J. Functional Programming **6** (5): 757–761, September 1996

A simple proof of the undecidability of inhabitation in λP

MARC BEZEM[1] and JAN SPRINGINTVELD[2†]

[1] *Department of Philosophy, Utrecht University, P.O. Box 80126, 3508 TC Utrecht, The Netherlands*
[2] *Computing Science Institute, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands*

Capsule Review

It had been known that the simplest system with dependent types, λP, is undecidable, in that sense that the set

$$\{(A, \Gamma) \mid \exists p\ \Gamma \vdash_{\lambda P} p : A\}$$

is non-computable.

The proof runs as follows. First, there is an obvious embedding of predicate logic into λP. This is the principle idea of one of the basic members of the AUTOMATH family, AUT-QE, and also later of Edinburgh LF. It can be shown that this embedding is conservative (Berardi; Barendsen and Geuvers). This is not completely obvious, since λP has functions of arbitrarily high type at its disposal. Now it follows from Gödel's technique (proving the incompleteness theorems) that arithmetic and even a finitely axiomatizable part of it (Robinson's arithmetic) is essentially undecidable. Therefore λP is also undecidable.

This is quite a long path to the result – admittedly quite beautiful, passing along classical details like the Chinese remainder theorem – but almost too much. Fortunately, the authors of this paper have given a very direct argument showing the same result, by a surprisingly straightforward encoding of a register machine.

An *inhabitation problem* in a given Pure Type System (PTS) λS is a pair $\langle \Gamma, B \rangle$ such that, for some sort s, $\Gamma \vdash_{\lambda S} B : s$. A *solution* for $\langle \Gamma, B \rangle$ is a term A such that $\Gamma \vdash_{\lambda S} A : B$. In this note we prove that in the PTS λP it is undecidable whether an inhabitation problem has a solution. This result also follows from Löb's results on embeddings of predicate logic in fragments of intuitionistic logic (Löb, 1976, p. 1, l. 10) plus the fact that λP is sound and complete with respect to minimal predicate logic (see Geuvers (1993)). The merit of our proof is that it is short, simple and intuitive.

Sometimes, inhabitation problems are defined with Γ the empty context. In λP, however, this makes little sense: the only statement valid in the empty context is $* : \Box$. After the proof of our result, we discuss related results concerning the other systems of the λ-cube. For a general introduction to PTSs the reader is referred to Barendregt (1992).

† Supported by the Netherlands Computer Science Research Foundation (SION) with financial support of the Netherlands Organisation for Scientific Research (NWO).

The machine model we use in this note is the register machine of Shepherdson and Sturgis (1963). It easily follows from Shepherdson and Sturgis (1963) that every partial recursive function can be computed by a register machine program using the instruction set below. As a consequence, the halting problem for register machines is undecidable. We shall encode this halting problem as an inhabitation problem in λP, thus proving that inhabitation in λP is undecidable too. To this end, we translate register machine programs into contexts of λP, similarly to the way in which Shepherdson (1985) translates them into logic programs. The advantage of this machine model and our translation is that it only requires primitives that are already present in λP, thus trivializing the encoding.

Definition 1

A *register machine program* P is a finite sequence $I_1, ..., I_n$ $(n > 0)$ of instructions which operate on registers $x_1, ..., x_m$, where each instruction is of one of the following two forms (with $1 \leq i \leq m$, $1 \leq j \leq n+1$):

$$x_i := x_i + 1 \tag{1}$$

$$\text{IF } x_i \neq 0 \text{ THEN } x_i := x_i - 1 \text{ AND GOTO } j \tag{2}$$

Every program is completed with a HALT instruction I_{n+1}. Execution of a register machine program with respect to given contents $n_1, ..., n_m \in \mathbb{N}$ of the registers $x_1, ..., x_m$ starts from I_1, executing the instructions in the obvious sequential way, and terminates when the HALT instruction I_{n+1} is reached.

The idea of the translation is that registers are modelled as arguments of predicates P_k (all having arity m), with k the program counter, and that state transitions of the register machine by the execution of an instruction are modelled by passing from the conclusion of an implication to its premiss ('backward reasoning'). So the instructions correspond to (universally closed) implications. The HALT instruction corresponds to a predicate P_{n+1} which is unconditionally true. Thus, a terminating execution corresponds to a normal proof of an atomic statement modelling the initial state of the register machine ('extracting proofs from programs'). Via the *formulas as types* interpretation, the formulas corresponding to the instructions can be interpreted as types of λP. Proofs of these formulas correspond to terms of the corresponding types. Thus a terminating execution corresponds to a normal term of an atomic type modelling the initial state of the register machine.

We now explain in detail how instructions of P are translated into types of λP corresponding to the logical statements above.

