The $Y$-hierarchy for PCF is strict
(Draft version)

John Longley

May 19, 2015

Abstract

Let PCF$_k$ denote the sublanguage of Plotkin’s PCF in which fixed point operators $Y_\sigma$ are admitted only for types $\sigma$ of level at most $k$. We show that the languages PCF$_k$ form a strict hierarchy as regards expressivity, in the sense that for each $k$, there are closed programs of PCF$_{k+1}$ that are not observationally equivalent to any programs of PCF$_k$. This answers positively a question posed by Berger in 1999. Our proof makes serious use of the theory of nested sequential procedures (also called PCF Böhm trees) as expounded in the forthcoming book of Longley and Normann.

1 Introduction

In this paper we study sublanguages of Plotkin’s functional programming language PCF, which we here take to be the simply typed $\lambda$-calculus over a single base type $\mathbb{N}$, with constants

$$
\begin{align*}
\widehat{n} & : \mathbb{N} \text{ for each } n \in \mathbb{N} , \\
\text{suc, pre} & : \mathbb{N} \rightarrow \mathbb{N} , \\
\text{ifzero} & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} , \\
Y_\sigma & : (\sigma \rightarrow \sigma) \rightarrow \sigma \text{ for each type } \sigma .
\end{align*}
$$

As usual, we consider this language to be endowed with a certain (call-by-name) operational semantics, which in turn gives rise to a notion of observational equivalence for PCF programs.

We define the level $\text{lv}(\sigma)$ of a type $\sigma$ is defined inductively by

$$
\text{lv}(\mathbb{N}) = 0 , \quad \text{lv}(\sigma \rightarrow \tau) = \max(\text{lv}(\sigma) + 1, \text{lv}(\tau)) ,
$$

and define the pure type $\overline{k}$ of level $k \in \mathbb{N}$ by

$$
\begin{align*}
\overline{0} & = \mathbb{N} , \\
\overline{k+1} & = \overline{k} \rightarrow \mathbb{N} .
\end{align*}
$$

Modifying the definition of PCF so that the constants $Y_\sigma$ are admitted only for types $\sigma$ of level $\leq k$, we obtain a sublanguage PCF$_k$ for each $k \in \mathbb{N}$. Our main result will be that for each $k$, the expressive power of PCF$_{k+1}$ strictly exceeds
that of PCF$_k$: specifically, there is no closed term of PCF$_k$ that is observation-
ally equivalent to the term $Y_{n \rightarrow n+1}$ of PCF$_{k+1}$. (Note that for ascertain-
ing the observational equivalence or otherwise of closed terms, it suffices to restrict at-
tention to observing contexts of the form $-N_0 \ldots N_{r-1}$ where the $N_i$ are closed
terms of PCF$_0$, as will be explained in Section 2.) This answers a question that
was posed explicitly by Berger in [1], but which has been present in the folklore
at least since the early 1990s. Our theorem can also be understood as saying
that in a suitably pure fragment of a programming language such as Haskell,
the computational power of recursive function definitions increases strictly with
their type level.

Our discussion will focus on two models of PCF, both studied extensively
in the forthcoming book of Longley and Normann [3]: the nested sequential
procedure model SP$_0$ (also known as the PCF Böhm tree model and by various
other names), and its extensional quotient SF consisting of sequential function-
als. It is well-known that SF is fully abstract for PCF, and indeed that SF is
isomorphic to the closed term model of PCF$_{\Omega}$ modulo observational equivalence
(here PCF$_{\Omega}$ denotes the extension of PCF with an ‘oracle’ $C_f$ for every partial
function $f : \mathbb{N} \rightarrow \mathbb{N}$). Our result can therefore be expressed more denotationally
by saying that none of the languages PCF$_{\Omega}$ suffice for denoting all the elements
of SF. (It follows readily that the same holds if SF is replaced by any other
adequate, compositional model of PCF.) We may also easily deduce that there
is no finite ‘basis’ $B \subseteq SF$ such that all elements of SF are $\lambda$-definable relative
to $B$.

