x}\vec{u}\vec{f},\vec{t})$$

and the (strengthened) *elimination axioms* for $I^{\mathbf{r}}_j$ are

$$\forall^{\mathsf{nc}}\vec{x},w.K^{\mathbf{r}}_1[\vec{I}^{\mathbf{r}},\vec{Q}] \to \cdots \to K^{\mathbf{r}}_k[\vec{I}^{\mathbf{r}},\vec{Q}] \to I^{\mathbf{r}}_j(w,\vec{x}) \to Q_j(w,\vec{x})$$

where $K^{\mathbf{r}}[\vec{I}^{\mathbf{r}},\vec{Q}]$ is

$$\forall^{\mathsf{nc}}\vec{x'}\forall\vec{x},\vec{u},\vec{f}.\vec{u}\,\mathbf{r}\,\vec{A} \to \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu,\vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu,\vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to Q_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to I^{\mathbf{r}}_j(c\vec{x}\vec{u}\vec{f},\vec{t}).$$

Examples. The accessible part of an ordering $\prec$. Recall the introduction axiom

$$K_1[\mathsf{Acc}] := \forall^{\mathsf{nc}}x.(\forall y.y \prec x \to \mathsf{Acc}(y)) \to \mathsf{Acc}(x).$$

The corresponding constructor type is

$$\kappa_1 := \tau(K_1) = (\rho \to \alpha) \to \alpha.$$

Then the algebra $\mathsf{algAcc} := \mu\alpha\,(\alpha,\kappa_1)$ has two constructors

$$\mathsf{DummyalgAcc}\colon \mathsf{algAcc},$$

$$\mathsf{Sup}\colon (\rho \to \mathsf{algAcc}) \to \mathsf{algAcc}.$$

Here $\mathsf{DummyalgAcc}$ had to be added, because there is no nullary constructor type κ_i for Acc. We obtain

$$K_1^{\mathbf{r}}[\mathsf{Acc}^{\mathbf{r}}] := \forall^{\mathsf{nc}} x \forall f.(\forall y.y \prec x \to \mathsf{Acc}^{\mathbf{r}}(f(y), y)) \to \mathsf{Acc}^{\mathbf{r}}(\mathsf{Sup}(f), x).$$

5.3. Soundness

Let $\vec{I} := \mu\vec{X}\,(K_1, \dots, K_k)$ be inductively defined. We define the extracted programs for the introduction axioms (32) and (strengthened) elimination axioms (33) by

$$[\![\mathsf{Intro}_K]\!] := c,$$
$$[\![\mathsf{Elim}_j]\!] := \mathcal{R}_j.$$

By $K^{\mathbf{r}}[\vec{I}^{\mathbf{r}}]$ we clearly have $c\,\mathbf{r}\,\mathsf{Intro}_K$. For to prove

$$\mathcal{R}_j\,\mathbf{r}\,\big(\forall^{\mathsf{nc}}\vec{x}.K_1[\vec{I}, \vec{P}] \to \cdots \to K_k[\vec{I}, \vec{P}] \to I_j(\vec{x}) \to P_j(\vec{x})\big)$$

let $\vec{x}, w_1, \dots, w_k$ be given such that $w_i\,\mathbf{r}\,K_i[\vec{I}, \vec{P}]$, i.e.

(34)
$$\begin{aligned}\forall^{\mathsf{nc}}\vec{x'}\forall\vec{x}, \vec{u}, \vec{f}, \vec{g}.\vec{u}\,\mathbf{r}\,\vec{A} \to {} & \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu, \vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to f_\nu\vec{y}_\nu\vec{v}_\nu\,\mathbf{r}\,I_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to \\ & \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu, \vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to g_\nu\vec{y}_\nu\vec{v}_\nu\,\mathbf{r}\,P_{j_\nu}(\vec{s}_\nu)\big)_{\nu=1,\dots,n} \to \\ & w_i\vec{x}\vec{u}\vec{f}\vec{g}\,\mathbf{r}\,P_j(\vec{t}).\end{aligned}$$

Let further w be given and assume $w\,\mathbf{r}\,I_j(\vec{x})$, i.e. $I_j^{\mathbf{r}}(w, \vec{x})$. Our goal is

$$\mathcal{R}_j\vec{w}w\,\mathbf{r}\,P_j(\vec{x}) =: Q_j(w, \vec{x}).$$

We use the (strengthened) elimination axiom for $I_j^{\mathbf{r}}$ with $Q_j(w, \vec{x})$, i.e.

$$\forall^{\mathsf{nc}}\vec{x}, w.K_1^{\mathbf{r}}[\vec{I}^{\mathbf{r}}, \vec{Q}] \to \cdots \to K_k^{\mathbf{r}}[\vec{I}^{\mathbf{r}}, \vec{Q}] \to I_j^{\mathbf{r}}(w, \vec{x}) \to Q_j(w, \vec{x}).$$

Hence it suffices to prove $K^{\mathbf{r}}[\vec{I}^{\mathbf{r}}, \vec{Q}]$ for every constructor formula K, i.e.

(35)
$$\begin{aligned}\forall^{\mathsf{nc}}\vec{x'}\forall\vec{x}, \vec{u}, \vec{f}.\vec{u}\,\mathbf{r}\,\vec{A} \to {} & \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu, \vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to I_{j_\nu}^{\mathbf{r}}(f_\nu\vec{y}_\nu\vec{v}_\nu, \vec{s}_\nu)\big)_{\nu=1,\dots,n} \to \\ & \big(\forall^{\mathsf{nc}}\vec{y'}_\nu\forall\vec{y}_\nu, \vec{v}_\nu.\vec{v}_\nu\,\mathbf{r}\,\vec{B}_\nu \to Q_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu, \vec{s}_\nu)\big)_{\nu=1,\dots,n} \to \\ & Q_j(c\vec{x}\vec{u}\vec{f}, \vec{t}).\end{aligned}$$

So assume $\vec{x'}, \vec{x}, \vec{u}, \vec{f}$ and the premises of (35). We must show $Q_j(c\vec{x}\vec{u}\vec{f}, \vec{t})$, i.e.,

$$\mathcal{R}_j\vec{w}(c\vec{x}\vec{u}\vec{f})\,\mathbf{r}\,P_j(\vec{t}).$$

Since $c = \mathsf{constr}_i$, by the conversion rules for $\mathcal{R}$ this is the same as

$$w_i\vec{x}\vec{u}\vec{f}\big((\mathcal{R}_{j_1}\vec{w}) \circ f_1\big) \dots \big((\mathcal{R}_{j_n}\vec{w}) \circ f_n\big)\,\mathbf{r}\,P_j(\vec{t}).$$

To this end we use (34) with $\vec{x'}, \vec{x}, \vec{u}, \vec{f}, (\mathcal{R}_{j_1}\vec{w}) \circ f_1, \dots, (\mathcal{R}_{j_n}\vec{w}) \circ f_n$. Its conclusion is what we want, and its premises follow from the premises of (35).

