Inhabitation in Typed Lambda-Calculi (A Syntactic Approach)*

Paweł Urzyczyn

Institute of Informatics
University of Warsaw
ul. Banacha 2, 02-097 Warszawa, Poland
urzy@mimuw.edu.pl

Abstract. A type is inhabited (non-empty) in a typed calculus iff there is a closed term of this type. The inhabitation (emptiness) problem is to determine if a given type is inhabited. This paper provides direct, purely syntactic proofs of the following results: the inhabitation problem is PSPACE-complete for simply typed lambda-calculus and undecidable for the polymorphic second-order and higher-order lambda calculi (systems **F** and $\mathbf{F}_\omega$).

1 Introduction

This paper is mostly of methodological and educational interest. Out of the three main theorems, two have been well-known for many years. The author does not know about a published proof of the third one, but it might have been known to some logicians already. However, the author believes that all the proofs below are much simpler than the existing ones. The motivation for this work was to provide a way to explain these results in a comprehensible way to a reader familiar with lambda-calculus but not necessarily with the proof theory and model theory of intuitionistic logic. The author hopes that the goal was achieved, at least in part.

Let us begin with explaining the problem. In a typed lambda-calculus (Curry or Church style) we normally assume a fixed set of types, and all these types may be assigned to term variables. Thus, there may be terms of arbitrary types, depending on the choice of free variables. However, it is normally the case that not all types may be assigned to closed terms. For instance, the type $\forall\alpha\alpha$ cannot be assigned to any closed term in the second-order lambda-calculus. Such an expression would be quite artifficial, as it would have to have all possible types at once. It is less obvious that, for instance, the type $((\alpha \to \beta) \to \alpha) \to \alpha$ is also *empty*, i.e. it cannot be assigned to any closed term. From a programmer's point of view, an empty type means a specification that should be avoided as it cannot be fulfilled by any program phrase. From a logician's point of view, under an appropriate Curry-Howard isomorphism, an *inhabited* (non-empty) type is just a

* This work was partly supported by NSF grant CCR-9417382, KBN Grant 8 T11C 034 10 and by ESPRIT BRA7232 "Gentzen".

provable formula. Thus, the above two examples must indeed be empty: the formula $\forall\alpha\alpha$ is a definition of falsity in the second-order propositional intuitionistic logic, and the other example is the famous Peirce's law — a purely implicational formula which is a classical, but not an intuitionistic tautology.

The inhabitation problem, which is the subject of this paper, can be stated as follows:

Given a type A, is there a closed term of type A?

But for any reasonable typed calculus, we can equivalently rephrase our problem so that it becomes more general:

Given a basis Γ and a type A, is there a term M such that $\Gamma \vdash M : A$?

Clearly, the first version is a special case of the second one. To see that there is also an effective reduction the other way, assume that Γ consists of the declarations $(x_i : A_i)$, for $i = 1, \ldots, n$. Then $\Gamma \vdash M : A$ holds iff there is a closed term of type $A_1 \to \cdots \to A_n \to A$.

The inhabitation problem was first addressed by logicians, as the provability problem for the corresponding logics. The simply-typed lambda-calculus corresponds to ordinary propositional intuitionistic logic, and it has been long well-known that the latter is decidable. This can be obtained from various finite model properties (w.r.t. Kripke models, Heyting algebras etc.) or with help of cut-elimination. (See for instance [5, 10, 6].) A precise characterization of the complexity of this logic was given by Statman [11], who first proved that it is PSPACE-complete. It follows that inhabitation for simply-typed lambda-calculus is also PSPACE-complete. We show in Section 2 how to directly prove the latter fact. As in the original paper of Statman, we use the QBF problem for PSPACE-hardness, but we reduce it directly to the implicational fragment of intuitionistic propositional logic, i.e., to ordinary simple types. Another source of simplification is that we work with lambda-calculus terms in normal form rather than with cut-free proofs.

The second-order lambda-calculus corresponds to the second-order propositional intuitionistic logic, see e.g. [9]. The best known proof that this logic is undecidable is due to Löb [8]. The undecidability result folows from a reduction of classical first-order logic into the second-order propositional intuitionistic logic. The proof is very complex and difficult to follow. In addition it is based on a semantic approach: it uses a completeness theorem for first-order intuitionistic logic extended with free second-order notions. The proof of Löb has been studied and slightly simplified by Arts [1] and Arts and Dekkers [2], but the resulting presentations are still quite complicated.

We reprove Löb's theorem using a coding suggested by Löb himself on page 716 of [8]. However, our proof is syntactic, by a reduction from a restricted fragment of intuitionistic first-order logic, which is simple but powerful enough to remain undecidable. Section 5 introduces this restricted first-order calculus, called system **P**, and Section 7 contains a purely syntactic reduction from provability in system **P** to inhabitation in system **F**. The only semantic element in

our proof is the soundness theorem for ordinary first-order classical logic used in Section 5. Replacing this with a syntactic argument about normal proofs is of course possible, but probably pointless.

Another known proof is due to Gabbay [4, 5]. But the system he proves undecidable is an extension of second-order propositional intuitionistic logic with an extra axiom scheme

$$\forall\alpha(A \vee B) \to (A \vee \forall\alpha B), \quad (\alpha \text{ not free in } A),$$

which is not intuitionistically valid. The proof is via completeness theorem for this logic w.r.t. a class of Kripke models.

The proof for system $\mathbf{F}$, which we give in Section 7 is considerably simpler than the original Löb's proof, but it is still a bit involved. To give the reader a feeling of the proof method without going into details, we first prove that inhabitation in system $\mathbf{F}_\omega$ is undecidable. This is done in Section 6. Due to the availability of free constructor variables, one can obtain the result in a simpler way, avoiding most of the details. In Section 7, we refine the approach of Section 6 to obtain a proof for system $\mathbf{F}$. In fact, for system $\mathbf{F}_\omega$, one does not have to use the same method. It is possible to have even a simpler proof, by a direct machine coding. We give such a proof in Section 4. Whether this approach could be improved to work for system $\mathbf{F}$ or not, should be a matter of further investigation.

Concluding, let us point out some related results. Inhabitation for intersection types is undecidable, see [13]. The dual problem to type inhabitation, the type reconstruction problem (given a term, can it be assigned a type?) is PTIME-complete for the simply-typed lambda-calculus, and undecidable for both systems $\mathbf{F}$ and $\mathbf{F}_\omega$, as well as for intersection types, see [3, 14, 12, 7].

In what follows we assume the reader to be familiar with typed lambda calculi, including systems $\mathbf{F}$ and $\mathbf{F}_\omega$. A reader who does not feel comfortable with $\mathbf{F}_\omega$ may skip Sections 4 and 6, and disregard any mention of "constructors" in Section 3. With the Curry-Howard isomorphism always in mind, we take the liberty to identify lambda-terms with proofs, and we use simplified notation of the form e.g. $\Gamma, A \vdash B$ instead of $\Gamma, x{:}A \vdash M : B$, whenever it is not essential which variable x and term M are considered.