In Section 2 we recall the necessary technical background on PCF and on
the models SP$_0$ and SF, fleshing out the ideas outlined informally above. In
Section 3 we define (for arbitrary $k \in \mathbb{N}$) a substructure $A_k \subseteq SP_0$, and show
that every closed term of PCF$_k$ has a denotation in $A_k$. In Section 4 we show
that this substructure excludes the standard interpretation of $Y_{k+1}$ in SP$_0$. This
suffices to establish a weak version of our result, namely that within the model
SP$_0$, the element $Y_{k+1}$ is not definable in PCF$_k$. However, this still leaves
open the possibility that there might be other NSPs, distinct from $Y_{k+1}$ but
extensionally equivalent to it, that are denotable in PCF$_k$. We will show in
Section 5 that this is not the case, so that the non-definability holds also for SF
as claimed.

2 Background

We here summarize the necessary definitions and technical background from [3],
especially from Chapters 6 and 7.

2.1 The language PCF

We first recall the definition of Plotkin’s language PCF [5]. We will work with
the same version of PCF as was used in [3], with the natural numbers as the
only base type. Our types $\sigma$ are generated by

$$\sigma ::= N \mid \sigma \to \sigma,$$

and our terms will be those of the simply typed $\lambda$-calculus constructed from the constants

\begin{align*}
\hat{n} & : N \quad \text{for each } n \in \mathbb{N}, \\
suc, \ pre & : N \to N, \\
\text{ifzero} & : N, N, N \to N, \\
Y_\sigma & : (\sigma \to \sigma) \to \sigma \quad \text{for each type } \sigma.
\end{align*}

As usual, we write $\Gamma \vdash M : \sigma$ to mean that $M$ is a well-typed term in the environment $\Gamma$ (where $\Gamma$ is a finite list of typed variables). For each $k \in \mathbb{N}$, the sublanguage PCF$_k$ is obtained by admitting the constants $Y_\sigma$ only for types $\sigma$ of level $\leq k$.

We endow the class of closed PCF terms with the following small-step reduction rules

\begin{align*}
(\lambda x.M)N & \rightsquigarrow M[x \mapsto N], \\
\text{ifzero } \hat{0} & \rightsquigarrow \lambda xy.x, \\
suc \hat{n} & \rightsquigarrow n + 1, \\
\text{ifzero } n + 1 & \rightsquigarrow \lambda xy.y, \\
pre n + 1 & \rightsquigarrow \hat{n}, \\
Y_\sigma M & \rightsquigarrow Mf(Y_\sigma M), \\
pren + 1 & \rightsquigarrow \hat{n}, \\
pren & \rightsquigarrow \hat{0}.
\end{align*}

These reductions may be applied in any evaluation context, where the set of evaluation contexts is generated by composition from the basic contexts

$$[-]N, \quad suc [-], \quad pre [-], \quad \text{ifzero } [-].$$

(For example, we have $\text{ifzero } (\text{suc } \hat{1}) \hat{3} \hat{5} \rightsquigarrow \text{ifzero } \hat{2} \hat{3} \hat{5}$.) We write $\rightsquigarrow^*$ for the reflexive-transitive closure of $\rightsquigarrow$. If $Q$ is any closed PCF term of type $N$, it is easy to see that either $Q \rightsquigarrow^* \hat{n}$ for some $n \in \mathbb{N}$, or the reduction sequence starting from $Q$ is infinite.

This completes the definition of the languages PCF and PCF$_k$. Whilst the language PCF$_0$ is too weak for practical programming purposes (it cannot even define addition), it is not hard to show that even PCF$_1$ is Turing-complete: that is, any partial computable function $\mathbb{N} \rightarrow \mathbb{N}$ is representable (in the obvious sense) by a closed PCF$_1$ term of type $N \to N$.