5.4. Notes

Part of the the material in the present Chapter 5 is based on Monika Seisenberger's Thesis [**33**].

CHAPTER 6

Program Extraction from Classical Proofs

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

We develop a refined method of extracting reasonable and sometimes unexpected programs from classical proofs. We also generalize previously known results since B in $\forall x \exists^{\mathsf{cl}} y B$ no longer needs to be quantifier-free, but only has to belong to the strictly larger class of *goal formulas* defined in Section 6.2. Furthermore we allow unproven lemmas D in the proof of $\forall x \exists^{\mathsf{cl}} y B$, where D is a *definite* formula (also defined in Section 6.2).

Other interesting examples of program extraction from classical proofs have been studied by Murthy [**29**], Coquand's group (see e.g. [**14**]) in a type theoretic context and by Kohlenbach [**23**] using a Dialectica interpretation.

There is also a different line of research aimed at giving an algorithmic interpretation to (specific instances of) the classical double negation rule. It essentially started with Griffin's observation [**21**] that Felleisen's control operator $\mathcal{C}$ [**16, 17**] can be given the type of the stability scheme $\neg\neg A \to A$. This initiated quite a bit of work aimed at extending the Curry-Howard correspondence to classical logic, e.g. by Barbanera and Berardi [**2**], Constable and Murthy [**12**], Krivine [**24**] and Parigot [**31**].

We now describe in more detail what the chapter is about. In Section 6.1 we fix our version of intuitionistic arithmetic for functionals, and recall how classical arithmetic can be seen as a subsystem. Then our argument goes as follows. It is well known that from a derivation of a classical existential formula $\exists^{\mathsf{cl}} y A := (\forall y.A \to \bot) \to \bot$ one generally cannot read off an instance. A simple example has been given by Kreisel: Let R be a primitive recursive relation such that $\exists^{\mathsf{cl}} z R(x,z)$ is undecidable. Clearly we have – even logically –

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

But there is no computable f satisfying

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

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

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

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

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

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

$$\begin{aligned}
\mathsf{Ind}_{n,A}\colon &\quad A[n{:=}0] \to (\forall n.A \to A[n{:=}n+1]) \to \forall nA\\
\mathsf{Ind}_{p,A}\colon &\quad A[p{:=}\mathsf{tt}] \to A[p{:=}\mathsf{ff}] \to \forall pA\\
\mathsf{Ax}_{\mathsf{true}}\colon &\quad \mathsf{atom}(\mathsf{tt})\\
\mathsf{Efq}_A\colon &\quad \mathsf{atom}(\mathsf{ff}) \to A
\end{aligned}$$

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

$$\vdash_{\mathrm{i}} \vec{D} \to D_i[\bot{:=}\exists yG], \tag{36}$$

$$\vdash_{\mathrm{i}} G[\bot{:=}\exists yG] \to \exists yG; \tag{37}$$

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

$$\vdash_{\mathrm{i}} \vec{D}[\bot{:=}\exists yG] \to (\forall y.G[\bot{:=}\exists yG] \to \exists yG) \to \exists yG.$$

Now (36) allows to drop the substitution in $\vec{D}$, and by (37) the second premise is derivable. Hence we obtain as desired

$$\vdash_{\mathrm{i}} \vec{D} \to \exists yG.$$

We shall identify classes of formulas – to be called *definite* and *goal* formulas – such that slight generalizations of (36) and (37) hold. This will be done in Section 6.2.

We will also give (in Section 6.4) an explicit and useful representation of the program term extracted (by the well-known modified realizability interpretation, cf. [**35**]) from the derivation M of $\vec{D} \to \exists yG$ just constructed. The program term has the form $pt_1 \dots t_n s$, where p is extracted from M and $t_1, \dots, t_n, s$ are determined by the formulas $\vec{D}$ and G only.

Since the constructive existential quantifier $\exists$ only enters our derivation in the context $\exists yG$, it is easiest to replace this formula everywhere by a new propositional symbol X and stipulate that a term r realizes X iff $G[y{:=}r]$. This allows for a short and self-contained exposition – in Section 6.3 – of all

we need about modified realizability, including the soundness theorem. In Section 6.4 we then prove our main theorem about program extraction from classical proofs.

The final Section 6.5 then contains some examples of our general machinery. From a classical proof of the existence of the Fibonacci numbers we extract in 6.5.1 a short and surprisingly efficient program, where λ-expressions rather than pairs are passed. In 6.5.2 we treat as a further example a classical proof of the wellfoundedness of $<$ on $\mathbb{N}$. Finally in 6.5.4 we take up a suggestion of Veldman and Bezem [**36**] and present a short classical proof of (the general form of) Dickson's Lemma, as an interesting candidate for further study.

6.1. Arithmetic for Functionals

We restrict the formal system introduced in Section 2.4 to the predicate constant atom and the predicate variables $\bot$ and moreover a special nullary predicate variable X. So *formulas* are

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

As *axioms* we take the induction schemes $\mathsf{Ind}_{n,A}$ and $\mathsf{Ind}_{p,A}$ for the ground types $\mathbf{N}$ and $\mathbf{B}$; for simplicity we only consider these two algebras in the present chapter. In addition we have the axioms

$$\begin{array}{rll} \mathsf{Ax}_{\mathsf{true}} \colon \mathsf{atom}(\mathtt{tt}) & & \text{truth axiom} \\ \mathsf{Efq}_A \colon \mathsf{atom}(\mathtt{ff}) \to A & & \text{ex-falso-quodlibet for } \mathsf{atom}(\mathtt{ff}) \\ \mathsf{Efq\text{-}Log}_A \colon \bot \to A & & \text{ex-falso-quodlibet for } \bot \end{array}$$

Let Z^X denote this system of intuitionistic arithmetic; Z is obtained from Z^X by omitting X. Z_0 (Z_0^X, resp.) is Z (Z^X, resp.) without the axioms $\mathsf{Efq\text{-}Log}_A$. For every Z_0-derivation M let M^X denote the Z_0^X-derivation resulting from M by substituting X for $\bot$. Write $C^X := C[\bot{:=}X]$. – $\mathcal{L}[X]$ ($\mathcal{L}$, resp.) denotes the language of Z^X (Z, resp.). We use P for atomic $\mathcal{L}$-formulas and A, B, C, D, G for $\mathcal{L}[X]$-formulas. $\vdash$ denotes derivability in minimal logic.