2 Simply-typed lambda calculus

In this section we reprove a result of Statman [11], that the inhabitation problem for the finitely-typed lambda calculus is PSPACE-complete. As we are only interested in the implicational fragment of intuitionistic propositional logic, we can make this proof shorter and perhaps easier to follow than the original one. First we prove that our problem is in PSPACE:

Lemma 1. *There is an alternating polynomial time algorithm (and thus also a deterministic polynomial space algorithm) to determine whether a given type A is inhabited in a given basis Γ in the simply-typed lambda calculus.*

Proof. If a type is inhabited, it is inhabited by a term in an long normal form. To determine if there exists a term M in an long normal form, satisfying $\Gamma \vdash M : A$, we proceed as follows:

- If $A = A_1 \to A_2$, then M must be an abstraction, $M \equiv \lambda x{:}A_1.M'$. Thus, we look for an M' satisfying $\Gamma, x{:}A_1 \vdash M' : A_2$.
- If A is a type variable, then M is an application of a variable to a sequence of terms. We nondeterministically choose a variable z, declared in Γ to be of type $A_1 \to \cdots \to A_n \to A$. If there is no such variable, we reject. If $n = 0$ then we accept. If $n > 0$, we answer in parallel the questions if A_i are inhabited in Γ.

This alternating (or recursive) procedure is repeated as long as there are new questions of the form $\Gamma \vdash ? : A$. Note that if there are two variables in Γ, say x and y, declared to be of the same type B, then each term M can be replaced with $M[x/y]$ with no change of type. This means that a type A is inhabited in Γ iff it is inhabited in $\Gamma - \{y : B\}$, and we can only consider bases with all declared types being different. At each step of our procedure, the basis Γ either stays the same or it expands. Thus the number of steps (depth of recursion) does not exceed the squared number of subformulas of types in Γ, A, where $\Gamma \vdash ? : A$ is the initially posed question. ∎

To show PSPACE-hardness, we define a reduction from the satisfiability problem for classical second-order propositional formulas (QBF). Assume that a second-order propositional formula Φ is given. Without loss of generality we may assume that the negation symbol $\neg$ does not occur in Φ, but in the context $\neg p$, where p is a propositional variable.

Assume that all bound variables of Φ are different and that no variable occurs both free and bound. For each propositional variable p, occurring in Φ (free or bound), let α_p and $\alpha_{\neg p}$ be fresh type variables. Also, for each subformula φ of Φ, let α_φ be a fresh type variable. We construct a basis Γ_Φ from the following types:

- $(\alpha_p \to \alpha_\psi) \to (\alpha_{\neg p} \to \alpha_\psi) \to \alpha_\varphi$, for each subformula φ of the form $\forall p\psi$;
- $(\alpha_p \to \alpha_\psi) \to \alpha_\varphi$ and $(\alpha_{\neg p} \to \alpha_\psi) \to \alpha_\varphi$, for each φ of the form $\exists p\psi$;
- $\alpha_\psi \to \alpha_\vartheta \to \alpha_\varphi$, for each subformula φ of the form $\psi \wedge \vartheta$;
- $\alpha_\psi \to \alpha_\varphi$ and $\alpha_\vartheta \to \alpha_\varphi$, for each subformula φ of the form $\psi \vee \vartheta$.

If v is a zero-one valuation of propositional variables, then Γ_v is defined as Γ extended with the type variables

- α_p, when $v(p) = 1$;
- $\alpha_{\neg p}$, when $v(p) = 0$.

The following lemma is proven by a routine induction w.r.t. the length of formulas. Details are left to the reader.

Lemma 2. *For every subformula φ of Φ, and every valuation v, defined on the free variables of φ, the type α_φ is inhabited in Γ_v iff $v(\varphi) = 1$.*

From Lemma 2 we obtain PSPACE-hardness, since the reduction can be performed in logarithmic space. This, together with Lemma 1 implies the main result of this Section.

Theorem 3. *The inhabitation problem for simply-typed lambda-calculus is complete for polynomial space.*

3 Fully applied polymorphic terms

Unless stated otherwise, all lambda terms considered below are Church terms of system $\mathbf{F}_\omega$. The definitions and lemmas of the present section apply however as well to system $\mathbf{F}$, under the restriction that the only constructors are types. Each time we consider a typed term without specifying a basis, we assume that a basis is implicitely given. We use the notation M^A to stress that $M : A$ holds in an appropriately defined basis. An *atom* type is either a type variable or a type of the form $\alpha\vec{A}$, where α is a constructor variable and $\vec{A}$ are constructors of appropriate kinds.

It is convenient to generalize the notion of an long normal form to polymorphic terms. A term M in normal form is said to be *fully applied* iff each subterm of M of a functional or universal type is either an abstraction or occurs in an applied position. More formally, we define fully applied terms of system $\mathbf{F}_\omega$ by induction as follows:

1. If N^B is fully applied then so is $(\lambda x{:}A.N)^{A\to B}$;
2. If N^B is fully applied then so is $(\Lambda\alpha.N)^{\forall\alpha B}$;
3. If x^A is a term variable, B is an atom type, and for some sequence $\vec{\Delta}$ of types (constructors) and fully applied terms we have $x\vec{\Delta} : B$, then $x\vec{\Delta}$ is fully applied.

In clause (2) above, the variable α may be a constructor variable of an arbitrary kind, and in clause (3) the sequence $\vec{\Delta}$ may also involve arbitrary constructors.

It should be obvious from the definition that fully applied terms have the following properties:

- A fully applied term of a functional type is an abstraction;
- A fully applied term of a universal type is a type (constructor) abstraction;
- A fully applied term of an atom type is an application or a variable.

The following lemma holds for both systems, $\mathbf{F}$ and $\mathbf{F}_\omega$.

Lemma 4. *If $\Gamma \vdash M : A$ then there exists a fully applied term M' such that $\Gamma \vdash M' : A$.*

A *target* of a type A is the type (constructor) variable that occurs in A along the rightmost path. A basis Γ is *ordinary* if targets of all types in the range of Γ are free variables. The following easy lemma holds for both systems: $\mathbf{F}$ and $\mathbf{F}_\omega$.

Lemma 5. *Let Γ be an ordinary basis and let $\Gamma \vdash M : \alpha$ or $\Gamma \vdash M : \alpha\vec{A}$, where α is a type (constructor) variable and M is a fully applied normal form. Then $M \equiv x\vec{\Delta}$, where α is the target of $\Gamma(x)$.*

4 Inhabitation in $\mathbf{F}_\omega$ — first proof

In this section we show that the inhabitation problem for $\mathbf{F}_\omega$ is undecidable. We use the simplest method: a direct coding of a Turing Machine. We assume that a deterministic TM must have the following properties:

- it never attempts to move left, when reading the leftmost tape cell;
- it never writes a blank;
- there is only one final state.

Tape symbols and internal states are coded with help of (different) type variables and are identified with these variables. In addition we fix a type variable $\#$, which will denote an empty sequence, and one constructor variable κ : **Prop** $\Rightarrow$ **Prop** $\Rightarrow$ **Prop**. All the above variables are free everywhere in the construction. All bound variables are type variables.