We will also refer to the (non-effective) language PCF$^\Omega$, or oracle PCF, obtained by extending the definition of PCF with a constant $C_f : N \rightarrow N$ for every set-theoretic partial function $f : \mathbb{N} \rightarrow \mathbb{N}$, along with a reduction rule $C_f \hat{n} \rightsquigarrow \hat{m}$ for every $n, m$ such that $f(n) = m$. (In PCF$^\Omega$, the evaluation of a closed term $Q : N \rightarrow N$ may fail to reach a value $\hat{n}$ either because it generates an infinite computation, or because it encounters some subterm $C_f(n)$ where $f(n)$ is undefined.) The languages PCF$^\Omega_k$ are now defined in the obvious way.
If $M, M'$ are closed PCF$^\Omega$ terms of the same type $\sigma$, and $L : \sigma$ is one of our languages PCF$k$, PCF$k^\Omega$, PCF, PCF$^\Omega$, we may say that $M, M'$ are observationally equivalent in $L$, and write $M \simeq_L M'$, if for all closed program contexts $C[-]\ : \ N$ of $L$ and all $n \in \mathbb{N}$, we have

$$C[M] \leadsto^n \iff C[M'] \leadsto^n \ .$$

Fortunately, it is easy to see that all of the above languages give rise to exactly the same relation $\simeq_L$. First, it is immediate from the definition that if $L, L'$ are two of our languages and $L \supseteq L'$, then $\simeq_L \subseteq \simeq_{L'}$. It therefore only remains to show that $M \simeq_{PCF_0} M'$ implies $M \simeq_{PCF^\Omega} M'$. For this, we use the fact that any of the constants $Y$, or $C_f$ in PCF$^\Omega$ can be ‘approximated’ to any desired accuracy by terms of PCF$_0$. Indeed, for any $j \in \mathbb{N}$, we may define PCF$_0$ terms

$$Y_s^{(j)} = \lambda f. f^j(\bot),$$

$$C_f^{(j)} = \lambda n. \text{case } n \text{ of } (0 \Rightarrow f(0)) \cdots | j - 1 \Rightarrow f(j-1)) \ ,$$

writing $\bot$ for $Y_0(\lambda x^0.x)$, and using some evident syntactic sugar in the definition of $C_f^{(j)}$. For any PCF$^\Omega$ term $M$, let $M^{(j)}$ denote the ‘approximation’ obtained from $M$ by replacing all occurrences of constants $Y_s, C_f$ by $Y_s^{(j)}, C_f^{(j)}$ respectively. It is then not hard to show that for closed $Q : \mathbb{N}$, we have

$$Q \leadsto^* \hat{n} \iff \exists j, Q^{(j)} \leadsto^* \hat{n} \ .$$

From this it follows easily that if $C[-]$ is an observing context of PCF$^\Omega$ that distinguishes $M, M'$, then some approximation $C^{(j)}[-]$ (a context of PCF$_0$) also suffices to distinguish them. This establishes that $\leadsto^{PCF_0} \subseteq \leadsto_{PCF^\Omega}$.

In fact, an even more restricted class of observing contexts suffices for ascertaining observational equivalence of PCF$^\Omega$ terms. The well-known context lemma, due to Milner [4], states that $M \simeq_{obs} M' : \sigma_0 \rightarrow \cdots \sigma_{r-1} \rightarrow \mathbb{N}$ iff $M, M'$ have the same behaviour in all applicative contexts of PCF—that is, if for all closed PCF terms $N_0 : \sigma_0, \ldots, N_{r-1} : \sigma_{r-1}$, we have

$$MN_0 \ldots N_{r-1} \leadsto^* n \iff M'N_0 \ldots N_{r-1} \leadsto^* n \ .$$

Furthermore, using the above ideas of approximation, it is easy to see that we obtain exactly the same equivalence relation if we allow the $N_i$ here to range only over closed PCF$0$ terms—this gives us the notion of PCF$0$ applicative equivalence, which we shall denote by $\sim_0$.