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

Lemma 6.1.1. *Let* $F := \mathsf{atom}(\mathtt{ff})$ *and* $A^F := A[\bot{:=}F]$. *Then*

$$Z^X \vdash A \quad \Longleftrightarrow \quad Z_0^X \vdash A^F.$$

Proof. $\Rightarrow$ holds since $(\mathsf{Efq\text{-}Log}_A)^F$ is Efq_{A^F}.

$\Leftarrow$. We have $Z^X \vdash \bot \leftrightarrow F$ by $\mathsf{Efq\text{-}Log}_F \colon \bot \to F$ and $\mathsf{Efq}_\bot \colon F \to \bot$. This implies the claim. □

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

Lemma 6.1.2 (Stability). $Z \vdash \neg\neg A \to A$ *for every* $\mathcal{L}$*-formula* A.

PROOF. Induction on A.

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

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

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

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

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

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

This concludes the proof. □

LEMMA 6.1.3 (Cases). *$Z^X \vdash (\neg C \to A) \to (C \to A) \to A$ for every quantifier-free $\mathcal{L}$-formula C.*

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

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

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

6.2. Definite and Goal Formulas

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

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

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

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

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

$| \forall x D.$

LEMMA 6.2.1. *For definite formulas D and goal formulas G we have*

(38) $Z^X \vdash (\neg D \to X) \to D^X$ *for D relevant,*

(39) $Z^X \vdash D \to D^X$,

(40) $Z^X \vdash G^X \to G$ *for G irrelevant,*

(41) $Z^X \vdash G^X \to (G \to X) \to X.$

PROOF. We prove all four claims (38) – (41) simultaneously, by induction on formulas.

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

Case $G \to D$.

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

Here we have used the induction hypotheses (38) for D and (41) for G.

Case $\forall x D$.

$$\dfrac{\dfrac{(\neg D \to X) \to D^X \qquad \dfrac{\dfrac{\neg \forall x D \to X \qquad \dfrac{\dfrac{\forall x D}{D} \quad \neg D}{\bot}}{\neg \forall x D}}{X}}{\neg D \to X}}{D^X}}{\dfrac{D^X}{\forall x D^X}}{(\neg \forall x D \to X) \to \forall x D^X}$$

Here we have used the induction hypothesis (38) for D.

(39). *Case* D relevant.

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

Here we have used (38) and $\bot \to X$.

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

$$\dfrac{\dfrac{\dfrac{}{D \to D^X} \quad \dfrac{\dfrac{|}{G \to D} \quad \dfrac{\dfrac{}{G^X \to G} \quad \dfrac{}{G^X}}{G}}{D}}{D^X}}{(G \to D) \to G^X \to D^X}$$

Here we have used the induction hypotheses (40) for G and (39) for D.

Subcase $\forall x D$. By the induction hypothesis (39) for D we have $D \to D^X$, which clearly implies $\forall x D \to \forall x D^X$.

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

Case $D \to G$.

$$\dfrac{\dfrac{\dfrac{}{G^X \to G} \quad \dfrac{\dfrac{|}{D^X \to G^X} \quad \dfrac{\dfrac{}{D \to D^X} \quad \dfrac{}{D}}{D^X}}{G^X}}{G}}{(D^X \to G^X) \to D \to G}$$

Here we have used the induction hypotheses (40) for G and (39) for D.

Case $\forall x G$.

$$\dfrac{\dfrac{\dfrac{|}{G^X \to G} \quad \dfrac{\dfrac{}{\forall x G^X}}{G^X}}{G}}{\forall x G}}{\forall x G^X \to \forall x G}$$

Here we have used the induction hypothesis (40) for G.

(41). We may assume that G is relevant, for otherwise the claim clearly follows from (40). *Case* $\bot$. Obvious, since $\bot^X = X$.

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

$$\dfrac{\dfrac{\dfrac{|}{G^X \to (G \to X) \to X} \quad \dfrac{D^X \to G^X \quad D^X}{G^X}}{(G \to X) \to X} \quad \dfrac{(D \to G) \to X \quad \dfrac{\dfrac{G}{D \to G}}{X}}{G \to X}}{X}}{D^X \to X}$$

(using the induction hypothesis (41) for G) and $\mathcal{D}_2[(D \to G) \to X]$ be

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

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

Subcase D relevant.

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

Here we have used the induction hypothesis (38) for D.

Subcase D irrelevant. Then D is quantifier-free. We use case distinction on D from Lemma 6.1.3, in the form $(D\to X)\to(\neg D\to X)\to X$. So it suffices to derive from $D^X\to G^X$ and $(D\to G)\to X$ both premises; recall that our goal was $(D^X\to G^X)\to((D\to G)\to X)\to X$. The negative case is provided by $\mathcal{D}_2[(D\to G)\to X]$, and the positive case by

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

Here we have used the induction hypothesis (39) for D. ∎

Lemma 6.2.2. *For goal formulas* $\vec{G}=G_1,\dots,G_n$ *we have*

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

Proof. By Lemma 6.2.1(41) we have

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

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

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

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

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

Theorem 6.2.3. *Assume that for definite formulas* $\vec{D}$ *and goal formulas* $\vec{G}$ *we have*

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

Then we also have

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

In particular, substitution of the formula

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

for X *yields*

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

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

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

Now by Lemma 6.2.1(39) we can drop X in $\vec{D}^X$, and by Lemma 6.2.2 also in $\vec{G}^X$.

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

The theorem can be viewed in the standard way to yield a method for program extraction from classical proofs. However, in Section 6.4 we give a finer analysis of the extracted programs, and an explanation of the role of definite and goal formulas.

EXAMPLE 6.2.4. Let us check the mechanism of working with definite and goal formulas for Kreisel's "non-example" given in the introduction. There we gave a trivial proof in classical logic of a $\forall\exists^{\mathsf{cl}}$-formula that cannot be realized by a computable function, and we better make sure that our general result also does not provide such a function. The example amounts to a proof in minimal logic of

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

Here $R(x,y) \to \forall z R(x,z)$ is a goal formula, but the premise $\forall z.\neg\neg R(x,z) \to R(x,z)$ is *not* definite. Replacing R by $\neg S$ (to get rid of the stability assumption) does not help, for then $\neg S(x,y) \to \forall z \neg S(x,z)$ is *not* a goal formula. A third possibility would be to use the fact that R is primitive recursive and write $\mathsf{atom}(rxy)$ instead of $R(x,y)$. However, then $\big(\forall y.(\mathsf{atom}(rxy) \to \forall z\, \mathsf{atom}(rxz)) \to \bot\big) \to \bot$ could only be proved in Z, *not* in Z_0 as required in Theorem 6.2.3.