A finite sequence $w = a \cdot w'$ is coded as the type $[w] = \kappa a[w']$, where $[w']$ codes w' (the variable $\#$ codes the empty sequence). The notation $\langle A, B, C\rangle$ abbreviates $\kappa A(\kappa BC)$. A Turing Machine ID of the form (w, q, v) is understood so that wv is the tape contents, q is the current state, and the first symbol of v is being scanned. The word v extends to the right up to (and including) the first blank. Such an ID is coded as $\langle [w^R], q, [v]\rangle$ (where w^R stands for the reverse of w), i.e. as $\kappa[w^R](\kappa q[v])$. That is, the initial ID (with empty input word) is coded by $D_0 = \kappa\#(\kappa q_0(\kappa(\text{blank})\#))$.

The transition relation of a TM is coded as a set of types, each representing one possible clause of the form $(q, a) \vdash (b, p, \varepsilon)$, where ε is either -1 or 0 or $+1$, and denotes "left", "stay" or "right" respectively. The coding depends on v as follows: A "stay" clause $(q, a) \vdash (b, p, 0)$ is coded by the following type:

$$\forall\alpha\beta(\langle\alpha, p, \kappa b\beta\rangle \to \langle\alpha, q, \kappa a\beta\rangle).$$

A "left" clause $(q, a) \vdash (b, p, -1)$ is coded by a set of types, one for each tape symbol c, of the following form:

$$\forall\alpha\beta(\langle\alpha, p, \kappa c(\kappa b\beta)\rangle \to \langle\kappa c\alpha, q, \kappa a\beta\rangle).$$

A "right" clause $(q, a) \vdash (b, p, +1)$, where $a \neq$ blank, is coded by a single type:

$$\forall\alpha\beta(\langle\kappa b\alpha, p, \beta\rangle \to \langle\alpha, q, \kappa a\beta\rangle).$$

Finally, a "right" clause $(q, \text{blank}) \vdash (b, p, +1)$, is represented by

$$\forall\alpha\beta(\langle\kappa b\alpha, p, \kappa(\text{blank})\#\rangle \to \langle\alpha, q, \kappa(\text{blank})\#\rangle).$$

Assuming f is the final state, the final ID's are represented by the type $A_0 = \forall\alpha\beta\langle\alpha, f, \beta\rangle$.

Let A_i, for $i = 1, \ldots, n$ be all types used to code the transition function. Let Γ be the context consisting of type assumptions $x_i : A_i$, for $i = 0, \ldots, n$, thus including the type A_0. The next lemma states the correctness of our representation (recall that D_0 is the type of the initial ID).

Lemma 6. *The machine converges on empty input word iff there exists M such that $\Gamma \vdash M : D_0$.*

Proof. For the "only if" part, let $\mathcal{D}_0, \ldots, \mathcal{D}_p$ be the sequence of ID's that forms a converging computation on empty input ($\mathcal{D}_p$ is a final ID), and let $D_0, \ldots, D_p$ be types representing these ID's. By a "backward" induction we prove that for all i there are M_i with $\Gamma \vdash M_i : D_i$. The "if" part is based on the following claim:

Suppose that $\mathcal{D}$ is an ID coded by a type D and that $\mathcal{D}'$, coded by D', is obtained from $\mathcal{D}$ in one machine step. Let $\Gamma \vdash M : D$ for some fully applied M. Then either $\mathcal{D}$ is final or $M = x_i BCM'$, for some B, C and some M' such that

$$\Gamma \vdash M' : D'.$$

Note that M' must be shorter than M, thus the above may only be applied a finite number of times. Details are left to the reader. ∎

The above lemma gives an effective reduction of the halting problem for TM's to inhabitation for $\mathbf{F}_\omega$. Therefore we obtain:

Theorem 7. *The inhabitation problem for system $\mathbf{F}_\omega$ is undecidable.*

Let us stress here that this construction works only because we have a free constructor variable κ. A type that begins with κ must necessarily be an application and thus κ may be used to build sequences of rigid "cells" able to store data, the contents of such cells never being accessed from outside and decomposed. This approach does not seem to be easily applicable to to system $\mathbf{F}$. For instance, a naive way of coding $a_1 a_2 \ldots a_n$ into something like $a_1 \to a_2 \to \cdots a_n \to \#$ does not work, because e.g. codes of ab, ba and abb are equivalent. The author does not know how to replace κ by a fixed type constructor definable in $\mathbf{F}$, so that the proof above remains correct. In fact, it seems that such a fixed constructor may not exist at all, for the following reason. An application κAB may neither bind type variables free in A or B, nor it can introduce new variable occurrences into A or B. Thus, even if many applications of κ can possibly introduce deep nesting of quantifier bindings, the scopes of quantifiers cannot "interleave", and may be considered disjoint. On the other hand, it is necessary in Section 5 to use interleaving quantifier scopes.

5 Provability in first-order logic

It is well-known that both classical and intuitionistic first-order logic are undecidable. However, the undecidabilty result holds also for quite restricted first-order systems. In this section we consider one such system, which we call system $\mathbf{P}$, and which is a fragment of intuitionistic first-order calculus. The language of system $\mathbf{P}$ consists of the following symbols:

- binary predicate letters (relation symbols);
- individual variables;

- the symbol **false**;
- the implication sign "$\rightarrow$" and the universal quantifier "$\forall$".

The restriction to binary relations is not essential, and is introduced to uniformize further consideration. The symbol **false** is introduced here for presentation reasons. In fact, we do not formally associate any particular logical meaning to this symbol, so it may be considered just to be a nullary relation symbol (this in turn simulated by a binary one). Later in Sections 6 and 7, we indentify individual variables with type variables, so we now adopt the convention that individual variables of system **P** are written with Greek letters.

The set of formulas of system **P** is a proper subset of first-order formulas over the above alphabet, and is divided into the following three categories:

Atomic formulas: the constant **false**, and expressions $\mathcal{P}(\alpha, \beta)$, where $\mathcal{P}$ is a predicate letter and α, β are individual variables;

Universal formulas: expressions of the form $\forall\vec{\alpha}(A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n)$, where all A_i's are atomic formulas, $A_i \neq$ **false** for $i \neq n$, and each free individual variable of A_n must also occur free in some A_i with $i < n$;

Existential formulas: formulas of the form $\forall\vec{\alpha}(A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_{n-1} \rightarrow \forall\beta(A_n \rightarrow \textbf{false}) \rightarrow \textbf{false})$, where all A_i's are atomic formulas, and all $A_i \neq$ **false**.

In addition we require that formulas of **P** contain no dummy quantifiers i.e., each quantified variable must actually occur in its scope. (This assumption, as well as the restriction on variable occurrences in universal formulas, is not essential, but simplifies the proofs.)

The name "existential formula" is of course used only for simplicity as it should rather be "universally-existential". Indeed, a classical equivalent of an "existential" formula as above is $\forall\vec{\alpha}(A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_{n-1} \rightarrow \exists\beta A_n)$, assuming the ordinary meaning of **false**. Intutionistically, these two formulas become equivalent if **false** is replaced by a universally quantified propositional variable.