We have concentrated so far on giving a purely operational description of PCF. We are now able to express the operational content of our main theorem as follows. As in Section 1, we define the type $\kappa$ by $\overline{0} = \mathbb{N}, \overline{k+1} = \overline{k} \rightarrow \mathbb{N}$; we shall write $\overline{k}$ simply as $k$ where there is no risk of confusion.

**Theorem 1** For any $k \geq 1$, there are functionals definable in PCF$^{k+1}$ but not in PCF$k$. More precisely, there is no closed term $M$ of PCF$^\Omega_k$ such that $M \simeq_{obs} Y_{0 \rightarrow (k+1)}$ (or equivalently $M \sim_0 Y_{0 \rightarrow (k+1)}$).\(^1\)

\(^1\)We conjecture that even $Y_{k+1}$ cannot be defined in PCF$^\Omega_k$ modulo observational equivalence. For our best result to date on $Y_{k+1}$, see Theorem 5 below.
Likewise, our theorem can be construed as saying that in a suitably pure fragment of a functional language such as Haskell, the computational strength of recursive function definitions increases strictly as the admissible type level for such recursions is increased. The formulation in terms of \( \sim_0 \) has the virtue that it manifests the robustness of our result: if \( M \in \text{PCF}_k \) were ‘equivalent’ to \( Y_{0 \rightarrow (k+1)} \) in any reasonable sense, we would surely expect them to induce the same partial function on closed PCF_0 terms.

A slightly more denotational formulation of the above ideas can be given in terms of the model \( SF \) of sequential functionals, which we may here define as the type structure of closed PCF\( ^\Omega \) terms modulo observational equivalence. Specifically, for each type \( \sigma \), let \( SF(\sigma) \) denote the set of closed PCF\( ^\Omega \) terms \( M : \sigma \) modulo \( \sim \); Clearly, application of PCF\( ^\Omega \) terms induces a well-defined function \( \cdots : SF(\sigma \rightarrow \tau) \times SF(\sigma) \rightarrow SF(\tau) \) for any \( \sigma, \tau \). The structure \( SF \) then consists of the sets \( SF(\sigma) \) along with these application operations. Using the context lemma, it is easy to show that \( SF(\mathsf{Nat}) \cong \mathbb{N} \uplus \{ \bot \} \), and that \( SF(\sigma \rightarrow \tau) \) is isomorphic to a set of functions \( SF(\sigma) \rightarrow SF(\tau) \): that is, if \( f, f' \in SF(\sigma \rightarrow \tau) \) and \( f \cdot x = f' \cdot x \) for all \( x \in SF(\sigma) \), then \( f = f' \).

Any closed PCF\( ^\Omega \) term \( M : \sigma \) naturally has a denotation \( [M] \) in \( SF(\sigma) \), namely its own equivalence class. We may therefore restate Theorem 1 as follows:

**Theorem 2** For any \( k \geq 1 \), the element \([Y_{0 \rightarrow (k+1)}]\) in \( SF \) is not PCF\( ^\Omega_k \)-definable.

It will follow immediately that in any other adequate, compositional model of PCF\( ^\Omega \) (such as Scott’s continuous model or Berry’s stable model), the element \([Y_{0 \rightarrow (k+1)}]\) is not PCF\( ^\Omega_k \)-definable, since the equality on PCF\( ^\Omega \) terms induces by such a model must be contained within \( \sim \).