How to obtain definite and goal formulas. To apply these results we have to know that our assumptions are definite formulas and our goal is given by goal formulas. For quantifier-free formulas this clearly can always be achieved by inserting double negations in front of every atom (cf. the definitions of definite and goal formulas). This corresponds to the original (unrefined) so-called A-translation of Friedman [**18**] (or Leivant [**25**]). However, in order to obtain reasonable programs which do not unneccessarily use higher types or case analysis we want to insert double negations only at as few places as possible.

We describe a more economical general way to obtain definite and goal formulas, following [**7, 10**]. It consists in singling out some predicate symbols as being "critical", and then double negating only the atoms formed with critical predicate symbols; call these *critical* atoms.

Assume we have a proof in minimal arithmetic Z_0 of

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

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

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

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

(i) $\bot$ is critical.

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

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

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

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

6.3. Program Extraction

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

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

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

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

We also need extracted programs for the axioms.

$$[\![\mathsf{Ind}_{p,A}]\!] := \mathcal{R}^\tau_{\mathbf{B}} \colon \tau \to \tau \to \mathbf{B} \to \tau \qquad \text{with } \tau := \tau(A) \neq \varepsilon,$$
$$[\![\mathsf{Ind}_{n,A}]\!] := \mathcal{R}^\tau_{\mathbf{N}} \colon \tau \to (\mathbf{N} \to \tau \to \tau) \to \mathbf{N} \to \tau \qquad \text{with } \tau := \tau(A) \neq \varepsilon,$$

$[\![\mathsf{Efq\text{-}Log}_X]\!] := \mathsf{dummy}^\nu$

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

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

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

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

THEOREM 6.3.1 (Soundness). *Assume that M is a Z^X-derivation of B. Then there is a Z-derivation of $[\![M]\!]\ \mathbf{r}_\mathcal{A}\ B$ from the assumptions*

$$\{\, x_u^{\tau(C)}\ \mathbf{r}_\mathcal{A}\ C \mid u^C \in \mathsf{FA}(M) \,\}.$$

PROOF. Induction on M. *Case* $\mathsf{Ind}_{n,A}$. Take $\mathcal{R}^\tau_\mathbf{N}$. *Case* $\mathsf{Ind}_{p,A}$. Take $\mathcal{R}^\tau_\mathbf{B}$. *Case* $\mathsf{Efq\text{-}Log}_A \colon \bot \to A$. Then

$$[\![\mathsf{Efq\text{-}Log}_A]\!]\ \mathbf{r}_\mathcal{A}\ (\bot \to A) = \bot \to [\![\mathsf{Efq\text{-}Log}_A]\!]\ \mathbf{r}_\mathcal{A}\ A,$$

which is an instance of the same axiom scheme. The inductive steps are straightforward. □

6.4. Computational Content of Classical Proofs

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

THEOREM 6.4.1. *Let $\vec{D} = D_1, \dots, D_n$ and $\vec{G} = G_1, \dots, G_m$ be arbitrary $\mathcal{L}$-formulas. Assume that we have terms $t_1, \dots, t_n, s_1, \dots, s_m, r$ such that*

(42) $Z \vdash \vec{D} \to t_j\ \mathbf{r}_\mathcal{A}\ D_j^X$ *for* $1 \le j \le n$,

(43) $Z \vdash \vec{D} \to w_i\ \mathbf{r}_\mathcal{A}\ G_i^X \to (G_i \to \mathcal{A}(v_i)) \to \mathcal{A}(s_i w_i v_i)$ *for* $1 \le i \le m$,

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

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

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

Then

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

PROOF. From the Z_0-derivation M we obtain by the substitution $\bot \mapsto X$ a Z_0^X-derivation $M^X \colon \vec{D}^X \to (\forall \vec{y}.\vec{G}^X \to X) \to X$. The Soundness Theorem 6.3.1 yields

$$\begin{aligned}
&[\![M^X]\!]\, \mathbf{r}_{\mathcal{A}}\, \big(\vec{D}^X \to (\forall \vec{y}.\vec{G}^X \to X) \to X\big) \\
&= \forall \vec{u} \forall v.\vec{u}\, \mathbf{r}_{\mathcal{A}}\, \vec{D}^X \to (v\, \mathbf{r}_{\mathcal{A}}\, \forall \vec{y}.\vec{G}^X \to X) \to \mathcal{A}([\![M^X]\!]\vec{u}v) \\
&= \forall \vec{u} \forall v.\vec{u}\, \mathbf{r}_{\mathcal{A}}\, \vec{D}^X \to \forall \vec{y} \forall \vec{w}\big(\vec{w}\, \mathbf{r}_{\mathcal{A}}\, \vec{G}^X \to \mathcal{A}(v\vec{y}\vec{w})\big) \to \mathcal{A}([\![M^X]\!]\vec{u}v). \qquad (45)
\end{aligned}$$

Instantiate (45) with $\vec{t}$ for $\vec{u}$ and s for v. Clearly $\vec{t}\, \mathbf{r}_{\mathcal{A}}\, \vec{D}^X$ is derivable from $\vec{D}$ by (42), so it remains to show $\vec{D} \to \vec{w}\, \mathbf{r}_{\mathcal{A}}\, \vec{G}^X \to \mathcal{A}(s\vec{y}\vec{w})$.

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

(46)
$$\vec{D} \to G_1 \to \dots \to G_i \to w_{i+1}\, \mathbf{r}_{\mathcal{A}}\, G^X_{i+1} \to \dots \to w_m\, \mathbf{r}_{\mathcal{A}}\, G^X_m \to \mathcal{A}(a_{i+1}).$$

Basis. For $j = 0$ we have $i = m$ and (46) holds by (44). *Step.* From the IH (46) and the assumption (43) we obtain

$$\vec{D} \to G_1 \to \dots \to G_{i-1} \to w_i\, \mathbf{r}_{\mathcal{A}}\, G^X_i \to \dots \to w_m\, \mathbf{r}_{\mathcal{A}}\, G^X_m \to \mathcal{A}(s_i w_i a_{i+1}).$$

