Extracting Herbrand trees in classical realizability using forcing*

Lionel Rieg

LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA), ENS de Lyon, Université de Lyon
46 allée d'Italie, 69364 LYON, FRANCE
lionel.rieg@ens-lyon.fr

Abstract

Krivine presented in [9] a methodology to combine Cohen's forcing with the theory of classical realizability and showed that the forcing condition can be seen as a reference that is not subject to backtracks. The underlying classical program transformation was then analyzed by Miquel [11] in a fully typed setting in classical higher-order arithmetic (PAω^+).

As a case study of this methodology, we present a method to extract a Herbrand tree from a classical realizer of inconsistency, following the ideas underlying the completeness theorem and the proof of Herbrand's theorem. Unlike the traditional proof based on Kőnig's lemma (using a fixed enumeration of atomic formulas), our method is based on the introduction of a particular Cohen real. It is formalized as a proof in PAω^+, making explicit the construction of generic sets in this framework in the particular case where the set of forcing conditions is arithmetical.
We then analyze the algorithmic content of this proof.

1998 ACM Subject Classification F.4.1 Lambda-calculus and related system

Keywords and phrases classical realizability, forcing, Curry-Howard correspondence, Herbrand trees

Digital Object Identifier 10.4230/LIPIcs.CSL.2013.597

1 Introduction

Forcing is a model transformation initially invented by Cohen [1, 2] to prove the relative consistency of the negation of the continuum hypothesis with respect to the axioms of Zermelo-Fraenkel (ZF) set theory. From a model-theoretic point of view, forcing is a technique to extend a given model of ZF—the *base model*—into a larger model—the *generic extension*—generated around the base model from a new set with good properties: the generic filter G. From a proof-theoretic point of view, forcing can be presented as a logical translation that maps formulas expressing properties of the extended model into formulas expressing (more complex) properties of the base model. Through this translation, the properties of the (fictitious) generic set G (in the extended universe) are reduced to the properties of the forcing poset C (in the base universe) that parametrizes the whole construction.

Recently, Krivine studied [9] Cohen forcing in the framework of the proofs-as-programs correspondence in classical logic [5, 13, 3] and showed how to combine it with the theory of classical realizability [8]. In particular, he discovered a program translation (independent from typing derivations) that captures the computational contents of the logical translation underlying forcing. Surprisingly, this program transformation acts as a *state passing style*

* This work was supported by the ANR project RÉCRÉ.

translation where the forcing condition is treated as a memory cell that is protected from the backtracks performed by control operators such as callcc [5] —thus opening an intriguing connection between forcing and imperative programming. Reformulating this work in classical higher-order arithmetic (PAω^+) and analyzing the corresponding program transformation, Miquel [11, 12] introduced an extension of the Krivine Abstract Machine (KAM) devoted to execution of proofs by forcing—the KFAM—where the forcing condition is explicitly treated as a memory cell in the context of the execution of a proof by forcing.

These analogies naturally suggest that Cohen forcing can be used not only to prove relative consistency results, but also to write computationally more efficient (classical) proofs by exploiting the imperative flavor of the forcing condition.

In this paper, we propose to instantiate this technique on one example, namely the extraction of a Herbrand tree (see section 2) from a validity proof of an existential formula $\exists\vec{x}.\, F(\vec{x})$ where $F(\vec{x})$ is quantifier-free. Our extraction procedure is based on a proof of a mix between completeness and Herbrand's theorem using the method of forcing. The key ingredient of this proof is the introduction of a Cohen real (using forcing) that represents all valuations at once. From a computational point of view, we will see that the corresponding program uses the forcing condition to store the tree under construction, thus protecting it from the backtracks induced by classical reasoning. The interest of this approach is that since the conclusion of our semantic variant of Herbrand's theorem is Σ^0_1, any proof (program) of the translation of the conclusion (through the forcing translation) can be turned into a proof (program) of the conclusion itself. From this, it is then possible to apply standard witness extraction techniques in classical realizability [10] to extract the desired Herbrand tree.

Contribution of the paper

This work follows on from [9] and [11]. Its contributions are the following:

- The extension of the program transformation underlying forcing to a generic filter G (when the forcing sort and its relativization predicate are invariant under forcing).
- A proof of a semantic variant of Herbrand's theorem (containing completeness) by forcing where a Cohen real represents all valuations at once in the forcing universe.
- A formalization of this proof in the formal system PAω^+ which, through the forcing transformation, gives an extraction process for Herbrand trees.
- An analysis of the computational content of this extraction process in classical realizability.

2 Herbrand trees

2.1 The notion of Herbrand tree

In what follows, we work in a given countable first-order language, and write Term and Atom the countable sets of closed terms and of closed atomic formulas, respectively. Throughout this paper we are interested in the following problem.

Let $\exists\vec{x}.\, F(\vec{x})$ be a purely existential formula, where $F(\vec{x})$ is quantifier-tree. Let us now assume that the formula $\exists\vec{x}.\, F(\vec{x})$ is true in all models, and actually in all *syntactic models*, where variables are interpreted by closed terms $t \in \mathrm{Term}$. From this information, we know that there is a function $H : (\mathrm{Atom} \to \mathrm{Bool}) \to \overrightarrow{\mathrm{Term}}$ that associates to every syntactic valuation $\rho : \mathrm{Atom} \to \mathrm{Bool}$ a tuple of closed terms $H(\rho) = \vec{t} \in \overrightarrow{\mathrm{Term}}$ such that $\rho \models F(\vec{t})$ (i.e. a 'witness' for the formula $\exists\vec{x}.\, F(\vec{x})$ in the valuation ρ).

However, the information provided by the function H is twice infinite: it is infinite in depth since each valuation $\rho : \mathrm{Atom} \to \mathrm{Bool}$ is (a priori) infinite, and it is infinite in width since the set of all such valuations has the power of continuum. Nevertheless, the completeness theorem combined with Herbrand's theorem says that we can compact the information given by the a priori infinite function H into a finite binary tree, which is called a *Herbrand tree.*

▶ **Definition 2.1** (Herbrand tree for a formula F)**.** A *Herbrand tree* is a finite binary tree H such that:

- The inner nodes of H are labeled with atomic formulas $a \in \text{Atom}$, so that every branch of the tree represents a partial valuation (going left means 'true', going right means 'false').
- Every leaf of H contains a witness for the corresponding branch, that is a tuple $\vec{t} \in \overrightarrow{\text{Term}}$ s.t. $\rho \models F(\vec{t})$ for every (total) valuation ρ extending that partial valuation of the branch.

▶ **Theorem 2.2.** *If the formula $\exists\vec{x}.\, F(\vec{x})$ is true in all syntactic models, then F has a Herbrand tree.*

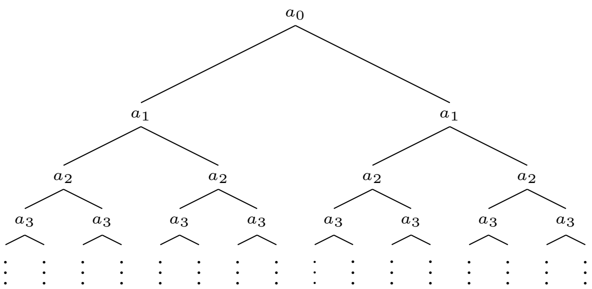
The aim of this paper is to describe a method to effectively extract a Herbrand tree from a proof (actually a classical realizer) of the proposition expressing that 'the formula $\exists\vec{x}.\, F(\vec{x})$ holds in all syntactic models'. Since the latter proposition is directly implied by the formula $\exists\vec{x}.\, F(\vec{x})$ itself (using the trivial implication of the completeness theorem), we will thus get a method to effectively extract a Herbrand tree from a proof/realizer of the formula $\exists\vec{x}.\, F(\vec{x})$.

Note that we will not give a proof of Theorem 2.2 but rather a proof of the validity of the *admissible rule* associated to it, namely: given a proof that 'the formula $\exists\vec{x}.\, F(\vec{x})$ holds in all syntactic models', we can build a proof of existence of a Herbrand tree for F. This statement is enough for extraction.

2.2 Extracting Herbrand trees effectively

In the framework of the Curry-Howard correspondence, the natural method to extract Herbrand trees is to use a classical realizer t_0 obtained from a formal proof of Theorem 2.2. By applying t_0 to a realizer u of the premise of Theorem 2.2, we get a realizer of the Σ^0_1-formula expressing the existence of a Herbrand tree for the formula $\exists x.F(\vec{x})$, from which we can retrieve the desired Herbrand tree using standard classical extraction techniques [10].

However, the efficiency of the extracted code highly depends on the proof of Theorem 2.2. In particular, the simplest proof of this theorem (Fig. 1), which relies on a fixed enumeration of all atoms, is not well suited to this task, since it gives terribly poor performances on



Given an enumeration $(a_i)_{i\in\mathbb{N}}$ of the closed instances of the atomic formulas appearing in $F(\vec{x})$, let us consider the infinite binary tree whose 2^i nodes at depth i are labeled with the atom a_i. Any infinite branch in this infinite tree is an valuation ρ, because all atoms appear along it. From our assumption, we know that there is a tuple $\vec{t} \in \overrightarrow{\text{Term}}$ such that $\rho \models F(\vec{t})$. But since the calculation of the truth value of the closed formula $F(\vec{t})$ only relies on a finite subset of ρ, we can cut the branch along ρ at some depth d, putting a leaf labeled with $\vec{t}$. Doing this in all branches simultaneously, we get a finite tree (by the fan theorem), which is by construction a Herbrand tree.

Figure 1 A proof of Theorem 2.2 by enumerating the atoms.

formulas $F(\vec{x})$ involving atoms that appear late in the chosen enumeration. What we want is a proof/realizer of Theorem 2.2 that chooses the atoms labeling the nodes only in function of the realizer of its premise.

