Martin-Löf Clashes With Griffin, Operationally

Paul Blain Levy

University of Birmingham, Birmingham B15 2TT, United Kingdom
pbl@cs.bham.ac.uk

Abstract. The computational reading of proofs in intutionistic predicate logic (IPL) as λ-terms can be extended in two well-known ways.
On the one hand, Martin-Löf showed how the Axiom of Choice (AC) can be interpreted as a term of dependent type theory. On the other hand, Griffin showed how Peirce's law (which generates classical logic from intuitionistic) can be interpreted as a program using control operators, and hence we obtain a program from any classical proof.
Together, these suggest the following naive computational interpretation of classical logic with AC. Take a proof in IPL + Peirce + AC and obtain a program using control effects following Griffin, using Martin-Löf's term to translate AC. Then choose an evaluation strategy (call-by-name or call-by-value) and run the program.
This paper shows that the naive approach does not work. We present a proof in IPL + Peirce + AC, which translates to a program that, when executed, regardless of the evaluation strategy chosen, gets stuck—it attempts to pattern-match a λ-abstraction.
For any inhabited type σ, we can restrict AC to σ and still construct this example.

1 Introduction

1.1 Previous Work: Classical Logic With Countable Choice

Much work has been done investigating the computational content of classical logic with Countable Axiom of Choice ($\mathrm{AC}_{\mathtt{nat}}$). This has yielded many positive results [1, 2, 7, 11], but these papers interpret $\mathrm{AC}_{\mathtt{nat}}$ in a quite different (successful) way from the (unsuccessful) Martin-Löf-style interpretation we will consider. So it is the *negative* results listed in [1] which are relevant to this paper, and which we must review.

- Classical logic with $\mathrm{AC}_{\mathtt{nat}}$ contains second-order arithmetic, precluding any predicative model. If proofs do indeed represent programs in some terminating programming language, we will need a strong logic to prove termination.
- There is no cut-free proof of $\mathrm{AC}_{\mathtt{nat}}$ in infinitary propositional logic [12]. This precludes using the game semantics of [3].
- Most importantly, "the halting problem is solvable" in the following sense.

In a (total) type theory containing types `bool` and `nat` and primitive recursion, we can define a term M of type $(\mathtt{nat} \times \mathtt{nat}) \to \mathtt{bool}$ denoting the function

that maps n, t to $\mathtt{true}$ iff program with code n terminates in t steps. Then, in classical predicate logic over this type theory, extended with $\mathrm{AC}_{\mathtt{nat}}$, we can prove

$$\exists \mathtt{f}_{\mathtt{nat}\to\mathtt{bool}}.\forall \mathtt{n}_{\mathtt{nat}}.\mathtt{if}\ (\mathtt{fn}) \begin{cases} \mathtt{then}\ \ \exists \mathtt{t}_{\mathtt{nat}}.\ \mathtt{assert}\ M(\mathtt{n},\mathtt{t}) \\ \mathtt{else}\ \ \neg\exists \mathtt{t}_{\mathtt{nat}}.\ \mathtt{assert}\ M(\mathtt{n},\mathtt{t}) \end{cases} \tag{1}$$

where $\mathtt{assert}\ N$ abbreviates $\mathtt{if}\ N\ \mathtt{then}\ T\ \mathtt{else}\ F$. (Here, T and F are propositions, not terms of type $\mathtt{bool}$.) This precludes using realizability semantics via negative translation, because, in a model where everything is computable, it cannot be realized.

However, this argument does not imply that a computational interpretation for classical logic with $AC_{\mathtt{nat}}$ is impossible. Such interpretations are given in [1, 2, 7]. In this paper, we attempt a different interpretation.

1.2 Proofs To Programs

The propositions-as-types reading [6] can be seen as a translation from intuitionistic predicate logic (IPL) into dependent type theory. Thus the connectives $\vee$,$\wedge$ and $\Rightarrow$ (implication) are translated as $+$, $\times$ and $\to$ (function type) respectively, and the quantifiers $\exists$ and $\forall$ are translated as $\sum$ and $\prod$.

Now, as observed in [9] (developing an idea in [6]), the Axiom of Choice

$$\forall \mathtt{x}_\tau.\exists \mathtt{y}_\sigma.A \Rightarrow \exists \mathtt{f}_{(\prod \mathtt{x}_\tau.\sigma)}.\forall \mathtt{x}_\tau.A[\mathtt{fx}/\mathtt{y}]$$

is translated as the type

$$\prod \mathtt{x}_\tau.\sum \mathtt{y}_\sigma.A \to \sum \mathtt{f}_{(\prod \mathtt{x}_\tau.\sigma)}.\prod \mathtt{x}_\tau.A[\mathtt{fx}/\mathtt{y}]$$

and there is an evident term of this type. Using this term, each proof in IPL+AC is translated as a term of the appropriate dependent type.

In [5], Griffin proposed a translation of classical logic, seen as intuitionistic logic extended with an additional axiom such as *Peirce's Law* $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$. (Other possible axioms are double-negation elimination $((A \Rightarrow \mathrm{F}) \Rightarrow \mathrm{F}) \Rightarrow A$ and excluded middle $A \vee (A \Rightarrow \mathrm{F})$—Griffin treated each of these cases.) The intuitionistic fragment is interpreted in the usual way—Griffin treated propositional rules, but moving to predicate logic presents no difficulty. Peirce's Law (or whichever additional axiom is being used) is interpreted by a program using *control operators*, i.e. operators that bind and change the current stack, such as are present in NJ-SML.

We note that, as an interpretation of classical logic, this is not canonical, because a program with control operators can give different answers depending on the evaluation strategy used. In this paper, we consider both call-by-name and call-by-value evaluation strategies.

Griffin's treatment of classical logic is in fact closely related to the approaches using double-negation translation and game semantics [3]. But it has the apparent advantage that it suggests the following naive way of interpreting classical logic with AC.