For $j = m$ we have $i = 0$ and hence we obtain from (46)

$$\vec{D} \to w_1\, \mathbf{r}_{\mathcal{A}}\, G^X_1 \to \dots \to w_m\, \mathbf{r}_{\mathcal{A}}\, G^X_m \to \mathcal{A}(a_1),$$

which was to be shown. □

In order to apply Theorem 6.4.1, we need $\mathcal{A} = \{\, y \mid A_0 \,\}$ and terms t_j, s_i, r such that (42)–(44) hold. The choice of $\mathcal{A}$ and r of course depends on the application at hand and should be done such that (44) holds. The rest follows from Lemma 6.2.1 by the Soundness Theorem 6.3.1:

THEOREM 6.4.2. *For every definite formula D and goal formula G we have terms t, s such that for an arbitrary $\mathcal{A} = \{\, y \mid A_0 \,\}$ with an $\mathcal{L}$-formula A_0:*

$$Z \vdash D \to t\, \mathbf{r}_{\mathcal{A}}\, D^X, \qquad (47)$$

$$Z \vdash w\, \mathbf{r}_{\mathcal{A}}\, G^X \to (G \to \mathcal{A}(v)) \to \mathcal{A}(swv) \qquad (48)$$

PROOF. (47). Let N_D be the Z^X-derivation of $D \to D^X$ provided by Lemma 6.2.1(39). The Soundness Theorem yields

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

(48). Let H_G be the Z^X-derivation of $G^X \to (G \to X) \to X$ from Lemma 6.2.1(41). By the Soundness Theorem

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

□

For the next corollary we use the notation $\langle\vec{r}\rangle$ for pairing of a non-empty list $\vec{r}$ of terms, and $r0$, $r1$, …, $r(k-1)$ for the projections.

COROLLARY 6.4.3. *Let $\vec{D} = D_1, \ldots, D_n$ be definite formulas and $\vec{G} = G_1, \ldots, G_m$ be goal formulas. Let $M\colon \vec{D} \to (\forall\vec{y}.\vec{G} \to \bot) \to \bot$ be a Z_0-derivation. Then*

$$Z \vdash \vec{D} \to G_i[y_1, \ldots, y_k{:=}[\![M^X]\!]t_1 \ldots t_n s0, \ldots, [\![M^X]\!]t_1 \ldots t_n s(k-1)],$$

with

$$s := \lambda\vec{y}\lambda\vec{w}.s_1 w_1(\ldots(s_m w_m\langle\vec{y}\rangle)\ldots)$$

and $t_1, \ldots, t_n, s_1, \ldots s_m$ determined by the formulas $\vec{D}, \vec{G}$ only, according to Theorem 6.4.2.

PROOF. Let ν be the product of the types of $\vec{y} = y_1, \ldots, y_k$, and y a variable of type ν. We use Theorem 6.4.1 with

$$\mathcal{A} := \{\, y^\nu \mid \bigwedge_i G_i[y_1, \ldots, y_k{:=}y0, \ldots, y(k-1)] \,\}.$$

Then (44) holds with $r := \lambda\vec{y}\langle\vec{y}\rangle$. Moreover, the conclusion

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

clearly yields the claim. □

6.5. Examples

We now want to give some simple examples of how to apply Theorems 6.4.1 and 6.4.2. Here we will always have a single goal formula G and $\mathcal{A}$ will always be chosen as $\{\, y \mid G \,\}$. Hence (44) trivially holds with $r := \lambda yy$.

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

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

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

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

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

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

Clearly the assumption formulas are definite and $G(n,k)$ is a goal formula. So Theorems 6.4.1 and 6.4.2 can be applied without inserting double negations.

To construct a derivation, assume

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

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

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

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

The derivation term is

$$\begin{aligned} M =& \lambda v_0^{G(0,0)} \lambda v_1^{G(1,1)} \lambda v_2^{\forall n \forall k \forall l.G(n,k) \to G(n+1,l) \to G(n+2,k+l)} \lambda u^{\forall k.G(n,k) \to \bot} \\ & \mathsf{Ind}_{n,B} M_{\text{base}} M_{\text{step}} n (\lambda k \lambda l \lambda u_1^{G(n,k)} \lambda u_2^{G(n+1,l)}.uku_1) \end{aligned}$$

where

$$\begin{aligned} M_{\text{base}} =& \lambda w_0^{\forall k \forall l.G(0,k) \to G(1,l) \to \bot}.w_0 01 v_0 v_1 \\ M_{\text{step}} =& \lambda n \lambda w^B \lambda w_1^{\forall k \forall l.G(n+1,k) \to G(n+2,l) \to \bot}. \\ & w(\lambda k \lambda l \lambda u_3^{G(n,k)} \lambda u_4^{G(n+1,l)}.w_1 l(k+l) u_4 (v_2 k l u_3 u_4)). \end{aligned}$$

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

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

where

$$\begin{aligned} [\![M^X_{\text{base}}]\!] &= \lambda w_0^{\mathbf{N} \to \mathbf{N} \to \mathbf{N}}.w_0 01 \\ [\![M^X_{\text{step}}]\!] &= \lambda n \lambda w^{(\mathbf{N} \to \mathbf{N} \to \mathbf{N}) \to \mathbf{N}} \lambda w_1^{\mathbf{N} \to \mathbf{N} \to \mathbf{N}}.w(\lambda k \lambda l.w_1 l(k+l)) \end{aligned}$$

Since there are no relevant formulas involved, the extracted term according to Theorem 6.4.1 is

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

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

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

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

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

To remove some of the tedium of doing all that by hand, we certainly want machine help. We have done such an implementation within our system

MINLOG; here is the original printout of the normalized extracted term, with only some indentation added.

```
[n0](Rec nat=>(nat=>nat=>nat)=>nat)
    ([f1]f1 0 1)
    ([n1,H2,f3]
     H2([n4,n5]f3 n5(n4+n5)))
```

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

6.5.2. Wellfoundedness of $\mathbb{N}$. An interesting phenomenon can occur when we extract a program from a classical proof which uses the minimum principle. Consider as a simple example the wellfoundedness of $<$ on $\mathbb{N}$, i.e.

$$\forall f^{\mathbf{N}\to\mathbf{N}}\exists^{\mathsf{cl}}k.f(k+1) < f(k) \to \bot.$$

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

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

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

$$\exists^{\mathsf{cl}}k\, R(k) \to \exists^{\mathsf{cl}}k.R(k) \wedge \forall l.l < k \to R(l) \to \bot \tag{49}$$

is to be applied with $R(k) := k \in \mathsf{ran}(f)$. Now (49) is logically equivalent to

$$\forall k\big(R(k) \to (\forall l.l < k \to R(l) \to \bot) \to \bot\big) \to \forall k.R(k) \to \bot. \tag{50}$$