In what follows, we present a novel proof of Theorem 2.2 that is tailored for this purpose, and that relies on the forcing techniques developed in [9, 11, 12]. In this case, the forcing condition is a Cohen real which behaves as a generic valuation, *i.e.* it represents all infinite branches at once. As we will see in section 6, it is computationally a scheduler that will extend the tree under construction on request, depending on which atoms are required by the realizer of the premise. It will scan the whole tree and schedule pending branches until the full Herbrand tree is built.

3 The higher-order arithmetic PAω^+

In this section, we recall PAω^+, the formal proof system in which this work takes place. It is a presentation of classical higher-order arithmetic with explicit (classical) proof terms, inspired by Church's theory of simple types. It features an extra congruence on terms, in the spirit of deduction modulo [4]. This section is a summary of the presentation of PAω^+ in [11], to which we refer the reader for more details and proofs of the results stated here.

3.1 Syntax

System PAω^+ distinguishes three kinds of syntactic entities: *sorts* (or *kinds*), *higher-order terms*, and *proof terms*, whose grammar is recalled in Fig 2.

Sorts	$\tau, \sigma ::= \iota \;\mid\; o \;\mid\; \tau \to \sigma$
Higher-order terms	$M, N, A, B ::= x^\tau \;\mid\; \lambda x^\tau .\, M \;\mid\; MN \;\mid\; 0 \;\mid\; S \;\mid\; \mathrm{rec}_\tau$ $\mid A \Rightarrow B \;\mid\; \forall x^\tau .\, A \;\mid\; M \doteq_\tau N \mapsto A$
Proof-terms	$t, u ::= x \;\mid\; \lambda x .\, t \;\mid\; tu \;\mid\; \mathrm{callcc}$

Figure 2 Syntax of PAω^+.

3.1.1 Sorts and higher-order terms

Sorts are simple types formed from the two basic sorts ι (the sort of *individuals*) and o (the sort of *propositions*). Higher-order terms (also called *terms* for short) are simply-typed λ-terms (à la Church) that are intended to represent *mathematical objects* that inhabit sorts.

Higher-order terms of sort ι, which are called *individuals*, are formed using the two constructors 0 (of sort ι), S (of sort $\iota \to \iota$) and the family of recursors rec_τ (of sort $\tau \to (\iota \to \tau \to \tau) \to \iota \to \tau$).

Higher-order terms of sort o, which are called *propositions* (and written A, B, C, etc. in what follows), are formed using implication $A \Rightarrow B$ (where A and B are propositions), universal quantification $\forall x^\tau .\, A$ (where A is a proposition possibly depending on the variable x^τ) and a new connective $M \doteq_\tau N \mapsto A$ called an *equational implication* (where M and N are of sort τ and where A is a proposition). This new connective must be thought of as a kind of implication, but giving more compact proof terms. It makes the computational contents of the forcing translation more transparent, but it is logically equivalent to the usual implication $M =_\tau N \Rightarrow A$, via the proof terms:

$$\lambda xy.\, y\, x \;:\; (M \doteq N \mapsto A) \Rightarrow (M = N \Rightarrow A), \qquad \lambda x.\, x\, (\lambda y.\, y) \;:\; (M = N \Rightarrow A) \Rightarrow (M \doteq N \mapsto A)$$

(See fFg. 4 for a definition of the proof system.)

As usual, application is left associative whereas implication and equational implication are both right associative and have same precedence: $A \Rightarrow M \doteq N \mapsto B \Rightarrow C \Rightarrow D$ has to be read as $A \Rightarrow (M \doteq N \mapsto (B \Rightarrow (C \Rightarrow D)))$. Logical connectives (absurdity, negation, conjunction, disjunction) are defined using the standard second-order encodings, as well as Leibniz equality, letting: $x =_\tau y := \forall Z^{\tau \to o}.\, Z\,x \Rightarrow Z\,y$. Existential quantification (possibly combined with conjunctions) is encoded classically using De Morgan laws: $\exists x^\tau.\, A_1 \,\&\, \ldots \,\&\, A_k := \neg(\forall x^\tau.\, A_1 \Rightarrow \ldots \Rightarrow A_k \Rightarrow \bot)$. We often omit the sort annotation τ to ease reading when this does not hinder understanding. On the opposite, when we want to give explicitly the sort of a term, we write it in exponent, *e.g.* M^τ, A^o, $\mathrm{rec}_\tau^{\tau\to(\iota\to\tau\to\tau)\to\iota\to\tau}$.

3.1.2 System T is a fragment of PAω^+

Gödel's system T can be recovered from PAω^+ as the subsystem where we restrict sorts to be T-sorts, that is sorts built with ι as the only base sort. This constraint casts out all logical constructions and limits the term construction rules exactly to those of system T. Recall that the expressiveness of system T is exactly the functions which are provably total in first-order arithmetic, which includes (and exceeds) all primitive recursive functions.

3.2 Proof system

3.2.1 Congruence

The proof system PAω^+ differs from higher-order arithmetic by the addition of a congruence $\simeq_{\mathcal{E}}$ to the proof system. This allows to reason modulo some equivalence on higher-order terms (hence on propositions) without polluting the proof terms with computationally irrelevant parts.

This congruence contains the usual $\beta\eta\iota$-conversion, some semantic equivalences on propositions (mostly commutations) and an equational theory $\mathcal{E}$. This *equational theory* is a finite set of equations $\mathcal{E} = M_1 = N_1, \ldots, M_k = N_k$, where M_i and N_i are higher-order terms of the same sort (that $\simeq_{\mathcal{E}}$ considers equal). Some rules for the congruence $\simeq_{\mathcal{E}}$ are given in Fig. 3, the full set is given in annex A.

$$\frac{}{M \simeq_{\mathcal{E}} N}\ (M = N) \in \mathcal{E} \qquad\qquad \frac{M \simeq_{\mathcal{E}} N \qquad P \simeq_{\mathcal{E}} Q \qquad A \simeq_{\mathcal{E}, M=P} B}{M \doteq P \mapsto A \simeq_{\mathcal{E}} N \doteq Q \mapsto B}$$

$$\frac{}{M \doteq M \mapsto A \simeq_{\mathcal{E}} A} \qquad\qquad \frac{}{A \Rightarrow M \doteq N \mapsto B \simeq_{\mathcal{E}} M \doteq N \mapsto A \Rightarrow B}$$

$$\frac{}{\forall x^\tau.\, M \doteq N \mapsto A \simeq_{\mathcal{E}} M \doteq N \mapsto \forall x^\tau.\, A}\ x \notin FV(M, N)$$

Figure 3 Some inference rules for the relation $\simeq_{\mathcal{E}}$.

3.2.2 Proof terms and inference rules

Proof terms (Fig. 2) are pure λ-terms enriched with an extra constant callcc; they are formed from a set of *proof variables* (notation: x, y, z, etc.) distinct from higher-order term variables. The deduction system of PAω^+ is defined around a typing judgment of the form $\mathcal{E}; \Gamma \vdash t : A$, where $\mathcal{E}$ is an equational theory and Γ a *context*, that is: a finite set of bindings of distinct proof variables x_i to propositions A_i. The inference rules, given in Fig 4, are the ones of higher-order arithmetic, with slight modifications to deal with the congruence and equational implication.

$$\frac{}{\mathcal{E};\Gamma, x : A \vdash x : A} \qquad \frac{\mathcal{E};\Gamma \vdash t : A}{\mathcal{E};\Gamma \vdash t : A'}\ {\scriptstyle A \simeq_{\mathcal{E}} A'} \qquad \frac{}{\mathcal{E};\Gamma \vdash \mathrm{callcc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

$$\frac{\mathcal{E};\Gamma, x : A \vdash t : B}{\mathcal{E};\Gamma \vdash \lambda x.\, t : A \Rightarrow B} \qquad \frac{\mathcal{E};\Gamma \vdash t : A \Rightarrow B \qquad \mathcal{E};\Gamma \vdash u : A}{\mathcal{E};\Gamma \vdash t\, u : B}$$

$$\frac{\mathcal{E}, M^\tau = N^\tau;\Gamma \vdash t : A}{\mathcal{E};\Gamma \vdash t : M \doteq_\tau N \mapsto A} \qquad \frac{\mathcal{E};\Gamma \vdash t : M \doteq_\tau M \mapsto A}{\mathcal{E};\Gamma \vdash t : A}$$

$$\frac{\mathcal{E};\Gamma \vdash t : A}{\mathcal{E};\Gamma \vdash t : \forall x^\tau.\, A}\ {\scriptstyle x \notin FV(\Gamma)} \qquad \frac{\mathcal{E};\Gamma \vdash t : \forall x^\tau.\, A}{\mathcal{E};\Gamma \vdash t : A[N^\tau/x^\tau]}$$

Figure 4 The inference rules of PAω^+.

▶ Remarks.

1. The only inference rules that alter proof terms are the axiom, Peirce's law, and the introduction and elimination rules of implication. The remaining rules do not affect proof terms and are said to be *computationally transparent*.
2. The proof system of PAω^+ enjoys no normalization property since the proposition $\top$ defined by $\top := \lambda xy.\, x \doteq_o \lambda xy.\, y \mapsto \bot$ acts as a type of all (untyped) proof terms [11, section II.E.3]. (Intuitively, $\top$ allows to equate any two propositions so that they are all equivalent to $\bot$.) Nevertheless, the system is sound with respect to the intended classical realizability semantics (see section 3.4).
3. This proof system allows full classical reasoning thanks to Peirce's law. Arithmetical reasoning (including reasoning by induction) can be recovered by relativizing all quantifications over the sort ι using the predicate $x \in \mathbb{N} := \forall Z^o.\, Z\, 0 \Rightarrow (\forall y^\iota.\, Z\, y \Rightarrow Z\, (S\, y)) \Rightarrow Z\, x$ (see below).