The proof rules of system **P** are ordinary natural deduction rules:

(Axiom) $\quad \Gamma, A \vdash A;$

(E$\rightarrow$) $\quad \dfrac{\Gamma \vdash A \rightarrow B,\ \Gamma \vdash A}{\Gamma \vdash B}$

(I$\rightarrow$) $\quad \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B}$

(I$\forall$) $\quad \dfrac{\Gamma \vdash B}{\Gamma \vdash \forall\alpha\, B}$ $\quad$ (where α is not free in Γ)

(E$\forall$) $\quad \dfrac{\Gamma \vdash \forall\alpha B}{\Gamma \vdash B\{\beta/\alpha\}}$

In the quantifier elimination rule above, β is an individual variable (there are no composite terms). Note that there is no rule for **false**, which means that **false** behaves like a propositional variable rather than a logical constant. However, it does not make difference for us, as long as we only ask about consistency (non-provability of **false**) of sets of **P**-formulas. Indeed, suppose that a set of assumptions Γ is consistent in **P**. By induction w.r.t. the length of normal proofs, one can show that all atomic formulas provable in Γ with help of **false**-elimination are also provable in **P**. (Consider the shortest proof with a **P**-consistent set of assumptions which uses **false**-elimination to prove something that cannot be proven in **P**.) It follows that a **P**-consistent set of assumptions remains consistent after allowing **false**-elimination.

Lemma 8. *For every deterministic two-counter automaton* **M**, *one can effectively construct a finite set* $\Gamma_{\mathbf{M}}$ *of* **P***-formulas, such that*

$$\mathbf{M} \textit{ terminates on input } (0,0) \quad \textit{iff} \quad \Gamma_{\mathbf{M}} \vdash \mathbf{false} \textit{ holds in system } \mathbf{P}.$$

Proof. We leave the proof to the full version of the paper. Here we would like to give only an overview of the idea.

An ID of our automaton of the form $C = \langle Q, m, n\rangle$ (the current state is Q, the value of the first counter is m, and the value of the second counter is n) is represented by the following set Γ_C of assumptions:

- $Q(\alpha)$;
- $P_1(\alpha, \alpha_0)$ and $P_1(\alpha_{i-1}, \alpha_i)$, for $i = 1, \ldots, m$;
- $P_2(\alpha, \beta_0)$ and $P_1(\beta_{i-1}, \beta_i)$, for $i = 1, \ldots, n$;
- $D(\alpha)$, $D(\alpha_i)$ and $D(\beta_j)$, for $i < m$ and $j < n$;
- $E(\alpha_m)$ and $E(\beta_n)$.

In the above, Q is a predicate letter representing the state Q, and the α's and β's form two "chains" representing the values of the counters. The role of D and E is to distinguish the end of a chain from an "internal element".

To guarantee the correctness of this representation, we need the following set of formulas Γ_1:

- $\forall\alpha\beta(S(\alpha,\beta) \to D(\beta))$;
- $\forall\alpha(D(\alpha) \to \forall\beta(P_1(\alpha,\beta) \to \mathbf{false}) \to \mathbf{false})$;
- $\forall\alpha(D(\alpha) \to \forall\beta(P_2(\alpha,\beta) \to \mathbf{false}) \to \mathbf{false})$;
- $\forall\alpha(\forall\beta(S(\alpha,\beta) \to \mathbf{false}) \to \mathbf{false})$.

If we think of these formulas as of existential ones, their meaning is to guarantee that a chain representing a counter can be extended as long as the end is not reached, and that there is always a next moment. The transition function of our automaton is represented by the set of formulas Γ_2, which contains 3 or 4 formulas for each internal state of our automaton.

For instance, assume that the machine instruction for state Q be to increase the first counter by 1 and go to state Q'. Then Γ_2 contains the formulas:

- $\forall\alpha\beta(Q(\alpha) \to S(\alpha,\beta) \to Q'(\beta))$, to represent the change of state;
- $\forall\alpha\beta\gamma\delta(Q(\alpha) \to S(\alpha,\beta) \to P_1(\alpha,\gamma) \to P_1(\beta,\delta) \to P_1(\delta,\gamma))$, to lenghten up the chain;
- $\forall\alpha\beta\gamma\delta(Q(\alpha) \to S(\alpha,\beta) \to P_1(\beta,\delta) \to D(\delta))$; to prevent an "artifficial" zero;
- $\forall\alpha\beta\gamma(Q(\alpha) \to S(\alpha,\beta) \to P_2(\alpha,\gamma) \to P_2(\beta,\gamma))$, to leave the other counter unmodified.

In addition, we put into Γ_2 the formula

$$\forall\alpha(Q_f(\alpha) \to \mathbf{false}),$$

where Q_f is the final state. Let $\Gamma = \Gamma_1 \cup \Gamma_2$. We define Γ_M to be $\Gamma \cup \Gamma_{C_0}$, where C_0 is the initial ID. The "only if" part of our hypothesis follows from the following claim that is proven by induction on the length of computations:

*Let C be an arbitrary ID. If a final ID is reachable from C then $\Gamma_C \cup \Gamma \vdash$ **false**.*

To prove the "if" part, we assume that the machine does not terminate, and we construct a first-order classical model for the set $\Gamma_{\mathbf{M}}$ in which the nullary predicate letter **false** is interpreted as falsity (and thus deserves its name). It follows that **false** cannot be proved from $\Gamma_{\mathbf{M}}$, even classically. ∎

Corollary 9. *Provability in system* **P** *is undecidable.*

6 Inhabitation in $\mathbf{F}_\omega$ — second proof

This section is not essential for the technical contents of rest of the paper. Its aim is to help the reader understand Section 7, where we apply the same approach, but with more technical difficulties. It is much easier to do the same with help of higher-order constructors.

Below we define a fragment of system $\mathbf{F}_\omega$, denoted $\mathbf{F}_\omega^{\mathbf{P}}$, which represents the system **P** of Section 5. The system $\mathbf{F}_\omega^{\mathbf{P}}$ is obtained from $\mathbf{F}_\omega$ by imposing appropriate syntactic restrictions on types. We begin with type and constructor variables:

- All constructor variables are either type variables or are of kind $\mathbf{Prop} \Rightarrow \mathbf{Prop} \Rightarrow \mathbf{Prop}$. There is one distinguished type variable **false**.
- Type variables, except **false**, are identified with the individual variables of **P**, while the first-order constructor variables are identified with the predicate letters of **P**.

In order to simulate **P** within $\mathbf{F}_\omega^{\mathbf{P}}$, we require that constructors may be applied only to individual variables, and only individual variables, except **false**, may be quantified. This restricts the use of constructor variables in types so that they occur as binary relation symbols applied to individual variables. Under this restriction, we may think of types as exactly corresponding to first-order formulas.

In order to obtain an exact correspondence between $\mathbf{F}_\omega^\mathbf{P}$ and $\mathbf{P}$, we define the set of types of $\mathbf{F}_\omega^\mathbf{P}$ (called also $\mathbf{F}_\omega^\mathbf{P}$-formulas) to consists exclusively of types representing formulas of $\mathbf{P}$. We identify these types with the underlying formulas. That is, we have again three kinds of expressions:

Atomic formulas: the variable **false**, and constructor applications of the form $\mathcal{P}\alpha\beta$, where $\mathcal{P}$ is a predicate letter and α, β are individual variables;

Universal formulas: types of the form $\forall\vec{\alpha}(A_1 \to A_2 \to \cdots \to A_n)$, where all A_i's are atomic formulas, $A_i \neq \mathbf{false}$ for $i \neq n$, and each free individual variable of A_n must also occur free in some A_i with $i < n$;

Existential formulas: types of the form $\forall\vec{\alpha}(A_1 \to A_2 \to \cdots \to A_{n-1} \to \forall\beta(A_n \to \mathbf{false}) \to \mathbf{false})$, where all A_i's are atomic formulas, and all $A_i \neq \mathbf{false}$.