The premise of (50) expresses the "progressiveness" of $R(k) \to \bot$ w.r.t. $<$; we abbreviate it to

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

We prove (50) by zero-successor-induction on n w.r.t. the formula

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

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

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

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

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

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

Now a complete formalization is easy. We express $m \le k$ by $k < m \to \bot$ and take $\forall m\, f(m) \neq k$ instead of $R(k) \to \bot$. The derivation term is

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

where

$$\begin{aligned} M_{\mathrm{cvind}} &= \lambda w_1^{\mathsf{Prog}} \lambda k.\mathsf{Ind}_{n,B} M_{\mathrm{base}} M_{\mathrm{step}}(k+1)kL^{k<k+1}, \\ M_{\mathrm{base}} &= \lambda k\lambda w_0^{k<0}\lambda m\lambda \tilde{w}_0^{f(m)=k}.v_1 k w_0, \\ M_{\mathrm{step}} &= \lambda n\lambda w_2^B \lambda k\lambda w_3^{k<n+1}.w_1 k(\lambda l\lambda w_4^{l<k}.w_2 l(L^{l<n}[w_4,w_3])), \\ M_{\mathrm{prog}} &= \lambda k\lambda u_1^{\forall l.l<k\to\forall m f(m)\neq l}\lambda m\lambda u_2^{f(m)=k}.um\lambda w_5^{f(m+1)<f(m)}. \\ & \quad u_1(f(m{+}1))L^{f(m+1)<k}[w_5,u_2](m{+}1)L^{f(m+1)=f(m+1)} \end{aligned}$$

Here we have used the abbreviations

$$\begin{aligned} \mathsf{Prog} &= \big(\forall k.\forall l.l<k\to\forall m\, f(m)\neq l\big)\to\forall m\, f(m)\neq k \\ B &= \forall k.k<n\to\forall m\, f(m)\neq k \end{aligned}$$

For program extraction let

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

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

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

where

$$\begin{aligned} [\![M^X_{\mathrm{cvind}}]\!] &= \lambda w_1^{\mathbf{N}\to(\mathbf{N}\to\mathbf{N}\to\mathbf{N})\to\mathbf{N}\to\mathbf{N}}\lambda k.\mathcal{R}^{\mathbf{N}\to\mathbf{N}\to\mathbf{N}}_{\mathbf{N}}[\![M^X_{\mathrm{base}}]\!][\![M^X_{\mathrm{step}}]\!](k+1)k \\ [\![M^X_{\mathrm{base}}]\!] &= \lambda k\lambda m.v_1 k \\ [\![M^X_{\mathrm{step}}]\!] &= \lambda n\lambda w_2^{\mathbf{N}\to\mathbf{N}\to\mathbf{N}}\lambda k.w_1 k(\lambda l.w_2 l), \\ [\![M^X_{\mathrm{prog}}]\!] &= \lambda k\lambda u_1^{\mathbf{N}\to\mathbf{N}\to\mathbf{N}}\lambda m.um(u_1(f(m+1))(m+1)). \end{aligned}$$

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

Now by (47) we generally have $D \to [\![N_D]\!]\ \mathbf{r}_{\mathcal{A}}\ D^X$ for every relevant definite formula D. In our case for $D = \forall k.k<0\to\bot$ we clearly can derive directly

$$(\forall k.k<0\to\bot)\to(\lambda n0)\ \mathbf{r}_{\mathcal{A}}\ \forall k.k<0\to X,$$

since we can use Efq_X. So we may assume $[\![N_D]\!] = \lambda n0$. Also, by (48) we generally have

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

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

$$(f(k+1) < f(k) \to \mathcal{A}(w)) \to ((f(k+1) < f(k) \to \bot) \to \mathcal{A}(v)) \to \mathcal{A}([\textbf{if } f(k+1) < f(k) \textbf{ then } w \textbf{ else } v]).$$

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

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

Then the extracted term according to Theorem 6.4.1 is

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

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

$$[\![M^X_{\mathrm{cvind}}]\!]' =_{\beta\eta} \lambda w_1 \lambda k'.(\lambda k \lambda m 0)(\lambda n \lambda w_2 \lambda k.w_1 k w_2)(k'+1)k',$$
$$[\![M^X_{\mathrm{prog}}]\!]' =_\beta \lambda k \lambda u_1 \lambda m.[\textbf{if } f(m{+}1){<}f(m) \textbf{ then } u_1(f(m{+}1))(m{+}1) \textbf{ else } m]$$

Therefore we obtain as extracted program

$$[\![M^X]\!][\![N_D]\!]s =_\beta \mathcal{R} r_{\mathrm{base}} r_{\mathrm{step}}((f0)+1)(f0)0$$

with

$$r_{\mathrm{base}} := \lambda k \lambda m.0,$$
$$r_{\mathrm{step}} := \lambda n \lambda w_2^{\mathbf{N}\to\mathbf{N}\to\mathbf{N}} \lambda k \lambda m. [\textbf{if } f(m{+}1){<}fm \textbf{ then } w_2(f(m{+}1))(m{+}1) \textbf{ else } m].$$

Since the recursion argument $(f0)+1$ is a successor, we can convert this term into

$$[\textbf{if } f1 < f0 \textbf{ then } \mathcal{R} r_{\mathrm{base}} r_{\mathrm{step}}(f0)(f1)1 \textbf{ else } 0].$$

To make this algorithm more readable we may define

$$h(0,k,m) = 0$$
$$h(n+1,k,m) = [\textbf{if } f(m+1) < fm \textbf{ then } h(n, f(m+1), m+1) \textbf{ else } m]$$

and then write the result as $h(f(0)+1, f(0), 0)$, or (unfolded) as

$$[\textbf{if } f(1) < f(0) \textbf{ then } h(f(0), f(1), 1) \textbf{ else } 0].$$

The machine extracted program is (original output of MINLOG, with renaming of variables and indentation done manually)

```
[f][if (f 1<f 0)
       ((Rec 2 nat=>nat nat=>nat nat=>nat=>nat=>nat)([n]n)f
        ([k,m]0)
        ([n,g,k,m]
         [if (f(Succ m)<f m)
             (g(f(Succ m))(Succ m))
             m])
        (f 0)
        (f 1)
        1)
       0]
```

We can rewrite this as a SCHEME program as follows.

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

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

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

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

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

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

REMARK 6.5.1. The assumption $v_1 \colon \forall m.m < 0 \to \bot$ used in the proof above is only there for "didactical resons": it serves as an example of how to treat definite formulas. We can of course omit it and use $\mathsf{Efq}_\bot$ instead.