3.3 Sets and datatypes

In PAω^+, a set is given by a sort τ together with a relativization predicate P of sort $\tau \to o$ expressing membership in the set. For instance, the set of total relations between individuals is given by the sort $\iota \to \iota \to o$ and the predicate $\mathrm{Tot} := \lambda R.\, \forall x^\iota.\, \exists y^\iota.\, R\, x\, y$.

Because the sort τ can be inferred from the sort of P, we will identify sets with their relativization predicates. For convenience, we use the suggestive notations $x \in P$ (resp. $\forall x \in P.\, A$, $\exists x \in P.\, A$) for $P\, x$ (resp. $\forall x.\, P\, x \Rightarrow A$, $\exists x.\, x \in P \,\&\, A$). In what follows, *datatypes* will be represented as particular sets based on the sort $\tau \equiv \iota$ and whose relativization predicate P is invariant under forcing (see section 4.2). For instance, the datatypes of Booleans and natural numbers are given by

$$\begin{array}{rcl} x \in \mathrm{Bool} & := & \forall Z^{\iota \to o}.\, Z\, 0 \Rightarrow Z\, 1 \Rightarrow Z\, x \\ x \in \mathbb{N} & := & \forall Z^{\iota \to o}.\, Z\, 0 \Rightarrow \big(\forall y^\iota.\, Z\, y \Rightarrow Z\, (S\, y)\big) \Rightarrow Z\, x \end{array}$$

(The proof of their invariance under forcing is delayed until section 5.2.) We also consider two abstract datatypes Term and Atom representing closed terms and closed atomic formulas (whose exact implementation is irrelevant). More generally, inductive datatypes are defined by implementing constructors as suitable functions from individuals to individuals and by defining the corresponding predicate by well-known second-order encodings. For instance, the datatype of binary trees

$$t, t' := \mathrm{Leaf}\, \vec{v} \quad | \quad \mathrm{Node}\, a\, t\, t' \qquad \text{where } \vec{v} \in \overrightarrow{\mathrm{Term}},\ a \in \mathrm{Atom}$$

is given by two injective functions $\mathrm{Leaf}^{\iota \to \iota}$ and $\mathrm{Node}^{\iota \to \iota \to \iota \to \iota}$ whose ranges do not overlap (the actual implementation is irrelevant here) and the corresponding relativization predicate $t \in \mathrm{Tree}$ is

$$\forall Z^{\iota \to o}.\, (\forall \vec{v} \in \mathrm{Term}.\, Z\,(\mathrm{Leaf}\,\vec{v})) \;\Rightarrow\; (\forall t_1^{\iota} t_2^{\iota}\, a \in \mathrm{Atom}.\, Z\,t_1 \Rightarrow Z\,t_2 \Rightarrow Z\,(\mathrm{Node}\,a\,t_1\,t_2)) \Rightarrow Z\,t\,.$$

We also introduce the inductive datatype Comp of quantifier-free formulas built above Atom:

$$c, c' := \;\perp\!\!\!\perp \quad | \quad a \quad | \quad c \Rrightarrow c' \qquad \text{where } a \in \mathrm{Atom}$$

This presentation based on implication is more suited to classical realizability (see below), but Comp is nothing but the free Boolean algebra generated by Atom.

3.4 Realizability semantics

System PAω^+ has a classical realizability semantics in the spirit of Krivine's [8] that is fully described in [11, 12]. This semantics is based on Krivine's λ_c-calculus (which contains all proof terms of PAω^+) and parametrized by a fixed set of processes (the *pole* of the realizability model). According to this semantics, every (closed) proof term t of a (closed) proposition A is a realizer of A (written $t \Vdash A$), and this independently from the choice of the pole. In the particular case where the pole is empty, the realizability model collapses to a Tarski model of PAω^+, from which we deduce the logical consistency of the system. This classical realizability semantics also provides simple methods to extract witnesses from realizers (and thus from proofs) of Σ_1^0-propositions [10].

4 The Forcing Transformation

4.1 Forcing in PAω^+

This section is a reformulation of Cohen's theory of forcing (developed for ZF set theory) in the framework of PAω^+. Here, we see forcing as a translation of facts about objects living in an *extended universe* (where sorts intuitively contain much more inhabitants) to facts about objects living in the *base universe*. Technically, we will first present forcing as a translation from PAω^+ to itself. But in section 4.3, we will see how to add a generic filter G to PAω^+, so that forcing will be actually a translation from PA$\omega^+ + G$ to PAω^+. We follow here the presentation of [11, 12], where the reader may find all missing proofs.

4.1.1 Definition of a forcing structure

As in [9, 11], we introduce the set of conditions as an upward closed subset C of a meet-semilattice $(\kappa, \cdot, 1)$. (Any poset with a greatest element can be presented in this way.)

▶ **Definition 4.1** (Forcing structure)**.** A *forcing structure* is given by:

- a set $C : \kappa \to o$ of *well-formed forcing conditions* ($p \in C$ being usually written $C[p]$),
- an operation $\cdot$ of sort $\kappa \to \kappa \to \kappa$ to form the *meet* of two conditions (denoted by juxtaposition),
- a greatest condition 1,
- nine closed proof terms representing the axioms that must be satisfied by the forcing structure:

$$\begin{array}{lll}
\alpha_0 : C[1] & \alpha_1 : \forall pq.\, C[pq] \Rightarrow C[p] & \alpha_2 : \forall pq.\, C[pq] \Rightarrow C[q] \\
\alpha_3 : \forall pq.\, C[pq] \Rightarrow C[qp] & \alpha_4 : \forall p.\, C[p] \Rightarrow C[pp] & \alpha_5 : \forall pqr.\, C[(pq)r] \Rightarrow C[p(qr)] \\
\alpha_6 : \forall pqr.\, C[p(qr)] \Rightarrow C[(pq)r] & \alpha_7 : \forall p.\, C[p] \Rightarrow C[p1] & \alpha_8 : \forall p.\, C[p] \Rightarrow C[1p]
\end{array}$$

$$
\begin{array}{|rcl rcl rcl|}
\hline
(\sigma \to \tau)^* & := & \sigma^* \to \tau^* & \iota^* & := & \iota & o^* & := & \kappa \to o \\
\hline
(x^\tau)^* & := & x^{\tau^*} & 0^* & := & 0 & (\forall x^\tau .\, A)^* & := & \lambda r^\kappa .\, \forall x^{\tau^*} .\, A^*\, r \\
\lambda x^\tau .\, M & := & \lambda x^{\tau^*} .\, M^* & S^* & := & S & (M \doteq N \mapsto A)^* & := & \lambda r^\kappa .\, M^* \doteq N^* \mapsto A^*\, r \\
(M\, N)^* & := & M^*\, N^* & \mathrm{rec}_\tau^* & := & \mathrm{rec}_{\tau^*} & (A \Rightarrow B)^* & := & \lambda r^\kappa .\, \forall q^\kappa \forall (r')^\kappa .\, r \doteq qr' \mapsto \\
 & & & & & & & & (\forall s^\kappa .\, C[qs] \Rightarrow A^*\, s) \Rightarrow B^*\, r' \\
\hline
x^* & := & x & (\lambda x.\, t)^* & := & \multicolumn{3}{l|}{\gamma_1 (\lambda x.\, t^* [(\beta_3 y)/y][(\beta_4 x)/x]) \qquad\qquad y \neq x} \\
(t\, u)^* & := & \gamma_3\, t^*\, u^* & \mathrm{callcc}^* & := & \multicolumn{3}{l|}{\lambda cx.\ \mathrm{callcc}(\lambda k.\, x\, (\alpha_{14}\, c)\, (\lambda cy.\, k\, (y\, \alpha_{15}\, c)))} \\
\hline
\end{array}
$$

$$\beta_3 := \lambda xc.\, x\, (\alpha_9\, c) \qquad \beta_4 := \lambda xc.\, x\, (\alpha_{10}\, c) \qquad \gamma_1 := \lambda xcy.\, x\, y\, (\alpha_6\, c) \qquad \gamma_3 := \lambda xyc.\, x\, (\alpha_{11} c)\, y$$

Figure 5 The forcing translations $\tau \mapsto \tau^*$, $M \mapsto M^*$ and $t \mapsto t^*$.

(This set of axioms is not minimal, since α_2, α_6 and α_8 can be defined from the others.)

The above axioms basically express that the set C is upward-closed with respect to the pre-ordering $p \leq q$ (‘p is stronger than q’) defined by $p \leq q := \forall r^\kappa .\, C[pr] \Rightarrow C[qr]$. From this definition of the preorder $p \leq q$, we easily check that pq is the meet of p and q and that 1 is the greatest element. On the other hand, all the elements of κ outside C are equivalent with respect to the ordering $\leq$; they intuitively represent an ‘inconsistent condition’ stronger than all well-formed conditions.