Given a proof in IPL + AC + Peirce, we interpret all the rules of IPL following Brouwer, and we interpret AC and Peirce using Martin-Löf's term and Griffin's term respectively. Thus the translation of a proof is a program with control operators. Admittedly there is no known type theory in which this program can be typed, but we can still execute it (in either call-by-name or call-by-value) and see what happens.

We accordingly make the following conjecture.

Conjecture 1 (good behaviour for call-by-name). Let π be a proof in IPL + AC + Peirce. Then the program obtained from π, when executed in call-by-name, terminates in an appropriate terminal form (as specified by the operational semantics).

Similarly, we can conjecture good behaviour for call-by-value.

The reader's reaction may be that these conjectures are obviously false, because of the provability of (1). But, despite some effort, no refutation along these lines has been found, so this reaction appears at present to be unjustified. It is worth remembering that the presence of control operators often invalidates plausible arguments about the behaviour of terms [13], and care is therefore required.

In this paper, we directly refute these conjectures, exhibiting a proof π whose program gets stuck, trying to pattern-match a λ-abstraction. The program is such that its behaviour does not depend on the choice of evaluation strategy.

Surprisingly, we do not need to use the type `nat` to construct our example. It suffices to use AC at `bool` (or indeed any inhabited type, such as $0 \to 0$). This is surprising, because AC at `bool` is already provable in IPL.

1.3 Dependent Types and Computational Effects

A problem in programming language design closely related to the material considered here is how to combine computational effects (imperative features such as storage or control) with dependent types. What can it mean for a type to depend on an effectful term? Certainly, one can consider a two-level language, where upper-level types are dependent on lower-level terms which are effect-free (or, perhaps, can perform limited effects such as divergence). This is what happens in classical predicate logic: the upper-level types are propositions. However, it does not really *combine* dependent types and effects, as they are kept separate.

By contrast, the Martin-Löf AC term mixes the two levels, and so, if our conjecture had been true for some evaluation strategy, that would have given a language where dependent types and effects are genuinely combined. This hope was part of the motivation for this work (the other part being the search for computational content in classical AC). The way in which we refute our conjectures, on the other hand, illustrates where the difficulty lies in achieving such a combination.

We can think of AC as claiming to turn a program performing control effects—the proof of $\forall \mathtt{x}_\tau.\exists \mathtt{y}_\sigma.A$—into an binding for $\mathtt{f} : \prod \mathtt{x}_\tau.\sigma$, which should be

effect-free. The equations for reasoning about `f` are based on the false assumption that it is effect-free, and we will use them to "show" that a term proves a disjunction when in fact it evaluates to a λ-abstraction.

2 Intuitionistic Predicate Logic

2.1 The Type Theory

The identifiers of our predicate logic range over the types of a type theory. This could be a dependent type theory, but a simple type theory, with finite sums and function types, suffices for our purposes. We do not require product types, `bool` or `nat`, although we could include them. We shall generally omit rules, equations etc. for the 0 type, since they are just the nullary analogues of those for $+$. We write Γ for a sequence of distinct identifiers with associated types. The term judgement is $\Gamma \vdash M : \tau$, and the rules are given in Fig. 1. The keyword `pm` is an abbreviation for "pattern-match".

The equation judgement has the form $\Gamma \vdash M = N : \tau$, but to reduce clutter we often omit Γ and τ. We omit all the assumptions required to make the equational laws well-typed: e.g. in the η-law for function type, M must be of type $A \to B$, and in the η-law for sum type, `z` and N must be of type $A + A'$. The rules for this judgement are given in Fig. 1.

We write ${}^{\mathtt{x}}M$ to mean the weakening of M by adding `x` to its context Γ—it is presumed that Γ does not already include `x`. This notation is more convenient that the traditional "$\mathtt{x} \notin \mathsf{FV}(M)$", because it makes clear where the weakening takes place.

2.2 Logic

The propositions of the logic are given by a judgement $\Gamma \vdash A\ \mathsf{prop}$ defined in Fig. 2. We do not require finite conjunction, although we could include it. We also do not need to make the equations of the type theory into propositions, although we could do this. We generally omit rules etc. for F because they are just the nullary analogues of those for $\vee$.

We have a proposition equality judgement $\Gamma \vdash A = A'$, defined in Fig. 2. We often omit Γ to reduce clutter, and omit the typing assumptions necessary to make these equations well-typed.

The sequent of the logic is $\Gamma | \Phi \vdash B$, where Φ is a sequence of propositions A such that $\Gamma \vdash A\ \mathsf{prop}$, and $\Gamma \vdash B\ \mathsf{prop}$. The rules of the logic are given in Fig. 3.

3 The Target Language

3.1 Syntax

We are going to translate proofs into an untyped language with control operators, whose syntax is presented in Fig. 4. For the moment, ignore the control operators `letstk` and `changestk`—we shall explain these in Sect. 3.4.

Types

$$\tau ::= \quad 0 \mid \tau + \tau \mid \tau \to \tau$$

Terms

$$\frac{}{\Gamma, \mathtt{x} : \tau, \Gamma' \vdash \mathtt{x} : \tau}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \mathtt{inl}\ M : \tau + \tau'} \qquad \frac{\Gamma \vdash M : \tau'}{\Gamma \vdash \mathtt{inr}\ M : \tau + \tau'}$$

$$\frac{\Gamma \vdash M : \tau + \tau' \quad \Gamma, \mathtt{x} : \tau \vdash N : \sigma \quad \Gamma, \mathtt{x} : \tau \vdash N' : \sigma}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} : \sigma}$$

$$\frac{\Gamma, \mathtt{x} : \tau \vdash M : \sigma}{\Gamma \vdash \lambda \mathtt{x}.M : \tau \to \sigma} \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}$$