REMARK 6.5.2. There is an alternative proof of the wellfoundedness of $\mathbf{N}$, which uses the minimum principle with a *measure function* instead. Here we can take the function f itself as a measure function. We refrain from analyzing this proof in the same detail as before, but rather present the machine extracted program, which in fact is slightly simpler.

```
[f][if (f 1<f 0)
       ((Rec 1 nat=>nat nat=>nat=>nat)f
        ([m]0)
        ([n,f1,m][if (f(Succ m)<f m) (f1(Succ m)) m])
        (f 0)
        1)
       0]
```

6.5.3. The *hsh*-Theorem. Let f, g, h, s denote unary functions on the natural numbers. We show $\exists^{\mathsf{cl}} n\, h(s(hn)) \neq n$ and extract an (unexpected) program from it (this example is due to Ulrich Berger).

LEMMA 6.5.3 (Surjectivity). *$g \circ f$ surjective implies g surjective.* □

LEMMA 6.5.4 (Injectivity). *$g \circ f$ injective implies f injective.* □

LEMMA 6.5.5 (Surjectivity-Injectivity). *$g \circ f$ surjective and g injective implies f surjective.*

PROOF. Assume y is not in the range of f. Consider $g(y)$. Since $g \circ f$ is surjective, there is an x with $g(y) = g(f(x))$. The injectivity of g implies $y = f(x)$, a contradiction. □

THEOREM 6.5.6 (hsh-Theorem). $\forall n\, s(n) \neq 0 \to \neg\forall n\, h(s(h(n))) = n$.

PROOF. Assume $h \circ s \circ h$ is the identity. Then by the Injectivity Lemma h is injective. Hence by the Surjectivity-Injectivity Lemma $s \circ h$ is surjective, and therefore by the Surjectivity Lemma s is surjective, a contradiction. □

From the Gödel-Gentzen translation and the fact that we can systematically replace triple negations by single negations (cf. 1.4.2) we obtain a derivation of

$$\forall n\, s(n) \neq 0 \to \exists^{\mathsf{cl}} n\, h(s(hn)) \neq n.$$

Now since $\forall n\, s(n) \neq 0$ is a definite formula, this is in the form where our general theory applies. The extracted program is, somewhat unexpectedly,

```
[s,h][if (h(s(h(h 0)))=h 0)
          [if (h(s(h(s(h(h 0)))))=s(h(h 0)))
              0
              (s(h(h 0)))]
          (h 0)]
```

Let us see why this program indeed provides a counterexample against the assumption that $h \circ s \circ h$ is the identity.

If $h(s(h(h0))) \neq h0$, take $h0$. So assume $h(s(h(h0))) = h0$. If

$$h(s(h(s(h(h0))))) = s(h(h0)),$$

then also $h(s(h0)) = s(h(h0))$, so 0 is a counterexample, because the right hand side cannot be 0 (this was our assumption on s). So assume

$$h(s(h(s(h(h0))))) \neq s(h(h0)).$$

Then $s(h(h0))$ is a counterexample.

6.5.4. Towards More Interesting Examples. Veldman and Bezem [**36**] suggested Dickson's Lemma [**15**] as an interesting case study for program extraction from classical proofs. It states that for k given infinite sequences $f_1, \ldots, f_k$ of natural numbers and a given number l there are indices $i_1, \ldots, i_l$ such that *every* sequence f_κ increases on $i_1, \ldots, i_l$, i.e. $f_\kappa(i_1) \leq \cdots \leq f_\kappa(i_l)$ for $\kappa = 1, \ldots, k$. Here is a short classical proof, using the minimum principle for undecidable sets.

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

LEMMA 6.5.7. *Let Q be unbounded and f a function from a superset of Q to $\mathbb{N}$. Then the set Q_f of left f-minima w.r.t. Q is unbounded; here*

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

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

(51)
$$(\exists^{\mathsf{cl}} y.Q(y) \wedge x < y) \to \exists^{\mathsf{cl}} y.Q(y) \wedge x < y \wedge \forall z.Q(z) \to x < z \to f(y) \leq f(z).$$

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

$$Q(y) \land \forall z.Q(z) \to y < z \to f(y) \le f(z).$$

So let z with $Q(z)$ and $y < z$ be given. From $x < y$ we obtain $x < z$, hence $f(y) \le f(z)$ by the conclusion of (51). □

LEMMA 6.5.8. *Let Q be unbounded and $f_1, \dots, f_k$ be functions from a superset of Q to $\mathbb{N}$. Then there is an unbounded subset Q_1 of Q such that $f_1, \dots, f_k$ increase on Q_1, i.e.*

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

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

$$Q_1(x) := Q_2(x) \land \forall y.Q_2(y) \to x < y \to f_1(x) \le f_1(y).$$

By Lemma 6.5.7 Q_1 is an unbounded subset of Q_2. Now on Q_1 f_1 increases, and because of $Q_1 \subseteq Q_2$ also $f_2, \dots, f_k$ increase. □

COROLLARY 6.5.9. *For every k, l we have*

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

For $k = 2$ (i.e. two sequences) this example has been treated in [**11**]. However, it is interesting to look at the general case, since then the brute force search takes time $O(n^k)$, and we can hope that the program extracted from the classical proof is better.

6.6. Notes

This chapter is based on [**8**]. Klaus Weich originally proposed the functional algorithm computing the Fibonacci numbers. Monika Seisenberger – apart from being a coauthor of [**11**] – and Felix Joachimski have contributed a lot to the MINLOG system, particularly to the implementation of the translation of classical proofs into constructive ones. We also benefitted from helpful comments by Peter Selinger and Matteo Slanina, who presented this material in a seminar in Stanford, in the fall of 2000.