In what follows, we will also need the following derived combinators:

$$
\begin{array}{ll}
\alpha_9 \ := \alpha_3 \circ \alpha_1 \circ \alpha_6 \circ \alpha_3 \ : \forall pqr.\, C[pqr] \Rightarrow C[pr] & \alpha_{10} := \alpha_2 \circ \alpha_5 : \forall pqr.\, C[pqr] \Rightarrow C[qr] \\
\alpha_{11} := \alpha_9 \circ \alpha_4 \qquad\qquad\quad : \forall pq.\, C[pq] \Rightarrow C[p(pq)] & \alpha_{12} := \alpha_5 \circ \alpha_3 : \forall pqr.\, C[p(qr)] \Rightarrow C[q(rp)] \\
\alpha_{13} := \alpha_3 \circ \alpha_{12} \qquad\qquad\quad : \forall pqr.\, C[p(qr)] \Rightarrow C[(rp)q] & \\
\alpha_{14} := \alpha_{12} \circ \alpha_{10} \circ \alpha_4 \circ \alpha_2 : \forall pqr.\, C[p(qr)] \Rightarrow C[q(rr)] & \alpha_{15} := \alpha_9 \circ \alpha_3 : \forall pqr.\, C[p(qr)] \Rightarrow C[qp]
\end{array}
$$

where $\alpha_i \circ \alpha_j \circ \cdots \circ \alpha_k$ stands for $\lambda c.\, \alpha_i\, (\alpha_j\, \ldots (\alpha_k\, c) \ldots)$ with c a fresh proof variable.

4.1.2 The three forcing translations

Given a forcing structure, the forcing transformation consists of three translations: $\tau \mapsto \tau^*$ on sorts, $M \mapsto M^*$ on higher-order terms (which is extended point-wise to equational theories) and $t \mapsto t^*$ on proof terms. The translations are given figure 5 (see [12] for the definition of all combinators).

▶ Remarks.

1. The translation on sorts simply replaces occurrences of o by $\kappa \to o$. This means that propositions will now depend on an extra parameter which is a forcing condition.
2. The translation on (higher-order) terms changes the sort of the term: N^τ is turned into $(N^*)^{\tau^*}$. The heart of this translation lies in the implication case and it merely propagates through the connectives in all the other cases.
3. The proof term translation instrumentalizes the computational interaction between abstractions and applications in proof terms:
   - it adds the γ_3 combinator in front of applications;
   - it shows the de Bruijn structure of bound variables: if an occurrence of the bound variable x has de Bruijn index n, it will be translated to $\beta_3^n\, (\beta_4\, x)$.

4.1.3 The forcing transformation on propositions

From the translation on terms, we define the usual forcing relation $p \Vdash A$ on propositions, letting:

$$p \Vdash A := \forall r^\kappa .\, C[pr] \Rightarrow A^* r \ .$$

This definition extends point-wise to contexts and we write it $p \Vdash \Gamma$. In addition to the expected properties of substitutivity and compatibility with the congruences $\simeq_{\mathcal{E}}$, this transformation on propositions enjoys the following important properties:

▶ **Proposition 4.2.**
1. *Forcing strongly commutes with universal quantification and equational implication:* $p \Vdash \forall x^{\tau}.\, A \;\simeq\; \forall x^{\tau^*}.\, (p \Vdash A) \qquad p \Vdash (M \doteq_{\tau} N \mapsto A) \;\simeq\; M^* \doteq_{\tau^*} N^* \mapsto (p \Vdash A)$
2. *Forcing is anti-monotonic:* $\forall pq.\, (p \Vdash A) \Rightarrow (pq \Vdash A)$
3. *Forcing an implication:* $p \Vdash A \Rightarrow B \iff \forall q^{\kappa}.\, (q \Vdash A) \Rightarrow (pq \Vdash B)$

▶ **Theorem 4.3** (Soundness). *If the judgment $\mathcal{E}; \Gamma \vdash t : A$ is derivable in PAω^+, then the judgment $\mathcal{E}^*; (p \Vdash \Gamma) \vdash t^* : p \Vdash A$ is derivable in PAω^+.*

This theorem is thus an effective way to turn a proof term $t : A$ (expressed in the forcing universe) into a proof term $t^* : p \Vdash A$ (expressed in the base universe).

4.2 Invariance under forcing

Clearly, the sorts that are invariant under the forcing translation are exactly the T-sorts defining Gödel's system T (see section 3.1.2). A proposition A whose free variables live in T-sorts is said to be *invariant under forcing* or *absolute* when there exist two closed proof terms ξ_A and ξ'_A such that

$$\xi_A : \forall p.\, (p \Vdash A) \Rightarrow (C[p] \Rightarrow A) \qquad\qquad \xi'_A : \forall p.\, (C[p] \Rightarrow A) \Rightarrow (p \Vdash A)\ .$$

An important class of absolute propositions is the class of *first-order propositions*, which contains the subclass of *arithmetical propositions* (in which all quantifications are relativized).

▶ **Definition 4.4** (First-order propositions). First-order propositions are defined by

$$A, B \quad ::= \quad \bot \quad | \quad M^{\tau} = N^{\tau} \quad | \quad A \Rightarrow B \quad | \quad \forall x^{\sigma}.\, A \quad | \quad M^{\iota} \in \mathbb{N}$$

where σ and τ are T-sorts (see section 3.1.2).

▶ **Theorem 4.5** (Invariance). *All first-order propositions are invariant under forcing.*

▶ **Theorem 4.6** (Elimination of a forced hypothesis). *If the propositions $1 \Vdash A$ and $A \Rightarrow B$ are derivable (in the empty context) and if B is absolute, then B is derivable too (in the empty context).*

Proof. Let u and s be proof terms such that $u : A \Rightarrow B$ and $s : 1 \Vdash A$. Using theorem 4.3, we have $u^* : 1 \Vdash A \Rightarrow B$. Because B is invariant under forcing, the previous theorem gives us $\xi_B : (1 \Vdash B) \Rightarrow C[1] \Rightarrow B$. We finally get $\xi_B\, (\gamma_3\, u^*\, s)\, \alpha_0 : B$. ◀

This theorem will be used to remove forcing in the proof of existence of a Herbrand tree.

4.3 The generic filter G

We now introduce PA$\omega^+ + G$, which extends PAω^+ with a constant G (the generic filter) and its axioms. To do so, we first assume that $\kappa \equiv \kappa^*$ (it is a T-sort) and that the set of well-formed conditions C (of sort $\kappa \to o$) is absolute, so that we have two proof terms ξ_C and ξ'_C such that

$$\xi_C \ :\ \forall pq.\, (p \Vdash C[q]) \Rightarrow (C[p] \Rightarrow C[q]) \qquad\qquad \xi'_C \ :\ \forall pq.\, (C[p] \Rightarrow C[q]) \Rightarrow (p \Vdash C[q])\ .$$

(At this stage, we do not need to know the particular implementation of C.)

The proof system PA$\omega^+ + G$ is defined from PAω^+ by adding a constant G of sort $\kappa \to o$ and five axioms expressing its properties. The first four axioms say that G is a filter in C:

$\boldsymbol{A_1}$ **:** G is a subset of C: $\forall p.\, p \in G \Rightarrow C[p]$,
$\boldsymbol{A_2}$ **:** G is non empty: $1 \in G$,
$\boldsymbol{A_3}$ **:** G is upward closed: $\forall pq.\, pq \in G \Rightarrow p \in G$,
$\boldsymbol{A_4}$ **:** G is closed under product: $\forall pq.\, p \in G \Rightarrow q \in G \Rightarrow pq \in G$,
The last axiom—*genericity*—relies on the following notion:

▶ **Definition 4.7** (Dense subset)**.** A set D of sort $\kappa \to o$ is said *dense* in C if for every element $p \in C$, there is an element $q \in C$ belonging to D and smaller than p. Formally, we let:

$$D \text{ dense } := \forall p^\kappa.\, C[p] \Rightarrow \exists q^\kappa.\, C[pq] \,\&\, pq \in D \quad (\Leftrightarrow \quad \forall p^\kappa.\, C[p] \Rightarrow \exists q^\kappa.\, C[q] \,\&\, q \in D \,\&\, q \leq p)$$

The last axiom on the set G is then:
$\boldsymbol{A_5}$ **:** G intersects every set $D^{\kappa \to o}$ (of the base universe) dense in C:
$(\forall p.\, C[p] \Rightarrow \exists q.\, C[pq] \,\&\, pq \in D) \;\Rightarrow\; \exists p.\, p \in G \,\&\, p \in D$.

Now we need to explain how the forcing translation extends to a translation from PAω^++G to PAω^+. The term translation on the generic filter G is defined by $G^* := \lambda pr.\, C[pr]$. This definition has the advantage of giving a very simple proposition for $p \Vdash q \in G$:

▶ **Fact 4.8.** $p \Vdash q \in G \;:=\; \forall r.\, C[pr] \Rightarrow (q \in G)^* r \;\simeq\; \forall r.\, C[pr] \Rightarrow C[qr] \;\simeq\; p \leq q$

We now need to prove the proposition $\forall p^\kappa.\, p \Vdash A_i$ (in PAω^+) for each of the five axioms A_1–A_5 of the generic filter G. Thanks to proposition 4.2 (anti-monotonicity), it is sufficient to prove that $1 \Vdash A_i$. Notice that the proof terms justifying the filter properties of G are small, except the proof term for genericity (the most complex property).