Definition 2

Let P be a register machine program. The *signature* of P is a context Σ_P defined by (with $N^m \to *$ denoting $N \to *$ if $m = 1$ and $N \to (N^{m-1} \to *)$ if $m > 1$):

$$\Sigma_P = \langle N : *, o : N, s : N \to N, P_1 : N^m \to *, \ldots, P_{n+1} : N^m \to * \rangle.$$

If the instruction I_k of a register machine P is of the form (1), then the corresponding type is:

$$\varphi_k = \Pi x_1 : N \ldots \Pi x_m : N . P_{k+1} x_1 \ldots (s x_i) \ldots x_m \to P_k x_1 \ldots x_i \ldots x_m.$$

https://doi.org/10.1017/S0956796800001969 Published online by Cambridge University Press

If the instruction I_k is of the form (2), then there are two corresponding types, one for each of the two cases $x_i \equiv o$ and (sx_i):

$$\varphi_{ko} = \Pi x_1{:}N \ldots \Pi x_m{:}N.P_{k+1}x_1 \ldots o \ldots x_m \to P_k x_1 \ldots o \ldots x_m,$$

$$\varphi_{ks} = \Pi x_1{:}N \ldots \Pi x_m{:}N.P_j x_1 \ldots x_i \ldots x_m \to P_k x_1 \ldots (sx_i) \ldots x_m.$$

The HALT instruction I_{n+1} corresponds to:

$$\varphi_{n+1} = \Pi x_1{:}N \ldots \Pi x_m{:}N.P_{n+1}x_1 \ldots x_m.$$

The above types are referred to as *φ-types*. The *context* of P is a context Γ_P extending the signature of P with declarations of the form $i_k : \varphi_k$ for all φ-types φ_k. So Γ_P is of the form:

$$\Gamma_P = \Sigma_P, \langle \ldots, i_k : \varphi_k, \ldots, i_{lo} : \varphi_{lo}, i_{ls} : \varphi_{ls}, \ldots \rangle.$$

We need a few technicalities. *Numerals* are defined as usual by: $0 \equiv o$; $n+1 \equiv (sn)$. Furthermore, sequences of numerals are denoted as vectors. Before proving the theorem, we give an example of a register machine program and its translation.

Example 3
We describe a register machine program P operating on one register.

$$\begin{aligned} I_1 &\equiv \text{IF } x \neq 0 \text{ THEN } x := x - 1 \text{ AND GOTO 4} \\ I_2 &\equiv x := x + 1 \\ I_3 &\equiv \text{IF } x \neq 0 \text{ THEN } x := x - 1 \text{ AND GOTO 1} \\ I_4 &\equiv \text{HALT} \end{aligned}$$

It is easy to see that this program terminates on every positive number, returning the predecessor of that number, but that it does not terminate on 0.

Next, we translate P into a context Γ_P of λP. The signature of P is given by

$$\Sigma_P = \langle N : *, o : N, s : N \to N, P_1 : N \to *, \ldots, P_4 : N \to * \rangle.$$

The context Γ_P consists of Σ_P, extended with the following declarations:

$$\begin{aligned} i_{1o} &: \Pi x{:}N.P_2 o \to P_1 o \\ i_{1s} &: \Pi x{:}N.P_4 x \to P_1(sx) \\ i_2 &: \Pi x{:}N.P_3(sx) \to P_2 x \\ i_{3o} &: \Pi x{:}N.P_4 o \to P_3 o \\ i_{3s} &: \Pi x{:}N.P_1 x \to P_3(sx) \\ i_4 &: \Pi x{:}N.P_4 x \end{aligned}$$

It will follow from the theorem below that for all numerals n the type $P_1(sn)$ is inhabited in Γ_P, while $P_1 o$ is not inhabited. To provide an intuitive explanation, we view types again as formulas (reading Π as $\forall$) and in particular the types of the variables $i_{1o}, \ldots, i_4$ as assumptions. We show that $P_1(so)$ is provable from these assumptions and argue that $P_1 o$ is not provable.

https://doi.org/10.1017/S0956796800001969 Published online by Cambridge University Press

To prove $P_1(so)$, use the assumption $\Pi x{:}N.P_4x \to P_1(sx)$, after which one has to prove P_4o. This follows from $\Pi x{:}N.P_4x$. Note that the inhabitant of $P_1(so)$ corresponding to this proof is $i_{1s}o(i_4o)$.

A proof of P_1o would necessarily have the following shape. First, use the assumption $\Pi x{:}N.P_2o \to P_1o$. This yields the obligation to prove P_2o. Next, use $\Pi x{:}N.P_3(sx) \to P_2x$. Then one has to prove $P_3(so)$. For this, use $\Pi x{:}N.P_1x \to P_3(sx)$. This leads back to the original goal P_1o.

Theorem 4

Let P be a register machine program. Then we have: P terminates on $\vec{n}$ if and only if $\Gamma_P \vdash_{\lambda P} M : P_1\vec{n}$ for some term M.

Proof

We start with an important observation: for every type $P_k\vec{n}$ there is exactly one instance of a φ-type whose conclusion matches with $P_k\vec{n}$. The uniqueness of the matching predicate symbol reflects the fact that register machines are deterministic. The uniqueness of the match with respect to the numerals follows from the fact that the matching is first order, so that matching substitutions are unique (if they exist).