By taking closed terms of PCF rather than PCF\( ^\Omega \) modulo observational equivalence, we obtain the type structure \( SF^{\text{eff}} \) of effective sequential functionals, which can clearly be seen as a substructure of \( SF \). Although this construction of \( SF^{\text{eff}} \) proceeds via the syntax of PCF, there are many other more mathematical constructions that also give rise to this structure, and experience shows it to be a mathematically natural class of higher-order functionals (as indeed is \( SF \) itself). We may now see that Theorem 2 implies an interesting absolute property of this type structure, not dependent on any choice of presentation for this structure or selection of language primitives:

**Corollary 3 (No finite basis)** There is no finite set \( B \) of elements of \( SF^{\text{eff}} \) such that all elements of \( SF^{\text{eff}} \) are \( \lambda \)-definable relative to \( B \).

**Proof** Suppose \( B = \{ b_0, \ldots, b_{n-1} \} \) were such a set. For each \( i \), take a closed PCF term \( M_i \) denoting \( b_i \). Then the terms \( M_0, \ldots, M_{n-1} \) between them contain only finitely many occurrences of constants \( Y_\sigma \), so for a sufficiently large \( k \), these are all such that \( \ell v(\sigma) \leq k \). But this means that \( M_0, \ldots, M_{n-1} \) are all terms of PCF\( _k \). So \( b_0, \ldots, b_{n-1} \), and hence all elements of \( SF^{\text{eff}} \), are PCF\( _k \)-definable, contradicting Theorem 2. \( \square \)
2.2 The model $SP^0$

We turn next to an overview of the construction of the nested sequential procedure (or NSP) model, denoted by $SP^0$. We give here only as much as is needed for the reading of the present paper, referring the reader to [3, Chapter 6] for a more thorough treatment, as well as a discussion of the rather complex history of the ideas.

We shall of course be interested primarily in NSPs of the simple types as defined above. However, it is convenient to work within a slightly more generous framework allowing for function types with infinitely many arguments, as such types will occasionally play a useful auxiliary role. We therefore consider the class of extended types $\sigma$ as defined by the grammar

$$\sigma ::= \sigma_0, \sigma_1, \ldots \to N$$

where $\sigma_0, \sigma_1, \ldots$ may be either a finite or infinite list of types (possibly the empty list). We write the extended type $\to N$ simply as $N$, and in general define a binary operator $\to$ on the class of extended types by

$$\sigma \to (\sigma_0, \sigma_1, \ldots \to N) = \sigma, \sigma_0, \sigma_1, \ldots \to N.$$  

This convention enables us to regard any simple type $\sigma$ as an extended type; we will not distinguish notationally between a simple type and its extended counterpart.

We define the level of an extended type inductively by $lv(\sigma_0, \sigma_1, \ldots \to N) = 1 + \max_i lv(\sigma_i)$, where $lv(N) = 0$. Thus, for example, the evident type $N^\omega \to N$ is considered for our present purposes as a type of level 1.

Much as in [3], our NSPs are generated by means of the following infinitary grammar, interpreted coinductively:

Procedures: $p, q ::= \lambda x_0x_1 \cdots . e$

Expressions: $d, e ::= \bot \mid n \mid \text{case } a \text{ of } (i \Rightarrow e_i \mid i \in \mathbb{N})$

Applications: $a ::= x q_0q_1 \cdots$

Here $x_0x_1 \cdots$ may be either a finite or infinite list of variables (possibly empty), and $q_0q_1 \cdots$ either a finite or infinite list of procedures (again possibly empty). We will use vector notation to denote such lists which may be finite or infinite: $\vec{x}, \vec{q}, \vec{\sigma}$. We may use $t$ to range over NSP terms of any of the above three kinds. We shall always work with terms up to (infinitary) $\alpha$-equivalence.