▶ **Proposition 4.9** (Forcing the properties of G)**.**

$$\gamma_1\,(\lambda x.\, \xi'_C\,(\alpha_1 \circ x \circ \alpha_3)) : 1 \Vdash \forall p.\, p \in G \Rightarrow C[p] \tag{4.9.i}$$

$$\lambda x.\, x : 1 \Vdash 1 \in G \tag{4.9.ii}$$

$$\gamma_1\,(\lambda x.\, \alpha_9 \circ x \circ \alpha_{10}) : 1 \Vdash \forall pq.\, pq \in G \Rightarrow p \in G \tag{4.9.iii}$$

$$\gamma_1(\lambda x.\, \gamma_1\,(\lambda y.\, \alpha_{13} \circ y \circ \alpha_{12} \circ x \circ \alpha_2 \circ \alpha_5 \circ \alpha_5)) : 1 \Vdash \forall pq.\, p \in G \Rightarrow q \in G \Rightarrow pq \in G \tag{4.9.iv}$$

$$\begin{aligned}\gamma_1\,(\lambda x.\, \gamma_1\,(\lambda y.\, \xi'_\perp\,(\lambda c.\, \xi_{\exists_2}\, \xi_C\, \xi_D(\gamma_3\, x\,(\xi'_c\,(\lambda\_.\, c)))\,(\alpha_2\,(\alpha_1\, c))\\ (\lambda c' d.\, \xi_\perp\,(\gamma_3\,(\gamma_3\,(\beta_3\,(\beta_4\, y))\; \mathrm{I})\,(\xi'_D\,(\lambda\_.\, d)))\, c'))))\\ :\; 1 \Vdash (\forall p.\, C[p] \Rightarrow \exists q.\, C[pq] \,\&\, pq \in D) \Rightarrow \exists p.\, p \in G \,\&\, p \in D\end{aligned} \tag{4.9.v}$$

where $\xi_{\exists_2}$ is the proof term (built using theorem 4.5) such that

$$\xi_{\exists_2}\, \xi_A\, \xi_B \;:\; (p \Vdash \exists n.\, A \,\&\, B) \Rightarrow (C[p] \Rightarrow \exists n.\, A \,\&\, B)$$

5 A proof of Herbrand's theorem by forcing

In order not to alter the meaning of the forcing poset through the forcing transformation, we choose to let $\kappa := \iota$ (the sort of individuals), because $\iota^* \equiv \iota$.

5.1 Interface for finite relations over Atom × Bool

We describe here an interface implementing finite relations over pairs of atoms and Booleans together with some operations (union, membership test) and properties. Everything can be implemented for instance by finite ordered lists of pairs (in the sort ι) without repetition. We assume given $\&\&^{\iota \to \iota \to \iota}$ and $||^{\iota \to \iota \to \iota}$, the (infix) Boolean conjunction and disjunction (at the term level) together with their defining equations ($e.g.$ $1 \mathbin{\&\&} b \simeq b$) that must hold at the congruence level (typically by β-reduction for suitable definitions of && and ||). Let us first describe the terms of the interface.

$\emptyset^\iota$	: the empty relation	$\mathrm{sing}^{\iota\to\iota\to\iota}$	: $\mathrm{sing}\, a\, b$ denotes $\{(a,b)\}$ and is written a^b
$\cup^{\iota\to\iota\to\iota}$	: union (infix symbol)	$\mathrm{test}^{\iota\to\iota\to\iota\to\iota}$	: $\mathrm{test}\, p\, a\, b$ tests if the atom a is mapped to b in p

The required properties over this structure are:

- associativity, commutativity and idempotence of $\cup$
- $\emptyset$ is a neutral element for $\cup$
- the specification equations of test: for all a, a', b, b', p, q with $a \neq a'$ or $b \neq b'$, $\mathrm{test}\, \emptyset\, a\, b = 0 \qquad \mathrm{test}\, a^b\, a\, b = 1 \qquad \mathrm{test}\, a^b\, a'\, b' = 0 \qquad \mathrm{test}\, (p \cup q)\, a\, b = \mathrm{test}\, p\, a\, b \,||\, \mathrm{test}\, q\, a\, b$

Using these terms and properties, we define two operations:

- testing membership: $\mathrm{mem}\, a\, p := \mathrm{test}\, p\, a\, 1 \,||\, \mathrm{test}\, p\, a\, 0$
- adding the binding (a, b) to p: $p \cup a^b$

Among finite relations, we can distinguish those that are *functional*, *i.e.* those representing finite functions from Atom to Booleans. We call them *finite valuations* and denote their set by FVal. Formally, this set (in the sense of section 3.3) is inductively defined using the following second-order encoding, which encompasses both finiteness and functionality:

$$\begin{aligned} p \in \mathrm{FVal} := \forall Z^{\iota\to o}.\, Z\, \emptyset \Rightarrow\ & (\forall r^\iota.\forall a \in \mathrm{Atom}.\ \mathrm{mem}\, a\, r \doteq_\iota 0 \mapsto Z\, r \Rightarrow Z\, (r \cup a^1)) \Rightarrow \\ & (\forall r^\iota.\forall a \in \mathrm{Atom}.\ \mathrm{mem}\, a\, r \doteq_\iota 0 \mapsto Z\, r \Rightarrow Z\, (r \cup a^0)) \Rightarrow Z\, p \end{aligned}$$

This shows the underlying computational structure of finite valuations: they are isomorphic to lists of atoms with two `cons` constructors (one for the atoms mapped to true, one for those mapped to false) without duplicates (thanks to the precondition $\mathrm{mem}\, a\, r \doteq_\iota 0 \mapsto \dots$ in the `cons` constructors).

Finally, we assume the existence of a function for testing membership, that is a proof term $\mathrm{Tot}_{\mathrm{test}}$ of the totality of test on finite valuations:

$$\mathrm{Tot}_{\mathrm{test}} \quad : \quad \forall p \in \mathrm{FVal}.\forall a \in \mathrm{Atom}.\forall b \in \mathrm{Bool}.\ \mathrm{test}\, p\, a\, b \in \mathrm{Bool}\ .$$

5.2 Programming in PAω^+

In order to ease writing proof terms in PAω^+, we introduce some macros:

$\langle a, b\rangle$	$\lambda f.\, f\, a\, b$	let $(x, y) = c$ in M	$c\, (\lambda xy.\, M)$
true, false	$\lambda xy.\, x$, $\lambda xy.\, y$	if b then f else g	$b\, f\, g$
$\mathrm{consT}\, a\, p$	$\lambda x_1x_2x_3.\, x_2\, a\, (p\, x_1\, x_2\, x_3)$	$\mathrm{consF}\, a\, p$	$\lambda x_1x_2x_3.\, x_3\, a\, (p\, x_1\, x_2\, x_3)$

They come with the inference rules (admissible in PAω^+) given Fig. 6.

$$\frac{\mathcal{E};\Gamma \vdash M : A \qquad \mathcal{E};\Gamma \vdash N : B}{\mathcal{E};\Gamma \vdash \langle M, N\rangle : A \wedge B} \qquad \frac{\mathcal{E};\Gamma \vdash M : A \wedge B \qquad \mathcal{E};\Gamma, x : A, y : B \vdash N : C}{\mathcal{E};\Gamma \vdash \text{let } (x, y) = M \text{ in } N : C}\ x, y \notin FV(M)$$

$$\frac{}{\mathcal{E};\Gamma \vdash \mathrm{true} : 1 \in \mathrm{Bool}} \qquad\qquad \frac{}{\mathcal{E};\Gamma \vdash \mathrm{false} : 0 \in \mathrm{Bool}}$$

$$\frac{\mathcal{E};\Gamma \vdash M : b \in \mathrm{Bool} \qquad \mathcal{E};\Gamma \vdash N : b \doteq 1 \mapsto A \qquad \mathcal{E};\Gamma \vdash P : b \doteq 0 \mapsto A}{\mathcal{E};\Gamma \vdash \text{if } M \text{ then } N \text{ else } P : A}$$

$$\frac{\mathcal{E};\Gamma \vdash M : a \in \mathrm{Atom} \qquad \mathcal{E};\Gamma \vdash N : p \in \mathrm{FVal}}{\mathcal{E};\Gamma \vdash \mathrm{consT}\, M N : p \cup a^1 \in \mathrm{FVal}}\ \mathrm{mem}\, a\, p \simeq_{\mathcal{E}} 0 \quad + \text{ idem for consF with } p \cup a^0 \in \mathrm{FVal}$$

Figure 6 Admissible inference rules in PAω^+.

5.3 Definition of our forcing structure

The interface and functions defined in the previous two sections allow us to build the forcing structure that we will use for Herbrand's theorem. In this setting, finite valuations will represent pieces of information about the current valuation that will be used to decide which closed instance of the proposition $F(\vec{x})$ is false. Note that most combinators are the identity thanks to the properties we imposed on the implementation of finite relations.

▶ **Definition 5.1** (Forcing structure for Herbrand's theorem)**.** Our forcing structure is given by

$$\begin{array}{lll} \kappa := \iota & C[p] := p \in \mathrm{FVal} \wedge (\mathrm{subH}\, p \Rightarrow \mathrm{subH}\, \emptyset) & p \cdot q := p \cup q \\ 1 := \emptyset & \alpha_3 = \alpha_4 = \alpha_5 = \alpha_6 = \alpha_7 = \alpha_8 := \mathrm{I} & \alpha_0 := \langle \lambda xyz.\, z, \mathrm{I} \rangle \end{array}$$

$$\alpha_1 = \alpha_2 := \lambda c.\, \mathrm{let}\ (p,t) = c\ \mathrm{in}\ \langle \mathrm{Up}_{\mathrm{FVal}}\ p, \lambda x.\, \mathrm{let}\ (x_1, x_2) = x\ \mathrm{in}\ t\ \langle x_1, \mathrm{Mon}_{\mathrm{subHtree}}\ x_2 \rangle \rangle$$

($\mathrm{Up}_{\mathrm{FVal}}$ and $\mathrm{Mon}_{\mathrm{subHtree}}$ will be defined in section 5.4.)

▶ Remarks.