Equations congruence rules together with

$$\begin{array}{lrcl}
(\beta+) & \mathtt{pm}\ (\mathtt{inl}\ M)\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} & = & N[M/\mathtt{x}] \\
(\beta+) & \mathtt{pm}\ (\mathtt{inr}\ M)\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} & = & N'[M/\mathtt{x}] \\
(\beta \to) & (\lambda \mathtt{x}.M)N & = & N[M/\mathtt{x}]
\end{array}$$

$$\begin{array}{lrcl}
(\eta+) & M[N/\mathtt{z}] & = & \mathtt{pm}\ N\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.{}^{\mathtt{x}}M[\mathtt{inl}\ \mathtt{x}/\mathtt{z}], \mathtt{inr}\ \mathtt{x}.{}^{\mathtt{x}}M[\mathtt{inr}\ \mathtt{x}/\mathtt{z}]\} \\
(\eta \to) & M & = & \lambda \mathtt{x}.({}^{\mathtt{x}}M\mathtt{x})
\end{array}$$

Fig. 1. Type Theory

Propositions

$$\frac{}{\Gamma \vdash \mathrm{F}\ \mathsf{prop}} \qquad \frac{\Gamma \vdash A\ \mathsf{prop} \quad \Gamma \vdash A'\ \mathsf{prop}}{\Gamma \vdash A \vee A'\ \mathsf{prop}}$$

$$\frac{\Gamma \vdash A\ \mathsf{prop} \quad \Gamma \vdash B\ \mathsf{prop}}{\Gamma \vdash A \to B\ \mathsf{prop}}$$

$$\frac{\Gamma, \mathtt{x} : \tau \vdash A\ \mathsf{prop}}{\Gamma \vdash \exists \mathtt{x}_\tau.A\ \mathsf{prop}} \qquad \frac{\Gamma, \mathtt{x} : \tau \vdash A\ \mathsf{prop}}{\Gamma \vdash \forall \mathtt{x}_\tau.A\ \mathsf{prop}}$$

$$\frac{\Gamma \vdash M : \tau + \tau' \quad \Gamma, \mathtt{x} : \tau \vdash A\ \mathsf{prop} \quad \Gamma, \mathtt{x} : \tau' \vdash A'\ \mathsf{prop}}{\Gamma \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.A, \mathtt{inr}\ \mathtt{x}.A'\}\ \mathsf{prop}}$$

Equations Between Propositions congruence rules together with

$$\begin{array}{lrcl}
(\beta+) & \mathtt{pm}\ (\mathtt{inl}\ M)\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.A, \mathtt{inr}\ \mathtt{x}.A'\} & = & A[M/\mathtt{x}] \\
(\beta+) & \mathtt{pm}\ (\mathtt{inr}\ M)\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.A, \mathtt{inr}\ \mathtt{x}.A'\} & = & A'[M/\mathtt{x}]
\end{array}$$

$$(\eta+)\ A[N/\mathtt{z}] = \mathtt{pm}\ N\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.{}^{\mathtt{x}}A[\mathtt{inl}\ \mathtt{x}/\mathtt{z}], \mathtt{inr}\ \mathtt{x}.{}^{\mathtt{x}}A[\mathtt{inr}\ \mathtt{x}/\mathtt{z}]\}$$

Fig. 2. Propositions

Intuitionistic Predicate Logic

$$\frac{\Gamma|\overrightarrow{A_i} \vdash B \quad \Gamma \vdash \overrightarrow{A_i = A_i'} \quad \Gamma \vdash B = B'}{\Gamma|\overrightarrow{A_i'} \vdash B'} \qquad \frac{}{\Gamma|\Phi, A, \Phi' \vdash A}$$

$$\frac{\Gamma|\Phi \vdash A}{\Gamma|\Phi \vdash A \vee A'} \qquad \frac{\Gamma|\Phi \vdash A'}{\Gamma|\Phi \vdash A \vee A'}$$

$$\frac{\Gamma|\Phi \vdash A \vee A' \quad \Gamma|\Phi, A \vdash B \quad \Gamma|\Phi, A' \vdash B}{\Gamma|\Phi \vdash B}$$

$$\frac{\Gamma|\Phi, A \vdash B}{\Gamma|\Phi \vdash A \Rightarrow B} \qquad \frac{\Gamma|\Phi \vdash A \Rightarrow B \quad \Gamma|\Phi \vdash A}{\Gamma|\Phi \vdash B}$$

$$\frac{\Gamma \vdash M : \tau \quad \Gamma|\Phi \vdash A[M/\mathtt{x}]}{\Gamma|\Phi \vdash \exists \mathtt{x}_\tau.A} \qquad \frac{\Gamma|\Phi \vdash \exists \mathtt{x}_\tau.A \quad \Gamma, \mathtt{x} : \tau|{}^{\mathtt{x}}\Phi, A \vdash {}^{\mathtt{x}}B}{\Gamma|\Phi \vdash B}$$

$$\frac{\Gamma, \mathtt{x} : \tau|{}^{\mathtt{x}}\Phi \vdash B}{\Gamma|\Phi \vdash \forall \mathtt{x}_\tau.B} \qquad \frac{\Gamma|\Phi \vdash \forall \mathtt{x}_\tau.B \quad \Gamma \vdash N : \tau}{\Gamma|\Phi \vdash B[N/\mathtt{x}]}$$

$$\frac{\Gamma \vdash M : \tau + \tau' \quad \Gamma, \mathtt{x} : \tau|\overrightarrow{A_i} \vdash B \quad \Gamma, \mathtt{x} : \tau'|\overrightarrow{A_i'} \vdash N' : B'}{\Gamma|\overrightarrow{\mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.A_i, \mathtt{inr}\ \mathtt{x}.A_i'\}} \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.B, \mathtt{inr}\ \mathtt{x}.B'\}}$$