Recall that we also assume that formulas contain no dummy quantifiers.

An $\mathbf{F}_\omega$-basis is called an $\mathbf{F}_\omega^\mathbf{P}$-*basis* iff it consists solely of $\mathbf{F}_\omega^\mathbf{P}$-types. We say that the $\mathbf{F}_\omega$-judgement $\Gamma \vdash M : A$ holds in $\mathbf{F}_\omega^\mathbf{P}$ iff all types occurring in it (including all types in M) are $\mathbf{F}_\omega^\mathbf{P}$-formulas. It should be obvious that

$$\Gamma \vdash M : A \text{ holds in } \mathbf{F}_\omega^\mathbf{P}, \text{ for some } M \quad \text{iff} \quad \Gamma \vdash A \text{ in } \mathbf{P}.$$

However, a valid $\mathbf{F}_\omega$-judgement $\Gamma \vdash M : A$, with an $\mathbf{F}_\omega^\mathbf{P}$-basis Γ and an atomic $\mathbf{F}_\omega^\mathbf{P}$-formula A, does not have to hold in $\mathbf{F}_\omega^\mathbf{P}$. This is because the term M may contain occurrences of types of the form e.g., $\mathcal{P}(B, C)$, where the predicate letter $\mathcal{P}$ is applied to some complex types B, C, and not to type variables. But such a term can always be replaced by another one, which makes a valid $\mathbf{F}_\omega^\mathbf{P}$ judgement.

Lemma 10. *Let Γ be an $\mathbf{F}_\omega^\mathbf{P}$-basis, and let an atomic formula A be inhabited in Γ in the system $\mathbf{F}_\omega$. Then A is inhabited in Γ in the system $\mathbf{F}_\omega^\mathbf{P}$, and thus it is provable in $\mathbf{P}$ from Γ.*

Proof. We first prove the following claim. Let $\mathcal{P}$ be of kind **Prop** $\Rightarrow$ **Prop** $\Rightarrow$ **Prop**. If Γ is an $\mathbf{F}_\omega^\mathbf{P}$-basis and $\Gamma \vdash M : \mathcal{P}AB$ holds in $\mathbf{F}_\omega$, for some types A and B, then A and B are type variables. To prove the claim, suppose the contrary, and assume that M is the shortest possible fully applied counterexample. By Lemma 5, we must have a variable z declared in Γ with target $\mathcal{P}$, such that $M \equiv z\vec{\Delta}$. If the type of z is an atomic formula, then A and B must be variables (and $\vec{\Delta}$ is empty). Otherwise, the type of z is an universal formula, say $\forall\vec{\alpha}(A_1 \to A_2 \to \cdots \to A_n \to \mathcal{P}\alpha_1\alpha_2)$. Then $M \equiv z\vec{C}\vec{N}$, for some types $\vec{C}$ and some terms $\vec{N}$. Types A and B occur in $\vec{C}$ at places corresponding to occurrences of the variables α_1 and α_2 in $\vec{\alpha}$. But α_1 and α_2 must occur in some of the A_i's. Thus, one of the terms in N is actually of type $\mathcal{Q}CD$, where $\mathcal{Q}$ is a predicate letter and either C or D is a non-variable type (equal to either A or B). This gives a counterexample to our claim, shorter than M.

Now suppose that an atomic formula A is inhabited in Γ in the system $\mathbf{F}_\omega$, say $\Gamma \vdash M : A$ with fully applied M. We proceed by induction on M. If A has the form $\mathcal{P}\alpha\beta$ then either M is a variable (and the hypothesis is obvious) or

$M \equiv z\vec{C}\vec{N}$, for some types $\vec{C}$ and some terms $\vec{N}$. It follows from the previous claim that the types $\vec{C}$ must be just type variables, as otherwise one of the terms in $\vec{N}$ would contradict the claim. Thus, the induction hypothesis applies for $\vec{N}$, i.e., types of these terms are inhabited in $\mathbf{F}^{\mathrm{P}}_{\omega}$. Let $\vec{N}'$ be a vector of inhabitants. Thus A is also inhabited, since the application $z\vec{C}\vec{N}'$ does not introduce unwanted types.

It remains to consider the case when $A \equiv \mathbf{false}$. One possibility is that $M \equiv z\vec{C}\vec{N}$, where the type of z is an universal formula (with target **false**). Then we proceed as in the previous case. Otherwise, M must be of the form $z\vec{C}\vec{N}L$, where $\vec{C}$ must be variables and $\vec{N}$ may be replaced by $\vec{N}'$, just as in the previous case. The term L is of type $\forall\beta(B \to \mathbf{false})$, for some atomic B, and has the form $\Lambda\beta.\lambda y{:}B.L_1^{\mathbf{false}}$. Thus, $\Gamma, y : B \vdash L_1 : \mathbf{false}$ and $\Gamma, y : B$ is an $\mathbf{F}^{\mathrm{P}}_{\omega}$ basis. This allows us to apply the induction hypothesis to L_1. ∎

From the above Lemma, and from Lemma 8, we conclude that a given two-counter automaton $\mathbf{M}$ terminates on input $(0,0)$ iff $\Gamma_{\mathbf{M}} \vdash \mathbf{false}$ is a valid $\mathbf{F}_{\omega}$-judgement. Thus, the second proof of Theorem 7 is completed.

The concluding observation is that although it is obvious that the above property holds for $\mathbf{F}^{\mathrm{P}}_{\omega}$, it is not obvious at all for the full system $\mathbf{F}_{\omega}$. Luckily, the proof is still easy, due to the fact that relation symbols are represented by free constructors. In the next Section, we reprove this once again, but this time for system $\mathbf{F}$. The main technical problem that must be solved for this proof is how to replace the free constructor variables with type expressions, so that an analogue of Lemma 10 is true.

7 Inhabitation in F

The departure point for this section is a given set $\Gamma_{\mathbf{M}}$ of closed formulas of $\mathbf{P}$, as in Section 5 (equivalently seen as a set of $\mathbf{F}^{\mathrm{P}}_{\omega}$ formulas, with no free individual variables). The goal is to effectively define a coding $\overline{\Gamma}_{\mathbf{M}}$ of the formulas (types) in $\Gamma_{\mathbf{M}}$, so that $\Gamma_{\mathbf{M}} \vdash \mathbf{false}$ holds in $\mathbf{P}$ if and only if $\overline{\Gamma}_{\mathbf{M}} \vdash \mathbf{false}$ holds in system $\mathbf{F}$.