If each variable is assigned an (extended) type, then we may restrict our attention to well-typed terms. Informally, a term will be well-typed unless a typing violation occurs at some specific point within its syntax tree. The typing rules will mostly play only a background role in the present paper, but for the sake of completeness we give them here. Specifically, a term $t$ is well-typed if for every application $xq$ appearing within $t$, the type of $x$ has the form $\vec{\sigma} \to N$ where $\vec{\sigma}$ and $\vec{q}$ are of the same length (finite or infinite), and for each $i$, the procedure $q_i$ has the form $\lambda \vec{x}_i . e_i$, where the variables $\vec{x}_i$ have types $\vec{\tau}_i$ and
\[ \sigma_i = \vec{\tau}_i \to \mathbb{N} \]. If \( \Gamma \) is any environment (i.e. set of variables), we write \( \Gamma \vdash e \) and \( \Gamma \vdash a \) to mean that \( e, a \) respectively are well-typed with free variables in \( \Gamma \); we also write \( \Gamma \vdash p : \tau \) when \( p \) is well-typed in \( \Gamma \) and of the form \( \lambda \vec{x}.e \), where the variables \( \vec{x} \) have types \( \vec{\sigma} \) and \( \tau = \vec{\sigma} \to \mathbb{N} \). We emphasize that environments \( \Gamma \) may be infinite, and that terms may contain infinitely many distinct variables.

With these in place, we may take \( SP^0(\sigma) \) to be the set of closed and well-typed procedures of (extended) type \( \sigma \).

The above definition of NSPs is more general than the typed notion of NSPs as considered in [3], insofar as we are working with extended types. However, our present class of NSPs is smoothly accommodated with the class of \emph{untyped} NSPs as discussed in [3, Section 6.1.7]. As explained there, all the main aspects of the theory of NSPs work equally well in the untyped setting, so in particular are applicable to the extended notion of NSPs that we consider here.

As in [3], we shall need to work not only with NSPs themselves, but with a more general calculus of NSP \emph{meta-terms} designed to accommodate the intermediate forms that arise in the course of evaluation:

\begin{align*}
\text{Meta-procedures:~} P, Q, R &::= \lambda \vec{x}.E \\
\text{Meta-expressions:~} D, E &::= \bot | n | \text{case } G \text{ of } (i \Rightarrow E_i \mid i \in \mathbb{N}) \\
\text{Ground meta-terms:~} G &::= E \mid x \vec{Q} \mid P \vec{Q}
\end{align*}

Here again, \( \vec{x} \) and \( \vec{Q} \) may denote finite or infinite lists. We shall use \( T \) to range over meta-terms of any of the above three kinds. (Unless otherwise stated, we use uppercase letters for general meta-terms and lowercase ones for terms.)

Again, there are some evident typing rules for meta-terms, leading to typing judgements \( \Gamma \vdash P : \sigma, \Gamma \vdash E, \Gamma \vdash D \) for meta-procedures, meta-expressions and ground meta-terms respectively. These typing rules are the obvious adaptation of those above for terms (cf. [3, Section 6.1.1]); again, they will most play only a background role in this present paper. We also have an evident notion of (simultaneous) substitution \( T[\vec{x} \mapsto \vec{Q}] \) for well-typed terms.

There is also a concept of \emph{evaluation} whereby any meta-term \( \Gamma \vdash T (: \sigma) \) evaluates to an ordinary term \( \Gamma \vdash \ll T \gg (: \sigma) \). To define this, the first stage is to introduce a \emph{basic reduction} relation \( \sim_b \) for ground meta-terms, which we do by the following rules:

\begin{enumerate}
\item[(b1)] \( (\lambda \vec{x}.E)\vec{Q} \sim_b E[\vec{x} \mapsto \vec{Q}] \) \hspace{1em} (\beta\text{-rule}).
\item[(b2)] \text{case } \bot \text{ of } (i \Rightarrow E_i) \sim_b \bot.
\item[(b3)] \text{case } n \text{ of } (i \Rightarrow E_i) \sim_b E_n.
\item[(b4)] \text{case } \text{case } G \text{ of } (i \Rightarrow E_i) \text{ of } (j \Rightarrow F_j) \sim_b \\
\text{case } G \text{ of } (i \Rightarrow \text{case } E_i \text{ of } (j \Rightarrow F_j)).
\end{enumerate}

Note that the \( \beta \)-rule applies even when \( \vec{x} \) is empty: thus \( \lambda.2 \sim_b 2 \).