Axiom of Choice at type σ

$$\Gamma|\Phi \vdash (\forall \mathtt{y}_\sigma.\exists \mathtt{v}_\tau.B) \Rightarrow (\exists \mathtt{g}_{\sigma\to\tau}.\forall \mathtt{y}_\sigma.B[\mathtt{gy}/\mathtt{v}])$$

Peirce's Law

$$\Gamma|\Phi \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$$

Fig. 3. Logic

$$\begin{aligned} M ::= \ & \mathtt{x} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{if}\ M\ \mathtt{then}\ N\ \mathtt{else}\ N' \\ & \mathtt{inl}\ M \mid \mathtt{inr}\ M \mid \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.M, \mathtt{inr}\ \mathtt{x}.M\} \\ & \mid \mathtt{pm}\ M\ \mathtt{as}\ \{\} \mid (M, M') \mid \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).M \mid \lambda \mathtt{x}.M \mid MM \\ & \mid \mathtt{letstk}\ \alpha.\ M \mid \mathtt{changestk}\ \alpha.\ M \end{aligned}$$

Fig. 4. Target Language (Excluding Stack Terms)

3.2 Call-By-Name Semantics: CK-Machine

Initial Configurations

	M	$\texttt{nil}$

Transitions (we omit the transition for `inr`)

	$\texttt{pm}\ M\ \texttt{as}\ \ldots$	K
$\rightsquigarrow$	M	$\texttt{pm}\ [\cdot]\ \texttt{as}\ \ldots :: K$
	$\texttt{inl}\ M$	$\texttt{pm}\ [\cdot]\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\} :: K$
$\rightsquigarrow$	$N[M/\texttt{x}]$	K
	(M, M')	$\texttt{pm}\ [\cdot]\ \texttt{as}\ (\texttt{x}, \texttt{y}).N :: K$
$\rightsquigarrow$	$N[M/\texttt{x}, M'/\texttt{y}]$	K
	MN	K
$\rightsquigarrow$	M	$[\cdot]N :: K$
	$\lambda\texttt{x}.M$	$[\cdot]N :: K$
$\rightsquigarrow$	$M[N/\texttt{x}]$	K

Terminal Configurations

$\lambda\texttt{x}.M$	$\texttt{nil}$
$\texttt{inl}\ M$	$\texttt{nil}$
$\texttt{inr}\ M$	$\texttt{nil}$
(M, M')	$\texttt{nil}$

Fig. 5. CK-Machine For Call-By-Name, Excluding Control Operators

We will give operational semantics in the form of a *CK-machine* [4]. In this machine a configuration is a pair M, K where M is the closed term we are currently evaluating and K is a stack of contexts. The call-by-name machine is shown in Fig. 5.

As an example of how this machine works, consider how we would evaluate $\texttt{pm}\ M\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\}$. We firstly need to evaluate M, so the rest of the term, viz. the context $\texttt{pm}\ [\cdot]\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\}$, is placed on the stack. Eventually, we will finish evaluating M, say into $\texttt{inl}\ M'$. (In call-by-name, we do not proceed to evaluate M'.) Now we can remove the context from the stack and continue to evaluate $N[M'/\texttt{x}]$.

The CK-machine is clearly deterministic, in the sense that if a configuration is terminal, there are no transitions from it, and if not, then there is most one transition from it. A non-terminal configuration from which there are no transitions is called *stuck*.

3.3 Call-By-Value Semantics: CK/VK-Machine

In the "left-to-right" variant of call-by-value that we present, the operator in an application is evaluated before the operand, and the left component of a tuple before the right component. It is easy to see how to modify our presentation to reverse either or both of these conventions.

Call-by-value semantics is described in terms of a special class of closed terms called *values*:

$$V ::= \quad \mathtt{inl}\ V \mid \mathtt{inr}\ V \mid (V, V) \mid \lambda \mathtt{x}.M$$

Now if we try to formulate a CK-machine for call-by-value in the same way that we did for call-by-name, we might propose a transition such as

$$\begin{array}{rll} & V & \lambda \mathtt{x}.M[\cdot] :: K \\ \rightsquigarrow & M[V/\mathtt{x}] & K \end{array}$$

But this is not acceptable, because determining whether a term is a value might take many steps. (This problem did not arise for the language of [4], because it did not contain tuples.) To avoid this problem, we use instead the *CK/VK-machine*, with two different kinds of configuration:

- $M < K$ where M is any closed term
- $V > K$ where V is a value.

The symbol between the term and the stack is an arrow indicating whether we are currently working on the term or on the stack. The CK-machine is presented in Fig. 6.

3.4 Control Operators

Control operators provide access to the stack. The syntax varies in different languages, but essentially there is a command `letstk` α that binds the current stack[1] to a *stack identifier* α, and a command `changestk` α that changes the current stack to α. We folllow [10] in using Greek letters for stack identifiers.

The syntax of programs with control operators is shown in Fig. 4. Note that `letstk` α binds α while `changestk` α does not. The additional transitions are shown in Fig. 7. If $M < K$ is a configuration, M might not conform to Fig. 4, because it might contain stacks. This could be rectified by extending the syntax, but we will not trouble to do this.

4 Translating Proofs To Programs

We want to translate a proof π of $\Gamma|\Phi \vdash B$ into a term M whose free identifiers are those of Γ together with an identifier $\mathtt{p}$ for each $A \in \Phi$. There should be no free stack identifiers in M. We write

$$\frac{\pi}{\Gamma|\mathtt{p}_0 : A_0, \ldots, \mathtt{p}_{n-1} : A_{n-1} \vdash M : B}$$

[1] In the call-by-value setting, the current stack is called the *current continuation*, but in call-by-name, a stack might not be a continuation, as explained in [8].