It is technically useful to assume that we have three disjoint sorts of type variables in system $\mathbf{F}$: the individual variables, the auxiliary variables used for the coding, and one distinguished type variable **false**. Of course, the variable **false** is used to code the symbol **false**. The coding of predicate letters uses a number of fresh auxiliary type variables, which remain free in all types. For each predicate letter $\mathcal{P}$ we need three such type variables denoted p, p_1 and p_2 (the convention is to use p for $\mathcal{P}$, and q for $\mathcal{Q}$ etc.) and we will use three more type variables ξ, η_1 and η_2. We code an atomic formula $\mathcal{P}(\alpha,\beta)$ as the type

$$\mathcal{P}(\alpha,\beta) \equiv [(\alpha \to p_1) \to (\beta \to p_2) \to p] \to \xi.$$

To avoid rewriting this long type each time, we will abbreviate the argument part to $\mathcal{P}_{\alpha\beta}$, so that we write

$$\mathcal{P}(\alpha,\beta) \equiv \mathcal{P}_{\alpha\beta} \to \xi.$$

We use the abbreviations $\mathcal{P}(A, B)$ and $\mathcal{P}_{AB}$ to denote the types $\mathcal{P}(\alpha, \beta)[A/\alpha, B/\beta]$ and $\mathcal{P}_{\alpha\beta}[A/\alpha, B/\beta]$, respectively.

To define codes of complex formulas, we need an additional set of types $\mathcal{U}(\alpha)$, for each individual variable α. This set consists of all types of the form

$$(\alpha \to p_i) \to \eta_1$$

for all $\mathcal{P}$ and all $i = 1, 2$, and one more type

$$\alpha \to \eta_2.$$

To simplify notation let us use the following conventions. We write $\Gamma \vdash \mathcal{U}(\alpha)$ to mean that $\Gamma \vdash A$, for all $A \in \mathcal{U}(\alpha)$. An expression of the form "$\mathcal{U}(\alpha) \to B$" stands for "$A_1 \to \cdots A_n \to B$", where $A_1, \ldots, A_n$ are all members of $\mathcal{U}(\alpha)$. We write $\mathcal{U}(\vec{\alpha})$ for the union of all $\mathcal{U}(\alpha)$, where α occurs in $\vec{\alpha}$, and we also use the abbreviation $\mathcal{U}(A)$ for $\mathcal{U}(\alpha)[A/\alpha]$.

The existential and universal formulas are now coded as follows: each atomic formula $\mathcal{P}\alpha\beta$ is replaced by its code $\mathcal{P}(\alpha, \beta)$, and each quantifier $\forall\alpha$ is relativized to $\mathcal{U}(\alpha)$, e.g., a formula of the form $\forall\alpha\beta(\mathcal{P}\alpha\beta \to \cdots)$ is coded as $\forall\alpha\beta(\mathcal{U}(\alpha) \to \mathcal{U}(\beta) \to \mathcal{P}(\alpha, \beta) \to \cdots)$.

We will use the notation $\overline{\varphi}$ for a code of a formula φ. A set Γ of **P**-formulas is represented by an **F**-basis, denoted $\overline{\Gamma}$, consisting of all codes of formulas in Γ, and in addition, of all types $\mathcal{U}(\alpha)$, for each individual variable α that is free in Γ. Our goal is to show the equivalence,

$$\Gamma \vdash_{\mathbf{P}} \mathbf{false} \quad \text{iff} \quad \overline{\Gamma} \vdash_{\mathbf{F}} \mathbf{false},$$

for each finite set Γ of **P**-formulas. The difficult part is the "if" part: if a code of a first-order judgement is derivable in **F**, then such a derivation may be translated back to a valid derivation in **P** of the original judgement (cf. Lemma 10). In other words, no "ad hoc" proof is possible.

From now on, all terms, types, judgements etc. are terms, types, judgements etc. of system **F**, unless stated otherwise. If we refer to formulas of **P**, we mean their codes. If not stated otherwise, all proofs are assumed to be normal and fully applied.

A crucial role in eliminating ad hoc proofs is played by the relativizer $\mathcal{U}(\alpha)$. Lemma 12 below states that each type A satisfying $\mathcal{U}(A)$ can be replaced by an individual variable. To prove this we need the following definitions.

An **F** basis Γ is *good* iff it satisfies the following restrictions:

- it is ordinary, i.e., targets of all types in Γ are free variables;
- no individual variable is a target in Γ;
- p_i is not a target in Γ, for any $\mathcal{P}$ and $i \in \{0, 1, 2\}$;
- the variables η_1,η_2 may occur as targets only in types of the form $\mathcal{U}(\alpha)$;

The last statement should be understood as follows: each occurrence of an η_i is a part of a full set of assumptions of the form $\mathcal{U}(\alpha)$, for some individual variable α.

We say that an individual variable α is an *upper bound* for a type A in a basis Γ iff

$$\Gamma, A \vdash \alpha,$$

A variable α is a *lower bound* for A in Γ iff for every predicate letter $\mathcal{P}$ and every $i \in \{1,2\}$ we have:

$$\Gamma, A \to p_i \ \vdash \alpha \to p_i. \tag{1}$$

Finally, we say that α *represents* A in Γ iff α is both an upper and lower bound.

Lemma 11. *Let β be an upper bound for A in an ordinary basis Γ, where β is not a target. Let α be an individual variable such that the condition (1) above holds for some $\mathcal{P}$ and $i \in \{1,2\}$. In addition, assume that p_i is not a target in Γ. Then $\alpha = \beta$.*

Proof. Assume $\Gamma, A \to p_i \vdash \alpha \to p_i$. Since $\Gamma, A \vdash \beta$, we also have $\Gamma, \beta \to p_i \vdash A \to p_i$, and thus $\Gamma, \beta \to p_i \vdash N : \alpha \to p_i$, for some N. It means that $\Gamma, \beta \to p_i, \alpha \vdash N' : p_i$, where N' may be assumed to be fully applied. There is only one declaration with target p_i, thus we must have $\Gamma, \beta \to p_i, \alpha \vdash \beta$, which is impossible (unless $\beta = \alpha$) as there is no declaration with target β. ∎

Lemma 12. *Let Γ be a good basis, and let $\Gamma \vdash \mathcal{U}(A)$, for some type A. Then there is a (unique) individual variable β representing A in Γ. In addition, Γ must contain the assumptions $\mathcal{U}(\beta)$.*

Proof. It follows immediately from Lemma 11, that there is at most one variable representing A. Because $\Gamma \vdash \mathcal{U}(A)$, we have in particular $\Gamma \vdash A \to \eta_2$, that is, $\Gamma, A \vdash \eta_2$. Since η_2 may occur in Γ only as target of some $\mathcal{U}(\beta)$, we must have $\Gamma, A \vdash N : \beta$, for some term N. We conclude that β is an upper bound. On the other hand, we also have $\Gamma \vdash (A \to p_i) \to \eta_1$, that is, $\Gamma, (A \to p_i) \vdash \eta_1$.

Again, η_1 may occur in Γ only as target of some $\mathcal{U}(\alpha)$, and we get $\Gamma, (A \to p_i) \vdash \alpha \to q_j$. Since q_j is not a target in Γ, it must be that $q_j = p_i$, that is, we actually have $\Gamma, (A \to p_i) \vdash \alpha \to p_i$. From Lemma 11, we conclude that $\alpha = \beta$. Repeating this argument for all $\mathcal{P}$ and i, we get that β is also a lower bound. ∎

A basis Γ is called a **P**-instance iff it has the form $\overline{\Delta} \cup \mathrm{U} \cup \Pi$, where:

- Δ is a set of **P**-formulas;
- U consists of formulas of the form $\mathcal{U}(\alpha)$, where α is an individual variable (not free in Δ);
- Π consists of instances of atomic formulas, i.e., types of the form $\mathcal{P}(A, B)$, such that $\Gamma \vdash \mathcal{U}(A)$ and $\Gamma \vdash \mathcal{U}(B)$.