'$\Rightarrow$' Assume P terminates on $\vec{n}$. Then we have

$$\Gamma_P \vdash_{\lambda P} i_{x_1}\vec{n}_1(i_{x_2}\vec{n}_2(\ldots(i_{x_p}\vec{n}_p)\ldots)) : P_1\vec{n},$$

where $x_1 = 1$, $x_p = n+1$ and the $\vec{n}_i$'s are suitable instantiations of the φ-types such that the conclusion of the type of $i_{x_1}\vec{n}_1$ matches $P_1\vec{n}$ and, for all $1 \leq q < p$, the conclusion of the type of $i_{x_{q+1}}\vec{n}_{q+1}$ matches the premiss of the type of $i_{x_q}\vec{n}_q$. These instances follow the execution step by step by the observation above.

'$\Leftarrow$' Assume $\Gamma_P \vdash_{\lambda P} M : P_1\vec{n}$ for some term M. Since λP is strongly normalising and the subject reduction property holds, we may assume that M is normal. Now termination of P on $\vec{n}$ follows from the claim below (to be proved by induction on the length of the normal form, using the observation above). □

Claim

If $\Gamma_P \vdash_{\lambda P} M : P_k\vec{n}$ and M is in normal form, then M is of the form $i_{x_1}\vec{n}_1(i_{x_2}\vec{n}_2(\ldots(i_{x_p}\vec{n}_p)\ldots))$.

We end this paper with a discussion of related results for systems of the λ-cube (Barendregt, 1992). Since all systems on the right hand side of the λ-cube are conservative over λP (Geuvers, 1993), the undecidability of inhabitation in these systems follows from the undecidability of inhabitation in λP. Shifting to the left-hand side of the cube, we observe that the proof in this note can easily be adapted to a proof of the undecidability of inhabitation in $\lambda\omega$. This is done by changing all occurrences of N to $*$ and deleting the declaration $N : *$. For example, the signature of P now reads (with $*^m \to *$ denoting $* \to *$ if $m = 1$ and $* \to (*^{m-1} \to *)$ if $m > 1$):

$$\Sigma_P = \langle o : *, s : * \to *, P_1 : *^m \to *, \ldots, P_{n+1} : *^m \to * \rangle.$$

Decidability of inhabitation in $\lambda\to$ can be proved by means of the *formulas as types* interpretation and finite Kripke models. The decidability of inhabitation in $\lambda\underline{\omega}$ is proved in Springintveld (1995) by reducing it to inhabitation in $\lambda\to$. From Löb

https://doi.org/10.1017/S0956796800001969 Published online by Cambridge University Press

(1976), it follows that inhabitation in $\lambda 2$ is undecidable. It would be very interesting to see if the simple proof method in this paper can be transferred to the setting of $\lambda 2$. In fact the adapted proof above is valid for $\lambda 3$, which is $\lambda 2$ with abstraction from variables of type $*^m \to *$ ($m > 0$). As a PTS, $\lambda 2$ is specified by the axiom $* : \Box$ and the rules $(*, *), (\Box, *)$. In addition, $\lambda 3$ has an extra sort $\Box'$, with axiom $* : \Box'$ and rules $(\Box, \Box'), (\Box', *)$. The axiom $* : \Box'$ and the rule $(\Box, \Box')$ justify typings of the form $*^m \to * : \Box'$ in $\lambda 3$. So $\lambda 3$ contexts may contain declarations of the form $s : * \to *$ and $P_k : *^m \to *$. The rule $(\Box', *)$ can be used to abstract from these variables. Note that we only need the presence of these variables in the context, and do not need to abstract from them. This means that our proof holds in a system $\lambda 3^-$, which is $\lambda 3$ minus the rule $(\Box', *)$.

References

Barendregt, H. P. (1992) Lambda calculi with types. In S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 2*, pp. 117–309. Oxford Science Publications.

Geuvers, H. (1993) Conservativity between logics and typed λ calculi. In H. P. Barendregt and T. Nipkow, editors, *Proceedings of the 1st Annual Workshop 'Types for Proofs and Programs' TYPES '93: Nijmegen, The Netherlands, Lecture Notes in Computer Science 806*, pp. 79–108. Springer-Verlag.

Löb, M. H. (1976) Embedding first order predicate logic in fragments of intuitionistic logic. *J. Symbolic Logic* **41** 705–719.

Shepherdson, J. C. (1985) Undecidability of Horn clause logic and pure Prolog. Unpublished manuscript.

Shepherdson, J. C. and Sturgis, H. E. (1963) Computability of recursive functions. *J. ACM* **10**: 217–255.

Springintveld, J. (1995) Algorithms for type theory. Phd thesis, Department of Philosophy, Utrecht University.

https://doi.org/10.1017/S0956796800001969 Published online by Cambridge University Press