Initial Configurations

$$M \quad < \quad \texttt{nil}$$

Transitions (we omit the transitions for `inr`)

$$\begin{array}{lll}
 & \texttt{pm}\ M\ \texttt{as}\ \ldots & < \quad K \\
\rightsquigarrow & M & < \quad \texttt{pm}\ [\cdot]\ \texttt{as}\ \ldots :: K \\
 & & \\
 & \texttt{inl}\ M & < \quad K \\
\rightsquigarrow & M & < \quad \texttt{inl}\ [\cdot] :: K \\
 & & \\
 & V & > \quad \texttt{inl}\ [\cdot] :: K \\
\rightsquigarrow & \texttt{inl}\ V & > \quad K \\
 & & \\
 & \texttt{inl}\ V & > \quad \texttt{pm}\ [\cdot]\ \texttt{as}\ \{\texttt{inl}\ \texttt{x}.N, \texttt{inr}\ \texttt{x}.N'\} :: K \\
\rightsquigarrow & N[V/\texttt{x}] & < \quad K \\
 & & \\
 & (M, M') & < \quad K \\
\rightsquigarrow & M & < \quad ([\cdot], M') :: K \\
 & & \\
 & V & > \quad ([\cdot], M') :: K \\
\rightsquigarrow & M' & < \quad (V, [\cdot]) :: K \\
 & & \\
 & V' & > \quad (V, [\cdot]) :: K \\
\rightsquigarrow & (V, V') & > \quad K \\
 & & \\
 & (V, V') & > \quad \texttt{pm}\ [\cdot]\ \texttt{as}\ (\texttt{x}, \texttt{y}).N :: K \\
\rightsquigarrow & N[V/\texttt{x}, V, \texttt{y}] & < \quad K \\
 & & \\
 & MN & < \quad K \\
\rightsquigarrow & M & < \quad [\cdot]N :: K \\
 & & \\
 & \lambda\texttt{x}.M & < \quad K \\
\rightsquigarrow & \lambda\texttt{x}.M & > \quad K \\
 & & \\
 & \lambda\texttt{x}.M & > \quad [\cdot]N :: K \\
\rightsquigarrow & N & < \quad \lambda\texttt{x}.M[\cdot] :: K \\
 & & \\
 & V & > \quad \lambda\texttt{x}.M[\cdot] :: K \\
\rightsquigarrow & M[V/\texttt{x}] & < \quad K
\end{array}$$

Terminal Configurations

$$V \quad > \quad \texttt{nil}$$

Fig. 6. CK/VK-Machine For Call-By-Value (Left-to-Right Variant), Excluding Control Operators

For call-by-name, ignore the $<$.

$$\begin{array}{lll} & \texttt{letstk}\ \alpha.\ M & < \quad K \\ \rightsquigarrow & M[K/\alpha] & < \quad K \\ \\ & \texttt{changestk}\ K.\ M & < \quad K' \\ \rightsquigarrow & M & < \quad K \end{array}$$

Fig. 7. Transitions For Control Operators

to indicate that the translation of the proof π is M. The translation is presented in Fig. 8. It uses the fact that a term of the type theory is already, by erasure of types, a term of the target language.

5 The Clash

We now want to prove the following.

Proposition 1. *Let σ be a type (e.g. $0 \to 0$) inhabited by a closed term d. There is a proof ϕ in IPL + AC_σ + Peirce, whose proofterm we call t, and a configuration $V > K$, such that, both in call-by-name and in all the variants of call-by-value*

- $t < \texttt{nil} \rightsquigarrow^* V > K$
- *$V > K$ is stuck.*

For call-by-name, ignore the $<$ and $>$.

In our proof, we write ν for $(\sigma \to (0+0)) + \sigma$, and type identifiers as follows.

$$\begin{array}{r} \mathtt{w} : 0 \\ \mathtt{x}, \mathtt{y} : \sigma \\ \mathtt{g}, \mathtt{g}', \mathtt{g}'' : \sigma \to (0+0) \\ \mathtt{v} : 0+0 \\ \mathtt{u} : \nu \\ \mathtt{f} : \sigma \to \nu \end{array}$$

The core of our example is the equation shown in Fig. 9. Consider the term obtained from the LHS by replacing $(\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}$ by $\texttt{inl}\ \lambda\mathtt{z}.\mathtt{z}$ and $\mathrm{F} \Rightarrow \mathrm{F}$ by $\lambda\mathtt{z}.\mathtt{z}$. It is clearly a proofterm of the LHS, and hence, by our equation, of $(\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}$. This is reasonable in the effect-free setting; the code arising from evaluating $\mathtt{f}d$ first to `inl` and then to `inr` is dead code.

But suppose `f` is bound to

$$\lambda\mathtt{x}.\ \texttt{letstk}\ \alpha.\ \texttt{inl}\ \lambda\mathtt{y}.\ \texttt{changestk}\ \alpha.\texttt{inr}\ d$$

Intuitionistic Predicate Logic

$$\frac{\Gamma|\overrightarrow{A_i} \vdash M : B \quad \Gamma \vdash \overrightarrow{A_i = A_i'} \quad \Gamma \vdash B = B'}{\Gamma|\overrightarrow{A_i'} \vdash M : B'} \qquad \frac{}{\Gamma|\Phi, \mathtt{p} : A, \Phi' \vdash \mathtt{p} : A}$$

$$\frac{\Gamma|\Phi \vdash M : A}{\Gamma|\Phi \vdash \mathtt{inl}\ M : A \vee A'} \qquad \frac{\Gamma|\Phi \vdash M : A'}{\Gamma|\Phi \vdash \mathtt{inr}\ M : A \vee A'}$$

$$\frac{\Gamma|\Phi \vdash M : A \vee A' \quad \Gamma|\Phi, \mathtt{p} : A \vdash N : B \quad \Gamma|\Phi, \mathtt{p} : A' \vdash N' : B}{\Gamma|\Phi \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{p}.N, \mathtt{inr}\ \mathtt{p}.N'\} : B}$$