Clearly, every **P**-instance is a good basis. Thus, by Lemma 12, each of the types substituted into atomic formulas is represented in Γ by a (unique) individual variable.

Let $\Gamma = \overline{\Delta} \cup \mathrm{U} \cup \Pi$ be a **P**-instance as above. By $|\Pi|$ we denote the set of atomic formulas obtained from Π, by replacing each type $\mathcal{P}(A, B)$ with the atomic formula $\mathcal{P}(\alpha, \beta)$, where α and β represent A and B, respectively. Then

by $|\Gamma|$ we denote the set $\Delta \cup |\Pi|$. In the particular case when the last component is empty, we have of course $|\Gamma| = \Delta$.

A crucial step in the proof of our main Lemma 14 is that whenever Γ is a **P**-instance and $\Gamma \vdash \mathcal{P}(A, B)$, with some types A, B, satisfying $\Gamma \vdash \mathcal{U}(A)$ and $\Gamma \vdash \mathcal{U}(B)$, then also $|\Gamma| \vdash_{\mathbf{P}} \mathcal{P}(\alpha, \beta)$, for α, β representing A, B. For the induction to work, we need a slightly more general statement.

Lemma 13. *Let Γ be a* **P***-instance, and let $\Sigma = \{(y_i : \mathcal{P}^i_{A_i B_i}) \mid i = 1, \ldots, n\}$. Assume that $\Gamma, \Sigma \vdash \xi$, and $\Gamma, \Sigma \vdash \mathcal{U}(A_i)$ and $\Gamma, \Sigma \vdash \mathcal{U}(B_i)$, for all $i \in \{1, \ldots, n\}$. Then there exists i such that $|\Gamma| \vdash_{\mathbf{P}} \mathcal{P}^i(\alpha, \beta)$, for some variables α, β, representing A_i and B_i, respectively, in the basis Γ, Σ.*

Proof. By Lemma 12, for all $i = 1, \ldots, n$, there are variables α_i and β_i representing A_i and B_i in Γ, Σ. We begin with showing the following claim:

Suppose that, for some types A and B, we have $\Gamma, \Sigma \vdash N : \mathcal{P}_{AB}$ with $\Gamma, \Sigma \vdash \mathcal{U}(A)$ and $\Gamma, \Sigma \vdash \mathcal{U}(B)$. Then $P = \mathcal{P}^i$, for some i, and the variables α_i and β_i represent A and B, respectively, in the basis Γ, Σ.

To prove the claim, assume that N is fully applied. Thus it must be an abstraction, and we have:

$$\Gamma, \Sigma, (p_0 \to A) \to p_1, (p_0 \to B) \to p_2 \vdash N' : p.$$

Since p is not a target in Γ, the term N' has the form $y_i N_1 N_2$, where $y_i : \mathcal{P}^i_{A_i B_i}$ is in Σ (in particular, $\mathcal{P}^i = \mathcal{P}$). This means that

$$\Gamma, \Sigma, (p_0 \to A) \to p_1, (p_0 \to B) \to p_2 \vdash N_1 : (p_0 \to A_i) \to p_1, \text{ and}$$
$$\Gamma, \Sigma, (p_0 \to A) \to p_1, (p_0 \to B) \to p_2 \vdash N_2 : (p_0 \to B_i) \to p_2,$$

for some $i \in \{1, \ldots, n\}$. Now observe that there are variables α and β, representing A and B in Γ. We have in fact $\alpha_i = \alpha$ and $\beta_i = \beta$. Indeed, $\Gamma, \Sigma, (p_0 \to A_i) \to p_1 \vdash (p_0 \to \alpha_i) \to p_1$ must hold, and it follows that $\Gamma, \Sigma, (p_0 \to A) \to p_1, (p_0 \to B) \to p_2 \vdash (p_0 \to \alpha_i) \to p_1$. By Lemma 11, this means that $\alpha_i = \alpha$. Similar argument yields $\beta_i = \beta$.

The main proof is by induction w.r.t. the size of fully applied proofs (terms). First observe that if $\Gamma, \Sigma \vdash M : \xi$ then M must be an application of a variable z to a sequence of terms and types. We consider two cases, depending on the type of z (the target of which must be ξ).

Case 1. The type of z is an instance of an atomic formula, say $\mathcal{P}(A, B)$. Of course then $M \equiv zN$, where $\Gamma, \Sigma \vdash N : \mathcal{P}_{AB}$. By the above claim, we have $P = \mathcal{P}^i$ and α_i and β_i must represent A and B.

But this means that the formula $P(\alpha_i, \beta_i)$ is in $|\Gamma|$, and thus it is provable.

Case 2. The type of z is a universal formula of the form $\forall \vec{\alpha}(\mathcal{U}(\vec{\alpha}) \to A_1 \to \cdots \to A_n \to \mathcal{P}(\gamma, \delta))$. The term M has the form $M \equiv z\vec{D}\vec{U}\vec{N}L$, where $\vec{D}$ is a sequence of types substituted for $\vec{\alpha}$, the symbol $\vec{U}$ denotes a sequence of proofs for the components of $\mathcal{U}(\vec{D})$, and $\vec{N}$ is a sequence of proofs for types $A_j[\vec{D}/\vec{\alpha}]$. Finally,

L is a proof of $\mathcal{P}_{AB}$, where A and B are the components of $\vec{D}$ corresponding to δ and γ. By Lemma 12, there are (unique) variables $\vec{\beta}$, representing the types $\vec{D}$ in Γ, Σ.

Assume that $N_j \equiv \lambda x : Q_{GH}.N_j'$ is a proof of $A_j[\vec{D}/\vec{\alpha}] = Q(G,H) = Q_{GH} \to \xi$. We apply the induction hypothesis to N_j' in the basis Γ, Σ, Q_{GH}. There are two cases. First suppose that $|\Gamma| \vdash_{\mathbf{P}} \mathcal{P}^i(\alpha,\beta)$, for α,β representing A_i and B_i for some i. Then we are done. Otherwise, $|\Gamma| \vdash_{\mathbf{P}} Q(\alpha,\beta)$ and α,β represent G and H, and this case may be assumed to hold for all j. Clearly, α,β occur in $\vec{\beta}$ at the positions corresponding to occurrences of G and H in $\vec{D}$. Thus, we have $|\Gamma| \vdash_{\mathbf{P}} A_j[\vec{\beta}/\vec{\alpha}]$, for all j.

We conclude that $|\Gamma| \vdash_{\mathbf{P}} \mathcal{P}(\alpha,\beta)$, where α,β are variables occurring in $\vec{\beta}$ at the same positions as γ,δ occur in $\vec{\alpha}$. It suffices to show that $\mathcal{P} = \mathcal{P}^i$, for some i with α,β representing A_i, B_i. This follows from the claim we made at the beginning, applied to the term $L : \mathcal{P}_{AB}$. ∎

Lemma 14. *Let* Γ *be a* **P***-instance. If* $\Gamma \vdash \mathbf{false}$ *then* $|\Gamma| \vdash_{\mathbf{P}} \mathbf{false}$.