1. We can simplify α_1 further if we replace $\mathrm{Mon}_{\mathrm{subHtree}}$ by I (which is a realizer of the same formula, see the remark after lemma 5.4). Note that in this case, α_1 is no longer a proof term but only a realizer (which is enough for our purpose) and we can write it $\alpha_1 := \lambda c.\, \mathrm{let}\ (c_1, c_2) = c\ \mathrm{in}\ \langle \mathrm{Up}_{\mathrm{FVal}}\ c_1, c_2 \rangle$.
2. Once we have proven the existence of a Herbrand tree (that is $\mathrm{subH}\, \emptyset$), the second part of the definition of the set C ($\mathrm{subH}\, p \Rightarrow \mathrm{subH}\, \emptyset$) is trivial. Therefore, the set C is logically equivalent to its first part FVal, the set of finite functions from Atom to Bool. It is interesting to notice that when Atom $= \mathbb{N}$, this is exactly the forcing conditions used to add a Cohen real [7]. This remark means that our forcing structure actually adds a single Cohen real (in the extended universe) which turns out to be the model we seek. It is a simple exercise of forcing to show that this real number is different from all real numbers of the base universe and that it is non computable.

In order to use all the results of section 4 and to be able to remove forcing using theorem 4.6, we need to prove that both subH and C are absolute.

▶ **Proposition 5.2.** *The sets* Tree, subH, FVal *and* C *are invariant under forcing.*

Proof. There exist proof terms in PAω^+ proving these properties. For instance, we have:

$$\xi_C := \xi_\wedge\, \xi_{\mathrm{FVal}}\, (\xi_\Rightarrow\, \xi'_{\mathrm{subH}}\, \xi_{\mathrm{subH}}) \qquad \xi'_C := \xi'_\wedge\, \xi'_{\mathrm{FVal}}\, (\xi'_\Rightarrow\, \xi_{\mathrm{subH}}\, \xi'_{\mathrm{subH}})$$ ◀

5.4 Formal statement of Herbrand's theorem in PAω^+

We now formalize in PAω^+ the statement of Herbrand's theorem presented in section 2 as:

> If $F(\vec{x})$ is a quantifier-free formula and all syntactic valuations validate $\exists \vec{x}.\, F(\vec{x})$,
> then $\exists \vec{x}.\, F(\vec{x})$ has a Herbrand tree.

Since we consider atomic formulas as elements of an abstract datatype (of sort ι) represented by the set Atom, a valuation is completely determined by its values on atoms and is thus defined as a function from atoms to propositions that we represent by a term of sort $\iota \to o$. We can extend a valuation ρ to quantifier-free formulas by the function $\mathrm{interp}^{(\iota \to o) \to \iota \to o}$ recursively defined by the following equations.

$$\mathrm{interp}\, \rho\, \perp\!\!\!\perp := \bot \qquad \mathrm{interp}\, \rho\, a := \rho\, a \qquad \mathrm{interp}\, \rho\, (c \Rrightarrow c') := (\mathrm{interp}\, \rho\, c) \Rightarrow (\mathrm{interp}\, \rho\, c')$$

The formula $F(\vec{x})$ is represented by a term $V^{\iota\to\iota}$ mapping any $\vec{v}$ to the corresponding quantifier-free formula $F(\vec{v})$ in Comp. The premise of Herbrand's theorem becomes the formula $\forall\rho^{\iota\to o}.\,\exists\vec{v}\in\mathrm{Term}.\,\mathrm{interp}\,\rho\,(V\,\vec{v})$.

We now need to define the proposition expressing that a binary tree is a Herbrand tree. Checking the correctness of a Herbrand tree is completely computational:

1. go down the tree and remember the partial valuation of your current branch,
2. evaluate $V\,\vec{v}$ at the leaves using the partial valuation accumulated so far.

This process is performed by the function subHtree recursively defined by these equations.

$$\begin{aligned}\mathrm{subHtree}\,p\,(\mathrm{Node}\,a\,t_1\,t_2) &:= \mathrm{subHtree}\;pa^1\,t_1\;\&\&\;\mathrm{subHtree}\;pa^0\,t_2\\ \mathrm{subHtree}\,p\,(\mathrm{Leaf}\,\vec{v}) &:= \mathrm{eval}\,p\,(V\,\vec{v})\,1\end{aligned}$$

The case of leaves is treated using a Boolean function $\mathrm{eval}^{\iota\to\iota\to\iota\to\iota}$ checking whether the truth value of $V\,\vec{v}$ (2nd arg.) is equal to b (3rd arg.) in the valuation p (1st arg.). The only non trivial case is the case of an atom where we need to look for the binding (a,b) into p, which can be done by the test function (see section 5.1). Since p is partial, $\mathrm{eval}\,p\,(V\,\vec{v})\,b=0$ can have two causes: either the truth value of $V\,\vec{v}$ in p is $1-b$ or p does not contain enough information to evaluate $V\,\vec{v}$. Conversely, when $\mathrm{eval}\,p\,(V\,\vec{v})\,b=1$, it means both that p contains enough information to evaluate $V\,\vec{v}$ and that the result is b. When $\mathrm{subHtree}\,p\,t=1$, we say that t *is a Herbrand tree below* p. Using subHtree, we finally define the predicate $\mathrm{subH}\,p$ expressing the existence of a Herbrand tree below the finite (and partial) valuation p: $\mathrm{subH}\,p := \exists t\in\mathrm{Tree}.\ \mathrm{subHtree}\,p\,t=1$.

Summing up, the formal statement of Herbrand's theorem in $\mathrm{PA}\omega^+$ is

$$(\forall\rho^{\iota\to o}.\,\exists\vec{v}\in\mathrm{Term}.\neg\,\mathrm{interp}\,\rho\,(V\,\vec{v}))\Rightarrow\mathrm{subH}\,\emptyset\ . \tag{H}$$

▶ **Lemma 5.3** (subH-merging). *Let p be a partial valuation and let a be an atom not appearing in p. If we have both* $\mathrm{subH}\;pa^1$ *and* $\mathrm{subH}\;pa^0$*, then we have* $\mathrm{subH}\,p$*.*

Proof. If t_1 and t_2 are Herbrand trees below pa^1 and pa^0 respectively, then $\mathrm{Node}\,a\,t_1\,t_2$ is a Herbrand tree below p. In $\mathrm{PA}\omega^+$, this lemma is formally stated and proved as follows.

$$\begin{aligned}\mathrm{merge}\ &:=\ \lambda x_a xy.\,\mathrm{let}\ (x_1,x_2)=x\ \mathrm{in}\ \mathrm{let}\ (y_1,y_2)=y\ \mathrm{in}\ \langle\mathrm{Node}\ x_a\,x_1\,y_1, y_2\circ x_2\rangle\\ &:\ \ \forall p^\iota.\forall a\in\mathrm{Atom}.\ \mathrm{mem}\,a\,p\doteq 0\mapsto\mathrm{subH}\,pa^1\Rightarrow\mathrm{subH}\,pa^0\Rightarrow\mathrm{subH}\,p\end{aligned}$$ ◀

▶ **Lemma 5.4** (Monotonicity). *The functions* test, eval *and* subHtree *are monotonic in* p*.*

Proof. There exists proof terms $\mathrm{Mon}_{\mathrm{test}}$, $\mathrm{Mon}_{\mathrm{eval}}$ and $\mathrm{Mon}_{\mathrm{subHtree}}$ of the propositions

$$\begin{aligned}&\forall pqab.\ \mathrm{test}\,p\,a\,b=1\Rightarrow\mathrm{test}\,(p\cup q)\,a\,b=1\\ &\forall pq.\forall c\in\mathrm{Comp}.\forall b\in\mathrm{Bool}.\,\mathrm{eval}\,p\,c\,b=1\Rightarrow\mathrm{eval}\,(p\cup q)\,c\,b=1\\ &\forall pq.\forall t\in\mathrm{Tree}.\ \mathrm{subHtree}\,p\,t=1\Rightarrow\ \mathrm{subHtree}(p\cup q)\,t=1\ .\end{aligned}$$

For instance, we have $\mathrm{Mon}_{\mathrm{test}} := \lambda xy.\,x\,y$. ◀

▶ Remark. In practice, there is no need to build formal proofs in $\mathrm{PA}\omega^+$ of monotonicity since their unrelativized version are realized by the identity (they are Horn formulas, true in the standard model): we can use them in proofs as axioms and later realize them by I.

▶ **Lemma 5.5** (FVal is upward-closed). *For all p and q, if* $(p\cup q)\in\mathrm{FVal}$*, then* $p\in\mathrm{FVal}$*.*

Proof. There exists a proof term $\mathrm{Up}_{\mathrm{FVal}}\ :\ \forall pq.\,(p\cup q)\in\mathrm{FVal}\Rightarrow p\in\mathrm{FVal}$. ◀

5.5 The full proof

5.5.1 The big picture

Now that we have our forcing setting, we can turn to the proof itself. It will be split between the base ($\mathbf{B}$) and forcing universes ($\mathbf{F}$) as shown by the following steps:

1. $\mathbf{B}$ Assume the premise $\forall \rho^{\iota \to o}.\, \exists \vec{v} \in \mathrm{Term}.\, \neg\, \mathrm{interp}\, \rho\, (V\, \vec{v})$.
2. $\mathbf{F}$ Lift the premise to the forcing universe.
3. $\mathbf{F}$ Make the proof: $t : \mathrm{subH}\, \emptyset$.
4. $\mathbf{B}$ Use the forcing translation: $t^* : 1 \Vdash \mathrm{subH}\, \emptyset$.
5. $\mathbf{B}$ Remove forcing: $\xi_{\mathrm{subH}}\, t^* \alpha_0 : \mathrm{subH}\, \emptyset$.
6. $\mathbf{B}$ Extract a witness.

▶ Remarks.