$$\frac{\Gamma|\Phi, \mathtt{p} : A \vdash M : B}{\Gamma|\Phi \vdash \lambda \mathtt{p}.M : A \Rightarrow B} \qquad \frac{\Gamma|\Phi \vdash M : A \Rightarrow B \quad \Gamma|\Phi \vdash N : A}{\Gamma|\Phi \vdash MN : B}$$

$$\frac{\Gamma \vdash M : \tau \quad \Gamma|\Phi \vdash N : A[M/\mathtt{x}]}{\Gamma|\Phi \vdash (M, N) : \exists \mathtt{x}_\tau.A} \qquad \frac{\Gamma|\Phi \vdash M : \exists \mathtt{x}_\tau.A \quad \Gamma, \mathtt{x} : \tau|\,^{\mathtt{x}}\Phi, \mathtt{p} : A \vdash N :\, ^{\mathtt{x}}B}{\Gamma|\Phi \vdash \mathtt{pm}\ M\ \mathtt{as}\ (\mathtt{x}, \mathtt{p}).\ N : B}$$

$$\frac{\Gamma, \mathtt{x} : \tau|\,^{\mathtt{x}}\Phi \vdash M : B}{\Gamma|\Phi \vdash \lambda \mathtt{x}.M : \forall \mathtt{x}_\tau.B} \qquad \frac{\Gamma|\Phi \vdash M : \forall \mathtt{x}_\tau.B \quad \Gamma \vdash N : \tau}{\Gamma|\Phi \vdash MN : B[N/\mathtt{x}]}$$

$$\frac{\Gamma \vdash M : \tau + \tau' \quad \Gamma, \mathtt{x} : \tau|\overrightarrow{\mathtt{p}_i : A_i} \vdash B \quad \Gamma, \mathtt{x} : \tau'|\overrightarrow{\mathtt{p}_i : A_i'} \vdash N' : B'}{\Gamma|\overrightarrow{\mathtt{p}_i : \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.A_i, \mathtt{inr}\ \mathtt{x}.A_i'\}} \vdash \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.N, \mathtt{inr}\ \mathtt{x}.N'\} : \mathtt{pm}\ M\ \mathtt{as}\ \{\mathtt{inl}\ \mathtt{x}.B, \mathtt{inr}\ \mathtt{x}.B'\}}$$

Axiom of Choice at type σ

$$\Gamma|\Phi \vdash \lambda \mathtt{p}.((\lambda \mathtt{y}.\mathtt{pm}\ (\mathtt{py})\ \mathtt{as}\ (\mathtt{z}, \mathtt{q}).\mathtt{z}), (\lambda \mathtt{y}.\mathtt{pm}\ (\mathtt{py})\ \mathtt{as}\ (\mathtt{z}, \mathtt{q}).\mathtt{q})) : (\forall \mathtt{y}_\sigma.\exists \mathtt{v}_\tau.B) \Rightarrow (\exists \mathtt{g}_{\sigma\to\tau}.\forall \mathtt{y}_\sigma.B[\mathtt{gy}/\mathtt{v}])$$

Peirce's Law

$$\Gamma|\Phi \vdash \lambda \mathtt{p}.\mathtt{letstk}\ \alpha.(\mathtt{p}\lambda \mathtt{q}.(\mathtt{changestk}\ \alpha.\ \mathtt{q})) : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$$

Fig. 8. Proofterm assignment

The first time that $\mathtt{f}d$ is evaluated, we move into the `inl` branch, and again the second time. But then when $\mathtt{g}'d$ is evaluated, the stack from the second time is restored, and we move into the `inr` branch and execute that "dead code", giving $\lambda\mathtt{z}.\mathtt{z}$. The attempt to pattern-match this will lead to a stuck configuration.

This argument can be turned into a proof ϕ, as shown in Fig. 10–12. To ensure that it behaves the same in CBN and CBV, we protect all arguments with a λ, and use pattern-matching to ensure sequencing.

$$
\begin{array}{rl}
 & \mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}.\ \mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}'.\ \mathtt{pm}\ (\mathtt{g}'d)\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{w}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\\ \mathtt{inr}\ \mathtt{w}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ \mathrm{F}\Rightarrow\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right.\\
=(\eta) & \mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}.\ \mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}'.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\\ \mathtt{inr}\ \mathtt{y}.\ \mathrm{F}\Rightarrow\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right.\\
= & \mathtt{pm}\ \mathtt{u}\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}.\ \mathtt{pm}\ \mathtt{u}\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}'.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\\ \mathtt{inr}\ \mathtt{y}.\ \mathrm{F}\Rightarrow\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right. \qquad [\mathtt{f}d/\mathtt{u}]\\
=(\eta) & \mathtt{pm}\ \mathtt{f}d\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}''\ \mathtt{pm}\ \mathtt{inl}\ \mathtt{g}''\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}.\ \mathtt{pm}\ \mathtt{inl}\ \mathtt{g}''\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}'.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\\ \mathtt{inr}\ \mathtt{y}.\ \mathrm{F}\Rightarrow\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ \mathtt{pm}\ \mathtt{inr}\ \mathtt{y}\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}.\ \mathtt{pm}\ \mathtt{inr}\ \mathtt{y}\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}'.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\\ \mathtt{inr}\ \mathtt{y}.\ \mathrm{F}\Rightarrow\mathrm{F}\end{array}\right.\\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right.\end{array}\right.\\
=(2\times\beta) & \mathtt{pm}\ \mathtt{f}d\ \mathtt{as}\ \left\{\begin{array}{l}\mathtt{inl}\ \mathtt{g}''\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}\end{array}\right.\\
=(\eta) & (\mathrm{F}\Rightarrow\mathrm{F})\vee\mathrm{F}
\end{array}
$$

Fig. 9. The Key Equation

Acknowledgement

I am grateful to Stefano Berardi for helpful discussion.