Bibliography

1. Andreas Abel and Thomas Altenkirch, *A predicative strong normalization proof for a λ-calculus with interleaving inductive types*, Types for Proofs and Programs, International Workshop, TYPES '99, Lökeberg, Sweden, June 1999, LNCS, vol. 1956, Springer Verlag, Berlin, Heidelberg, New York, 2000, pp. 21–40.
2. Franco Barbanera and Stefano Berardi, *Extracting constructive content from classical logic via control–like reductions*, Typed Lambda Calculi and Applications (M. Bezem and J.F. Groote, eds.), LNCS Vol. 664, 1993, pp. 45–59.
3. Holger Benl, *Konstruktive Interpretation induktiver Definitionen*, Master's thesis, Mathematisches Institut der Universität München, 1998.
4. Holger Benl and Helmut Schwichtenberg, *Formal correctness proofs of functional programs: Dijkstra's algorithm, a case study*, Computational Logic (U. Berger and H. Schwichtenberg, eds.), Series F: Computer and Systems Sciences, vol. 165, Proceedings of the NATO Advanced Study Institute on Computational Logic, held in Marktoberdorf, Germany, July 29 – August 10, 1997, Springer Verlag, Berlin, Heidelberg, New York, 1999, pp. 113–126.
5. Ulrich Berger, *Program extraction from normalization proofs*, Typed Lambda Calculi and Applications (M. Bezem and J.F. Groote, eds.), LNCS, vol. 664, Springer Verlag, Berlin, Heidelberg, New York, 1993, pp. 91–106.
6. ———, *A constructive interpretation of positive inductive definitions*, Draft, March 1995.
7. ———, *Programs from classical proofs*, Symposia Gaussiana. Proceedings of the 2nd Gauss Symposium. Conference A: Mathematics and Theoretical Physics. Munich, Germany, August 2-7, 1993 (Berlin, New York) (M. Behara, R. Fritsch, and R.G. Lintz, eds.), Walter de Gruyter, 1995, pp. 187–200.
8. Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg, *Refined program extraction from classical proofs*, Annals of Pure and Applied Logic **114** (2002), 3–25.
9. Ulrich Berger, Matthias Eberl, and Helmut Schwichtenberg, *Term rewriting for normalization by evaluation*, Information and Computation **183** (2003), 19–42.
10. Ulrich Berger and Helmut Schwichtenberg, *Program extraction from classical proofs*, Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, IN, USA, October 1994 (D. Leivant, ed.), LNCS, vol. 960, Springer Verlag, Berlin, Heidelberg, New York, 1995, pp. 77–97.
11. Ulrich Berger, Helmut Schwichtenberg, and Monika Seisenberger, *The Warshall Algorithm and Dickson's Lemma: Two Examples of Realistic Program Extraction*, Journal of Automated Reasoning **26** (2001), 205–221.
12. Robert L. Constable and Chetan Murthy, *Finding computational content in classical proofs*, Logical Frameworks (G. Huet and G. Plotkin, eds.), Cambridge University Press, 1991, pp. 341–362.
13. Catarina Coquand, *A proof of normalization for simply typed lambda calculus written in ALF*, Proceedings of the 1992 Workshop on Types for Proofs and Programs, Baastad (Bengt Nordstroem, Kent Peterson, and Gordon Plotkin, eds.), June 1992, pp. 80–87.
14. Thierry Coquand and Hendrik Persson, *Gröbner Bases in Type Theory*, Types for Proofs and Programs (T. Altenkirch, W. Naraschewski, and B. Reus, eds.), LNCS, vol. 1657, Springer Verlag, Berlin, Heidelberg, New York, 1999.
15. L.E. Dickson, *Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors*, Am. J. Math **35** (1913), 413–422.

16. Matthias Felleisen, Daniel P. Friedman, E. Kohlbecker, and B.F. Duba, *A syntactic theory of sequential control*, Theoretical Computer Science **52** (1987), 205–237.
17. Matthias Felleisen and R. Hieb, *The revised report on the syntactic theory of sequential control and state*, Theoretical Computer Science **102** (1992), 235–271.
18. Harvey Friedman, *Classically and intuitionistically provably recursive functions*, Higher Set Theory (D.S. Scott and G.H. Müller, eds.), Lecture Notes in Mathematics, vol. 669, Springer Verlag, Berlin, Heidelberg, New York, 1978, pp. 21–28.
19. Gerhard Gentzen, *Untersuchungen über das logische Schließen*, Mathematische Zeitschrift **39** (1934), 176–210, 405–431.
20. Jean-Yves Girard, *Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types*, Proceedings of the Second Scandinavian Logic Symposium (J.E. Fenstad, ed.), North–Holland, Amsterdam, 1971, pp. 63–92.
21. Timothy G. Griffin, *A formulae–as–types notion of control*, Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, 1990, pp. 47–58.
22. Gérard Huet, *A unification algorithm for typed λ–calculus*, Theoretical Computer Science **1** (1975), 27–57.
23. Ulrich Kohlenbach, *Analysing proofs in analysis*, Logic: from Foundations to Applications. European Logic Colloquium (Keele, 1993) (W. Hodges, M. Hyland, C. Steinhorn, and J. Truss, eds.), Oxford University Press, 1996, pp. 225–260.
24. Jean-Louis Krivine, *Classical logic, storage operators and second-order lambda-calculus*, Annals of Pure and Applied Logic **68** (1994), 53–78.
25. Daniel Leivant, *Syntactic translations and provably recursive functions*, The Journal of Symbolic Logic **50** (1985), no. 3, 682–688.
26. Per Martin-Löf, *Hauptsatz for the intuitionistic theory of iterated inductive definitions*, Proceedings of the Second Scandinavian Logic Symposium (J.E. Fenstad, ed.), North–Holland, Amsterdam, 1971, pp. 179–216.
27. Dale Miller, *A logic programming language with lambda–abstraction, function variables and simple unification*, Journal of Logic and Computation **2** (1991), no. 4, 497–536.
28. Grigori Mints, *A short introduction to intuitionistic logic*, Kluwer Academic/Plenum Publishers, New York, 2000.
29. Chetan Murthy, *Extracting constructive content from classical proofs*, Technical Report 90–1151, Dep.of Comp.Science, Cornell Univ., Ithaca, New York, 1990, PhD thesis.
30. Tobias Nipkow, *Higher-order critical pairs*, Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science (Los Alamitos) (R. Vemuri, ed.), IEEE Computer Society Press, 1991, pp. 342–349.
31. Michel Parigot, *$\lambda\mu$–calculus: an algorithmic interpretation of classical natural deduction*, Proc. of Log. Prog. and Automatic Reasoning, St. Petersburg, LNCS, vol. 624, Springer Verlag, Berlin, Heidelberg, New York, 1992, pp. 190–201.
32. Christine Paulin-Mohring and Benjamin Werner, *Synthesis of ML programs in the system Coq*, J. Symbolic Computation **11** (1993), 1–34.
33. Monika Seisenberger, *On the constructive content of proofs*, Ph.D. thesis, Mathematisches Institut der Universität München, 2003.
34. Viggo Stoltenberg-Hansen, Edward Griffor, and Ingrid Lindström, *Mathematical theory of domains*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1994.
35. Anne S. Troelstra and Dirk van Dalen, *Constructivism in mathematics. an introduction*, sil, vol. 121, 123, North–Holland, Amsterdam, 1988.
36. Wim Veldman and Marc Bezem, *Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics*, Logic Group Preprint Series 72, University of Utrecht, Dept of Philosophy, January 1992.

Index