1. Steps 1 and 2 are automatic (a proof in the base universe is correct in the forcing one),
2. Step 5 has already been explained in the general case,
3. Step 6 uses standard classical realizability techniques and will not be discussed here.
4. Since the premise is not absolute (because of the quantification over valuations ρ of sort $\iota \to o$), we do not have a proof of Herbrand's theorem (in the base universe) and only get this admissible rule (see section 2.1): $\dfrac{\mathcal{E}; \Gamma \vdash u : \forall \rho^{\iota \to o}.\, \exists \vec{v} \in \mathrm{Term}.\ \mathrm{interp}\, \rho\, (V\, \vec{v})}{\mathcal{E}; \Gamma \vdash t(u) : \mathrm{subH}\, \emptyset}$.

5.5.2 The proof in the forcing universe (step 3)

Recall the formal statement of Herbrand's theorem (H) given in section 5.4. Since we are now in the forcing universe, we can use the properties of the generic filter G given in section 4.3. As usual with proof in forcing, we start by building the generic valuation $g = \bigcup G$, which is legal because G is a filter. We would like to let $g := \bigcup G$ and prove that it is total. However, its simpler to define $g := \lambda a.\, \exists p \in G.\ \mathrm{test}\, p\, a\, 1 = 1$ (total by definition) and then prove that it is equal to the union of G. To do so, instead of full genericity, we use a specialized axiom

$$\forall a \in \mathrm{Atom}.\, \exists p \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, p\, a\, b = 1\ . \tag{A}$$

First of all, we lift this axiom to quantifier-free formulas:

▶ **Lemma 5.6** (Evaluation by G). *There exists a proof term proving the proposition*

$$\forall c \in \mathrm{Comp}.\, \exists p \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{eval}\, p\, c\, b = 1\, \&\, \textit{if } b \textit{ then } \mathrm{interp}\, g\, c \textit{ else } \neg(\mathrm{interp}\, g\, c)\ .$$

Proof. The second part of the conjunct simply says that g must interpret a quantifier-free formula c exactly as any p in G would do, which is obvious by definition of g. We can therefore focus our attention on the first part on the conjunct, which is proved by induction on c, using property (4.9.iv) for the case of implication and axiom (A) for the case of atom. ◀

Because g is a valuation, we can feed it to the premise of (H) to get terms $\vec{v}$ such that $\vec{v} \in \mathrm{Term}$ (1) and $\neg\, \mathrm{interp}\, g\, (V\, \vec{v})$ (2). Using lemma 5.6 above with $V\, \vec{v}$, we get $p \in G$ and $b \in \mathrm{Bool}$ such that $\mathrm{eval}\, p\, (V\, \vec{v})\, b = 1$ (3) and if b then $\mathrm{interp}\, g\, (V\, \vec{v})$ else $\neg(\mathrm{interp}\, g\, (V\, \vec{v}))$ (4). Since $b \in \mathrm{Bool}$, we can make a case analysis:

1. $b = 1$: By (4), we have $\mathrm{interp}\, g\, (V\, \vec{v})$ which is in contradiction with (2).
2. $b = 0$: The equation (3) gives us $\mathrm{eval}\, p\, (V\, \vec{v})\, 0 = 1$ which, combined with (1), makes a proof of $\mathrm{subH}\, p$ (take $t := \mathrm{Leaf}\, \vec{v}$). But $p \in G$ and $G \subset C$ so that we have $C[p]$ and thus $\mathrm{subH}\, p \Rightarrow \mathrm{subH}\, \emptyset$ which allows us to conclude.

```
$\lambda c a f.$ let $(p, t) = \alpha_1\, c$ in                                  $a' := \xi_{\mathrm{Atom}}\, a\, (\alpha_1\, c) \;:\; a \in \mathrm{Atom}$
    if $\mathrm{Tot}_{\mathrm{test}}\; p\, a'$ true then $f\,(\alpha_1\, c)$ I true$^*$ I$^*$ else
    if $\mathrm{Tot}_{\mathrm{test}}\; p\, a'$ false then $f\,(\alpha_1\, c)$ I false$^*$ I$^*$ else
    $f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,(\mathrm{consT}\, a'\, p), \lambda t_1.\, f\,\langle \mathrm{Up}_{\mathrm{FVal}}\,(\mathrm{consF}\, a'\, p), \lambda t_2.\, t\,(\mathrm{merge}\, a'\, t_1\, t_2)\rangle\, \mathrm{I}\; \mathrm{false}^*\; \mathrm{I}^*\rangle\, \mathrm{I}\; \mathrm{true}^*\; \mathrm{I}^*$
```

Figure 7 The program realizing the axiom (A).

5.5.3 Back to the base universe (step 4)

Converting our proof term $t : \mathrm{subH}\,\emptyset$ in the forcing universe into a proof term $t^* : 1 \Vdash \mathrm{subH}\,\emptyset$ in the base universe follows exactly the methodology of section 4. The only subtlety is that instead of the genericity property of G (property (4.9.v)), we use the axiom (A) and we now need to translate it.

▶ **Proposition 5.7** (Forcing the axiom (A)). *There is a proof term in PAω^+ proving*

$$1 \Vdash \forall a \in \mathrm{Atom}.\, \neg(\forall p b.\, p \in G \Rightarrow b \in \mathrm{Bool} \Rightarrow \mathrm{test}\, p\, a\, b = 1 \Rightarrow \bot)\ .$$

Proof. The corresponding realizer is given in Fig. 7. Note that we use the simplified version of α_1. The (textual) proof is given in annex B. ◀

6 Computational interpretation

By analyzing the proof from the previous section, we obtain an algorithm for computing Herbrand trees. In order to study this algorithm, we use Krivine's classical realizability (see section 3.4), the setting in which the computational content of forcing has been studied [9, 11].

Overall, the interest of using forcing in this case is twofold. First, it allows to reason (in the forcing universe) on a single valuation, the *generic valuation*, instead of considering all of them. The forcing translation takes care of 'moving' this generic valuation across the tree to make sure we cover every possible branch. In short, forcing transparently manages the tree structure. Second, the forcing condition stores the tree under construction (see below), thus protecting it from any backtrack that might occur in the realizer of the premise of (H).

Computationally, a realizer of $C[p]$ is a dependent type of a zipper [6] at position p. Its first part ($p \in \mathrm{FVal}$) behaves as a finite list of atoms with two *cons* constructors, representing a finite approximation of the generic valuation g. Its second part ($\mathrm{subH}\, p \Rightarrow \mathrm{subH}\,\emptyset$) is the return continuation: provided we can find a Herbrand tree below p, we have a full Herbrand tree; it represents a *tree context* where the hole is at position p.

From this perspective, the key ingredient of the proof is axiom (A), which is responsible for the insertion of new nodes in the Herbrand tree and the scheduling of the computation of the subtrees. Indeed, it is the only place where the second component of the forcing condition (the tree context) is modified. It can be seen as the primitive called by the user program (the premise) to build the tree, like a system call giving access to g: given an atom a, this program (given Fig. 7) computes the truth value b of a in g, together with a witness of its answer: $p \in G$ containing a (remember that $g = \bigcup G$). To do so, it first checks whether a belongs to the current forcing condition q and if so, returns the associated value (lines 2 & 3) by feeding it to its continuation f. When a does not belong to q, we need to extend q. Since a can be mapped to either true or false in g, we consider both cases and hence make two calls to f (last line). These two calls can be understood intuitively as follows: first we lead f to believe we have a tree context for $p := qa^1$ (*i.e.* a fictitious realizer T' of $\mathrm{subH}\, qa^1 \Rightarrow \mathrm{subH}\,\emptyset$) although at the time, we only have one for q. When the computation inside f uses T', it must provide a Herbrand tree t_1 below qa^1. We then swap branches and call f again with $p := qa^0$ because this time, we do have a tree context for qa^0, namely $\lambda t_2.\, t\,(\mathrm{merge}\, a\, t_1\, t_2)$. Summing up, this last line contains both the extension of the tree (in $\mathrm{merge}\, a\, u\, v$) and the scheduling of the subtree computation (the two calls to f).

Furthermore, our realizer is completely intuitionistic (no callcc), which means that any backtrack during execution originates from the realizer of the premise of (H) and cannot affect the partial tree under construction which is stored in the second part of $C[p]$. Indeed callcc* takes care of saving and restoring the forcing condition. This restricted form of backtrack becomes a real instruction in the KFAM [12] (Krivine's Forcing Abstract Machine) which hard-wires the forcing translation of section 4 and features two *execution modes*:

- a *real mode* where terms have their usual KAM behavior,
- a *forcing mode* (or *protected mode*) where the first slot on the stack is considered as a forcing condition and terms behave as if they were translated through the forcing transformation.

In this machine, the premise of Herbrand's theorem would be executed only in forcing mode and could not affect the forcing condition (stored on the first slot of the stack).

Finally, the proof of section 5.5.2 in the forcing universe $\mathrm{PA}\omega^+ + G$ never uses the upward closure of G (property 4.9.iii). This means that we do not need to erase information from the partial Herbrand tree and suggests that our realizer is efficient.

References

1 Paul J. Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Science of the USA*, 50:1143–1148, 1963.

2 Paul J. Cohen. The independence of the continuum hypothesis II. *Proceedings of the National Academy of Science of the USA*, 51:105–110, 1964.

3 P.-L. Curien and Hugo Herbelin. The duality of computation. In *International Conference on Functional Programming*, pages 233–243, 2000.

4 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003.

5 Timothy G. Griffin. A formulae-as-types notion of control. In *Principles of Programming Languages (POPL'90)*, pages 47–58, 1990.

6 Gérard Huet. The zipper. *Journal of Functional Programming*, 7(5):549–554, 1997.

7 Thomas Jech. *Set theory.* Springer Monographs in Mathematics. Springer-Verlag, Berlin, 2003. The third millennium edition, revised and expanded.

8 J.-L. Krivine. Realizability in classical logic. In *Interactive models of computation and program behaviour*, volume 27 of *Panoramas et synthèses*, pages 197–229. SMF, 2009.