References

1. Stefano Berardi, Marc Bezem, and Thierry Coquand. On the computational content of the Axiom of Choice. *The Journal of Symbolic Logic*, 63(2):600–622, 1998.
2. U. Berger and P. Oliva. Modified bar recursion and classical dependent choice. Logic Colloquium 2001, to appear.
3. T. Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, 60(1):325–337, March 1995.

We write

C for $\forall \mathtt{x}.(\mathrm{F} \Rightarrow \mathrm{F})$

C_0 for $\mathtt{pm}\ (\mathtt{g}'d)\ \mathtt{as} \begin{cases} \mathtt{inl}\ \mathtt{w}.C \\ \mathtt{inr}\ \mathtt{w}.C \end{cases}$ $\quad$ B_0 for $\mathtt{pm}\ (\mathtt{g}'d)\ \mathtt{as} \begin{cases} \mathtt{inl}\ \mathtt{w}.\ (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F} \\ \mathtt{inr}\ \mathtt{w}.\ (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F} \end{cases}$

C_1 for $\mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as} \begin{cases} \mathtt{inl}\ \mathtt{g}'.\ C \\ \mathtt{inr}\ \mathtt{y}.\ C \end{cases}$ $\quad$ B_1 for $\mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as} \begin{cases} \mathtt{inl}\ \mathtt{g}'.\ B_0 \\ \mathtt{inr}\ \mathtt{y}.\ \mathrm{F} \Rightarrow \mathrm{F} \end{cases}$

C_2 for $\mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as} \begin{cases} \mathtt{inl}\ \mathtt{g}.\ C \\ \mathtt{inr}\ \mathtt{y}.\ C \end{cases}$ $\quad$ B_2 for $\mathtt{pm}\ (\mathtt{f}d)\ \mathtt{as} \begin{cases} \mathtt{inl}\ \mathtt{g}.\ B_1 \\ \mathtt{inr}\ \mathtt{y}.\ (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F} \end{cases}$

Let π_0 be the following proof:

$$\dfrac{\dfrac{\mathtt{f},\mathtt{g},\mathtt{g}',\mathtt{w}|C,\mathrm{F} \vdash \mathrm{F}}{\mathtt{f},\mathtt{g},\mathtt{g}',\mathtt{w}|C \vdash \mathrm{F} \Rightarrow \mathrm{F}} \quad \dfrac{\mathtt{f},\mathtt{g},\mathtt{g}',\mathtt{w}|C,\mathrm{F} \vdash \mathrm{F}}{\mathtt{f},\mathtt{g},\mathtt{g}',\mathtt{w}|C \vdash \mathrm{F} \Rightarrow \mathrm{F}}}{\dfrac{\mathtt{f},\mathtt{g},\mathtt{g}' \vdash \mathtt{g}'d : 0+0 \quad \mathtt{f},\mathtt{g},\mathtt{g}',\mathtt{w}|C \vdash (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}}{\mathtt{f},\mathtt{g},\mathtt{g}'|C_0 \vdash B_0} \quad \mathtt{f},\mathtt{g},\mathtt{g}' \vdash C_0 = C}{\mathtt{f},\mathtt{g},\mathtt{g}'|C \vdash B_0}$$

Let π_1 be the following proof:

$$\dfrac{\mathtt{f} \vdash \mathtt{f}d : \nu \quad \dfrac{\pi_0}{\mathtt{f},\mathtt{g},\mathtt{g}'|C \vdash B_0} \quad \dfrac{\mathtt{f},\mathtt{g},\mathtt{y}|C,\mathrm{F} \vdash \mathrm{F}}{\mathtt{f},\mathtt{g},\mathtt{y}|C \vdash \mathrm{F} \Rightarrow \mathrm{F}}}{\mathtt{f},\mathtt{g}|C_1 \vdash B_1} \quad \mathtt{f},\mathtt{g} \vdash C_1 = C}{\mathtt{f},\mathtt{g}|C \vdash B_1}$$

Let π be the following proof:

$$\dfrac{\mathtt{f} \vdash \mathtt{f}d : \nu \quad \dfrac{\pi_1}{\mathtt{f},\mathtt{g}|C \vdash B_1} \quad \dfrac{\dfrac{\mathtt{f},\mathtt{y}|C,\mathrm{F} \vdash \mathrm{F}}{\mathtt{f},\mathtt{y}|C \vdash \mathrm{F} \Rightarrow \mathrm{F}}}{\mathtt{f},\mathtt{y}|C \vdash (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}}}{\mathtt{f}|C_2 \vdash B_2} \quad \mathtt{f} \vdash C_2 = C \quad \mathtt{f} \vdash B_2 = (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}}{\mathtt{f}|C \vdash (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}}$$

where the bottom right premise is proved in Fig. 9.

Fig. 10. Construction of π

Write D for $(\exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})) \Rightarrow (\exists \mathtt{v}.(\mathrm{F} \Rightarrow \mathrm{F}))$.
Let θ_0 be the following proof:

$$\dfrac{\dfrac{\dfrac{}{\mathtt{x}, \mathtt{y} \vert D \vert \mathrm{F} \vdash \mathrm{F}}}{\mathtt{x}, \mathtt{y} \vert D \vert \mathrm{F} \Rightarrow \mathrm{F}} \quad \dfrac{}{\mathtt{x}, \mathtt{y} \vdash \mathtt{inr}\ d : \nu}}{\mathtt{x}, \mathtt{y} \vert D \vdash \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})} \quad \dfrac{}{\mathtt{x}, \mathtt{y} \vert D \vdash (\exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})) \Rightarrow (\exists \mathtt{v}.(\mathrm{F} \Rightarrow \mathrm{F}))}}{\dfrac{\mathtt{x}, \mathtt{y} \vert D \vdash \exists \mathtt{v}.(\mathrm{F} \Rightarrow \mathrm{F})}{\mathtt{x} \vert D \vdash \forall \mathtt{y}.\exists \mathtt{v}.(\mathrm{F} \Rightarrow \mathrm{F})}}$$