Proof. We proceed by induction w.r.t. length of proofs in **F**. Suppose that $\Gamma \vdash M : \mathbf{false}$. As in the previous proof, M must be an application of a variable z to a sequence of terms and types, and we have two cases.

Case 1. The type of z is a universal formula of the form $\forall\vec{\alpha}(\mathcal{U}(\vec{\alpha}) \to A_1 \to \cdots \to A_n \to \mathbf{false})$. Then $M \equiv z\vec{D}\vec{U}\vec{N}$, where $\vec{D}$ is a sequence of types substituted for $\vec{\alpha}$, the symbol $\vec{U}$ denotes a sequence of proofs for the components of $\mathcal{U}(\vec{D})$, and $\vec{N}$ is a sequence of proofs for types $A_j[\vec{D}/\vec{\alpha}]$. For each j there are Q and G, H with $A_j[\vec{D}/\vec{\alpha}] = Q(G,H) = Q_{GH} \to \xi$. The term N_j must be an abstraction, say $N_j \equiv \lambda x{:}Q_{GH}.N_j'$, and we have $\Gamma, x{:}Q_{GH} \vdash N_j' : \xi$. From Lemma 13 (applied to a one-element set $\Sigma = \{(x{:}Q_{GH})\}$) we obtain $|\Gamma| \vdash_{\mathbf{P}} Q(\gamma,\delta)$, where γ,δ represent G and H. In other words, if $\vec{\beta}$ is the sequence of variables representing $\vec{D}$, then for all j we have $|\Gamma| \vdash_{\mathbf{P}} A_j[\vec{\beta}/\vec{\alpha}]$. It follows that $|\Gamma| \vdash_{\mathbf{P}} \mathbf{false}$.

Case 2. The type of z is an existential formula $\forall\vec{\alpha}(\mathcal{U}(\vec{\alpha}) \to A_1 \to \cdots \to A_n \to \forall\gamma(\mathcal{U}(\gamma) \to \mathcal{P}(\delta,\gamma) \to \mathbf{false}) \to \mathbf{false})$. The term M has the form $z\vec{D}\vec{U}\vec{N}L$, where the meaning of $\vec{D}$, $\vec{U}$ and $\vec{N}$ is similar as in Case 1. The term L has type $\forall\gamma(\mathcal{U}(\gamma) \to \mathcal{P}(A,\gamma) \to \mathbf{false})$, where A is the component of $\vec{D}$ corresponding to δ. Using Lemma 13, we obtain as in Case 1, that $|\Gamma| \vdash_{\mathbf{P}} A_i[\vec{\beta}/\vec{\alpha}]$ for all $i \leq n$. Thus, it suffices to show that $|\Gamma| \vdash_{\mathbf{P}} \forall\gamma(\mathcal{P}(\alpha,\gamma) \to \mathbf{false})$, where α is the component of $\vec{\beta}$ corresponding to δ, i.e., representing the type A.

As we observed, L has type $\forall\gamma(\mathcal{U}(\gamma) \to \mathcal{P}(A,\gamma) \to \mathbf{false})$. This means that we have $\Gamma, \mathcal{U}(\gamma), \mathcal{P}(A,\gamma) \vdash \mathbf{false}$, with a shorter proof. By the induction hypothesis, $|\Gamma|, \mathcal{P}(\alpha,\gamma) \vdash_{\mathbf{P}} \mathbf{false}$, where α represents A. It follows that $|\Gamma| \vdash \forall\gamma(\mathcal{P}(\alpha,\gamma) \to \mathbf{false})$, as desired. ∎

Corollary 15. *Let* Δ *be a set of* **P***-formulas. Then* $\overline{\Delta} \vdash \mathbf{false}$ *iff* $\Delta \vdash_{\mathbf{P}} \mathbf{false}$.

It follows that there is an effective reduction from **P** to **F**. Together with Lemma 8, this gives a proof of Löb's theorem.

Theorem 16. *The inhabitation problem for system* **F** *is undecidable.*

Let us note here that the above proof is based on the possibility of nested quantification. The author conjectures that the universal fragment of the second-order propositional intuitionistic logic is decidable.

Acknowledgments

This work originated from e-mail discussions with Daniel Leivant. Morten Heine Sørensen made me aware of the work of Thomas Arts, who provided copies of relevant papers. Aleksy Schubert proofread an initial draft version.

References

1. Arts, T., Embedding first order predicate logic in second order propositional logic, Master Thesis, Katholieke Universiteit Nijmegen, 1992.
2. Arts, T., Dekkers, W., Embedding first order predicate logic in second order propositional logic, Technical report 93-02, Katholieke Universiteit Nijmegen, 1993.
3. Barendregt, H.P., Lambda calculi with types, chapter in: *Handbook of Logic in Computer Science* (S. Abramsky, D.M. Gabbay, T. Maibaum eds.), Oxford University Press, 1990, pp. 117–309.
4. Gabbay, D.M., On 2nd order intuitionistic propositional calculus with full comprehension, *Arch. math. Logik,* **16** (1974), 177–186.
5. Gabbay, D.M., *Semantical Investigations in Heyting's Intuitionistic Logic*, D. Reidel Publ. Co., 1981.
6. Girard, J.-Y., *Proofs and Types,* Cambridge Tracts in Theoretical Computer Science **7**, Cambridge University Press, Cambridge, 1989.
7. Leivant, D., Polymorphic type inference, *Proc. 10-th ACM Symposium on Principles of Programming Languages*, ACM, 1983.
8. Löb, M. H., Embedding first order predicate logic in fragments of intuitionistic logic, *Journal Symb. Logic,* **41**, No. 4 (1976), 705–718.
9. Prawitz, D., Some results for intuitionistic logic with second order quantification rules, in: *Intuitionism and Proof Theory* (A. Kino, J. Myhill, R.E. Vesley eds.), North-Holland, Amsterdam, 1970, pp. 259–270.
10. Rasiowa, H., Sikorski, R., *The Mathematics of Metamathematics*, PWN, Warsaw, 1963.
11. Statman, R., Intuitionistic propositional logic is polynomial-space complete, *Theoretical Computer Science* **9**, 67–72 (1979)
12. Urzyczyn, P., Type reconstruction in $\mathbf{F}_\omega$, to appear in *Mathematical Structures in Computer Science*. Preliminary version in *Proc. Typed Lambda Calculus and Applications* (M. Bezem and J.F. Groote, eds.), LNCS 664, Springer-Verlag, Berlin, 1993, pp. 418–432.
13. Urzyczyn, P., The emptiness problem for intersection types, *Proc. 9th IEEE Symposium on Logic in Computer Science*, 1994, pp. 300–309. (To appear in *Journal of Symbolic Logic.*)
14. Wells, J.B., Typability and type checking in the second-order λ-calculus calculus are equivalent and undecidable, *Proc. 9th IEEE Symposium on Logic in Computer Science*, 1994, pp. 176–185. (To appear in *Annals of Pure and Applied Logic.*)