9 J.-L. Krivine. Realizability algebras: a program to well order ℝ. *Logical Methods in Computer Science*, 7, 2011.

10 Alexandre Miquel. Existential witness extraction in classical realizability and via a negative translation. In *Logical Methods in Computer Science*, 2010.

11 Alexandre Miquel. Forcing as a program transformation. *Logic in Computer Science*, pages 197–206, 2011.

12 Alexandre Miquel. Forcing as a program transformation. *Mathematical Structure in Computer Science*, 2013. to appear.

13 Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, 1997.

A Definition of the congruence relation

Reflexivity, symmetry, transitivity and base case

$$\frac{}{M \simeq_{\mathcal{E}} M} \qquad \frac{M \simeq_{\mathcal{E}} N}{N \simeq_{\mathcal{E}} M} \qquad \frac{M \simeq_{\mathcal{E}} N \qquad N \simeq_{\mathcal{E}} P}{M \simeq_{\mathcal{E}} P}$$

$$\frac{}{M \simeq_{\mathcal{E}} N}\,{\scriptstyle (M = N) \in \mathcal{E}}$$

Context closure

$$\frac{M \simeq_{\mathcal{E}} N}{\lambda x.\, M \simeq_{\mathcal{E}} \lambda x.\, N} \qquad \frac{A \simeq_{\mathcal{E}} B}{\forall x^\tau .\, A \simeq_{\mathcal{E}} \forall x^\tau .\, B} \qquad \frac{M \simeq_{\mathcal{E}} N \qquad P \simeq_{\mathcal{E}} Q}{M\,P \simeq_{\mathcal{E}} N\,Q}$$

$$\frac{A \simeq_{\mathcal{E}} B \qquad C \simeq_{\mathcal{E}} D}{A \Rightarrow C \simeq_{\mathcal{E}} B \Rightarrow D} \qquad \frac{M \simeq_{\mathcal{E}} N \qquad P \simeq_{\mathcal{E}} Q \qquad A \simeq_{\mathcal{E}, M=P} B}{M \doteq P \mapsto A \simeq_{\mathcal{E}} N \doteq Q \mapsto B}$$

$\beta\,\eta\,\iota$-conversion

$$\frac{}{(\lambda x^\tau .\, M)\, N^\tau \simeq_{\mathcal{E}} M[N^\tau / x^\tau]} \qquad \frac{}{\lambda x.\, M\, x \simeq_{\mathcal{E}} M}\,{\scriptstyle x \notin FV(M)}$$

$$\frac{}{\mathrm{rec}_\tau\, M\, N\, 0 \simeq_{\mathcal{E}} M} \qquad \frac{}{\mathrm{rec}_\tau\, M\, N (S\, P) \simeq_{\mathcal{E}} N\, P\, (\mathrm{rec}_\tau\, M\, N\, P)}$$

Semantically equivalent propositions

$$\frac{}{\forall x^\tau \forall y^\sigma .\, A \simeq_{\mathcal{E}} \forall y^\sigma \forall x^\tau .\, A} \qquad \frac{}{\forall x^\tau .\, A \simeq_{\mathcal{E}} A}\,{\scriptstyle x \notin FV(A)}$$

$$\frac{}{A \Rightarrow \forall x^\tau .\, B \simeq_{\mathcal{E}} \forall x^\tau .\, A \Rightarrow B}\,{\scriptstyle x \notin FV(A)}$$

$$\frac{}{M \doteq M \mapsto A \simeq_{\mathcal{E}} A} \qquad \frac{}{M \doteq N \mapsto A \simeq_{\mathcal{E}} N \doteq M \mapsto A}$$

$$\frac{}{M \doteq N \mapsto P \doteq Q \mapsto A \simeq_{\mathcal{E}} P \doteq Q \mapsto M \doteq N \mapsto A}$$

$$\frac{}{A \Rightarrow M \doteq N \mapsto B \simeq_{\mathcal{E}} M \doteq N \mapsto A \Rightarrow B}$$

$$\frac{}{\forall x^\tau .\, M \doteq N \mapsto A \simeq_{\mathcal{E}} M \doteq N \mapsto \forall x^\tau .\, A}\,{\scriptstyle x \notin FV(M, N)}$$

B Proof that the axiom (A) is forced

We want to prove (in PAω^+) that $1 \Vdash \forall a \in \mathrm{Atom}.\, \exists p \in G.\, \exists b \in \mathrm{Bool}.\ \mathrm{test}\, p\, a\, b = 1$. Unfolding the existential quantifiers, we need to prove

$$1 \Vdash \forall a \in \mathrm{Atom}.\, \neg(\forall p \in G.\, \forall b \in \mathrm{Bool}.\ \mathrm{test}\, p\, a\, b = 1 \Rightarrow \bot),$$

that is

$$1 \Vdash \forall a.\, a \in \mathrm{Atom} \Rightarrow \neg(\forall p \forall b.\, p \in G \Rightarrow b \in \mathrm{Bool} \Rightarrow \mathrm{test}\, p\, a\, b = 1 \Rightarrow \bot)\ .$$

Using proposition 4.2, it amounts to proving $(1 q_a) q_f \Vdash \bot$ given

$$x_a : \ q_a \Vdash a \in \mathrm{Atom}$$
$$x_f : \ q_f \Vdash \forall p \forall b.\, p \in G \Rightarrow b \in \mathrm{Bool} \Rightarrow \mathrm{test}\, p\, a\, b = 1 \Rightarrow \bot\ .$$

Since 1 is neutral for the product, this is the same as proving that $q_a q_f \Vdash \bot$. With repeated use of γ_3, we can turn x_f into $y := \lambda uv.\, \gamma_3(\gamma_3\,(\gamma_3\,(\beta_4\, y)\, u)\, v)$ which is a proof term for

$$\forall q \forall p \forall b.\, (q q_f \Vdash p \in G) \Rightarrow (q q_f \Vdash b \in \mathrm{Bool}) \Rightarrow (q q_f \Vdash \mathrm{test}\, p\, a\, b = 1) \Rightarrow (q q_f \Vdash \bot)\ .$$

Because test is total and we have $\mathrm{Tot}_{\mathrm{test}}\ :\ \forall p \in \mathrm{FVal}. \forall a \in \mathrm{Atom}. \forall b \in \mathrm{Bool}.\ \mathrm{test}\, p\, a\, b \in \mathrm{Bool}$ (both assumed in the interface for finite relations), we can proceed by case analysis:

- $\mathrm{test}\,(q_a q_f)\,a\,1 = 1$: We take $p := q_a q_f$ and $b := 1$. We use y with $q := q_a$ to prove $q_a q_f \Vdash \bot$ so that we have to prove its premises:
  - $\mathrm{I} : q_a q_f \le q_a q_f \equiv q_a q_f \Vdash q_a q_f \in G$,
  - $\gamma_1(\lambda u.\, \gamma_1(\lambda v.\, \beta_4\,(\beta_3\, u))) : q_a q_f \Vdash 1 \in \mathrm{Bool}$,
  - $\mathrm{I}^* : q_a q_f \Vdash \mathrm{test}\,(q_a q_f)\,a\,1 = 1$ because the equality holds in the equational theory (thanks to the case analysis).
- $\mathrm{test}\,(q_a q_f)\,a\,0 = 1$: It is similar to the previous case.
- $\mathrm{test}\,(q_a q_f)\,a\,1 = 0$ and $\mathrm{test}\,(q_a q_f)\,a\,0 = 0$: This case means that a does not appear in $q_a q_f$.

  We use $\xi'_\bot$ and we are left to prove $C[q_a q_f] \Rightarrow \bot$. We first prove $C[(q_a a^1) q_f] \Rightarrow \bot$ by using first $\xi_\bot$ then y with $q \equiv p := q_a a^1$ and $b := 1$.
  - $\alpha_9 : (q_a a^1) q_f \le q_a a^1 \equiv (q_a a^1) q_f \Vdash q_a a^1 \in G$ because product is the glb for $\le$,
  - $(q_a a^1) q_f \Vdash 1 \in \mathrm{Bool}$ proved as before,
  - $(q_a a^1) q_f \Vdash \mathrm{test}\,(q_a a^1)\,a\,1 = 1$ proved as before.

  In a similar fashion, we prove $C[(q_a a^0) q_f] \Rightarrow \bot$.

  Let us come back to the proof of $C[q_a q_f] \Rightarrow \bot$. Assume $C[q_a q_f]$. Applying the proof term for $C[(q_a a^1) q_f] \Rightarrow \bot$, we have to prove $C[(q_a a^1) q_f] \equiv (q_a a^1) q_f \in \mathrm{FVal} \wedge (\mathrm{subH}\,(q_a a^1) q_f \Rightarrow \mathrm{subH}\,\emptyset)$. The first part present no difficulty because we have $C[q_a q_f]$ and $\mathrm{mem}\,a\,(q_a q_f) = 0$: we just need to apply the second constructor of FVal. For the second part, we assume $\mathrm{subH}\,(q_a a^1) q_f$ and we want to prove $\mathrm{subH}\,\emptyset$. Instead we choose to prove $\bot \equiv \forall Z.\, Z$. Applying again the same method with $C[(q_a a^0) q] \Rightarrow \bot$, we end up proving $\mathrm{subH}\,\emptyset$ with $\mathrm{subH}\,(q_a a^1) q$ and $\mathrm{subH}\,(q_a a^0) q_f$ as extra hypotheses. For this, we just have to use merge (proposition 5.3) to get $\mathrm{subH}\, q_a q_f$ and apply it to $\mathrm{subH}\, q_a q_f \Rightarrow \mathrm{subH}\,\emptyset$ which we get from $C[q_a q_f]$.