Let θ_1 be the following proof:

$$\dfrac{\dfrac{}{\mathtt{x} \vert D \vdash (\forall \mathtt{y}.\exists \mathtt{v}.(\mathrm{F} \Rightarrow \mathrm{F})) \Rightarrow (\exists \mathtt{g}.\forall \mathtt{y}.(\mathrm{F} \Rightarrow \mathrm{F}))} \quad \dfrac{\theta_0}{\mathtt{x} \vert D \vdash \forall \mathtt{y}.\exists \mathtt{v}.(\mathrm{F} \Rightarrow \mathrm{F})}}{\mathtt{x} \vert D \vdash \exists \mathtt{g}.\forall \mathtt{y}.(\mathrm{F} \Rightarrow \mathrm{F})}$$

where the left premise is an instance of AC_σ.
Let θ_2 be the following proof:

$$\dfrac{\dfrac{}{\mathtt{x} \vert D \vdash \exists \mathtt{g}.\forall \mathtt{y}.(\mathrm{F} \Rightarrow \mathrm{F})} \quad \dfrac{\dfrac{\dfrac{}{\mathtt{x}, \mathtt{g} \vert D, \forall \mathtt{y}.(\mathrm{F} \Rightarrow \mathrm{F}), \mathrm{F} \vdash \mathrm{F}}}{\mathtt{x}, \mathtt{g} \vert D, \forall \mathtt{y}.(\mathrm{F} \Rightarrow \mathrm{F}) \vdash \mathrm{F} \Rightarrow \mathrm{F}} \quad \dfrac{}{\mathtt{x}, \mathtt{g} \vdash \mathtt{inl}\ \mathtt{g} : \nu}}{\mathtt{x}, \mathtt{g} \vert D, \forall \mathtt{y}.(\mathrm{F} \Rightarrow \mathrm{F}) \vdash \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}}{\mathtt{x} \vert D \vdash \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}$$

Let θ_3 be the following proof:

$$\dfrac{\dfrac{}{\mathtt{x} \vert \vdash (D \Rightarrow \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})) \Rightarrow \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})} \quad \dfrac{\dfrac{\theta_2}{\mathtt{x} \vert D \vdash \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}}{\mathtt{x} \vert \vdash D \Rightarrow \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}}{\mathtt{x} \vert \vdash \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}$$

where the left premise is an instance of Peirce's Law.
Let θ be the following proof:

$$\dfrac{\dfrac{\dfrac{}{\vert \vdash (\forall \mathtt{x}.\exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})) \Rightarrow (\exists \mathtt{f}.\forall \mathtt{x}.(\mathrm{F} \Rightarrow \mathrm{F}))} \quad \dfrac{\dfrac{\theta_3}{\mathtt{x} \vert \vdash \exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}}{\vert \vdash \forall \mathtt{x}.\exists \mathtt{u}.(\mathrm{F} \Rightarrow \mathrm{F})}}{\vert \vdash \exists \mathtt{f}.\forall \mathtt{x}.(\mathrm{F} \Rightarrow \mathrm{F})} \quad \dfrac{\pi}{\mathtt{f} \vert \forall \mathtt{x}.(\mathrm{F} \Rightarrow \mathrm{F}) \vdash (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}}}{\vert \vdash (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}}$$

where the left premise is an instance of AC_σ.

Fig. 11. Construction of θ, whose proofterm is s'

Let ϕ be the following proof:

$$\dfrac{\dfrac{\theta}{| \vdash (\mathrm{F} \Rightarrow \mathrm{F}) \vee \mathrm{F}} \qquad \dfrac{\dfrac{}{|\mathrm{F} \Rightarrow \mathrm{F}, \mathrm{F} \vdash \mathrm{F}}}{|\mathrm{F} \Rightarrow \mathrm{F} \vdash \mathrm{F} \Rightarrow \mathrm{F}} \qquad \dfrac{\dfrac{}{|\mathrm{F}, \mathrm{F} \vdash \mathrm{F}}}{|\mathrm{F} \vdash \mathrm{F} \Rightarrow \mathrm{F}}}{| \vdash \mathrm{F} \Rightarrow \mathrm{F}}$$

where the sequent $|\mathrm{F}, \mathrm{F} \vdash \mathrm{F}$ is proved from its second assumption.

Fig. 12. Construction of ϕ, whose proofterm gets stuck

4. M. Felleisen and D. P. Friedman. Control operators, the SECD-machine, and the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts*, pages 193–217. North-Holland, 1986.
5. T. G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages*. ACM Press, 1990.
6. W. A. Howard. The formulae-as-types notion of constructions. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays on Computational Logic, Lambda Calculus and Formalism.*, pages 479–490. Academic Press, 1980.
7. J.-L. Krivine. Dependent choice, "quote" and the clock. to appear in Theoretical Computer Science.
8. P. B. Levy. *Call-by-push-value*. PhD thesis, Queen Mary, University of London, 2001. Revised version to appear in Semantic Structures of Computation, Klüwer.
9. P. Martin-Löf. *Intuitionistic type theory*. Bibliopolis, Napoli, 1980.
10. M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Proc. of the International Conference on Logic Programming and Automated Reasoning*, volume 624 of *LNAI*. Springer, 1992.
11. C. Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In F. D. E. Dekker, editor, *Recursive function theory: Proc. symposia in pure mathematics*, pages 1–27. American Mathematical Society, Providence, Rhode Island, 1962.
12. W. W. Tait. Normal derivability in classical logic. In J. Barwise, editor, *The Syntax and Semantics of Infinitary Languages*, volume 72 of *LNM*. Springer, 1968.
13. H. Thielecke. Using a continuation twice and its implications for the expressive power of `call/cc`. *Higher-Order and Symbolic Computation*, 12(1):47–74, 1999.