From this, a \emph{head reduction} relation \( \sim_h \) on meta-terms is defined inductively:

\begin{enumerate}
\item[(h1)] If \( G \sim_h G' \) then \( G \sim_h G' \).
\end{enumerate}
(h2) If \( G \leadsto_h G' \) and \( G \) is not a case meta-term, then
\[
\text{case } G \text{ of } (i \Rightarrow E_i) \leadsto_h \text{ case } G' \text{ of } (i \Rightarrow E_i).
\]

(h3) If \( E \leadsto_h E' \) then \( \lambda \vec{x}. E \leadsto_h \lambda \vec{x}. E' \).

Clearly, for any meta-term \( T \), there is at most one \( T' \) with \( T \leadsto_h T' \). We call a meta-term a head normal form if it cannot be further reduced using \( \leadsto_h \).

The possible shapes of head normal forms are \( \perp \), \( n \), case \( y\vec{Q} \text{ of } (i \Rightarrow E_i) \) and \( y\vec{Q} \), the first three optionally prefixed by \( \lambda \vec{x} \) (where \( \vec{x} \) may contain \( y \)).

We now define the general reduction relation \( \leadsto \) inductively as follows:

(g1) If \( T \leadsto_h T' \) then \( T \leadsto T' \).

(g2) If \( E \leadsto E' \) then \( \lambda \vec{x}. E \leadsto \lambda \vec{x}. E' \).

(g3) If \( Q_j = Q'_j \) except at \( j = k \) where \( Q_k \leadsto Q'_k \), then
\[
\begin{align*}
x\vec{Q} & \leadsto x\vec{Q}', \\
\text{case } x\vec{Q} \text{ of } (i \Rightarrow E_i) & \leadsto \text{ case } x\vec{Q}' \text{ of } (i \Rightarrow E_i).
\end{align*}
\]

(g4) If \( E_i = E'_i \) except at \( i = k \) where \( E_k \leadsto E'_k \), then
\[
\text{case } x\vec{Q} \text{ of } (i \Rightarrow E_i) \leadsto \text{ case } x\vec{Q} \text{ of } (i \Rightarrow E'_i).
\]

It is easy to check that this reduction system is sound with respect to the typing rules.

An important point to note is that the terms \( t \) are precisely the meta-terms in normal form, i.e. those that cannot be reduced using \( \leadsto \). We write \( \leadsto^* \) for the reflexive-transitive closure of \( \leadsto \).

The above reduction system captures the finitary aspects of evaluation. In general, however, since terms and meta-terms may be infinitely deep, evaluation must be seen as an infinite process. To account for this infinitary aspect, we use some familiar domain-theoretic ideas.

We write \( \sqsubseteq \) for the evident syntactic orderings on meta-procedures and on ground meta-terms: thus, \( T \sqsubseteq U \) iff \( T \) may be obtained from \( U \) by replacing zero or more subterms (possibly infinitely many) by \( \perp \). It is easy to see that each set \( \text{SP}(\sigma) \) or \( \text{SP}'(\sigma) \) forms a directed-complete partial order under \( \sqsubseteq \), with least element \( \lambda \vec{x}. \perp \).

By a finite term \( t \) we shall mean one generated by the following grammar, this time construed inductively:

\[
\begin{align*}
\text{Procedures:} & \quad p, q ::= \lambda x_0 \ldots x_{r-1}. e \\
\text{Expressions:} & \quad d, e ::= \perp \mid n \mid \text{case } a \text{ of } (0 \Rightarrow e_0 \mid \cdots \mid r - 1 \Rightarrow e_{r-1}) \\
\text{Applications:} & \quad a ::= xq_0 \ldots q_{r-1}
\end{align*}
\]
We regard finite terms as a subset of general terms by identifying the conditional branching \((0 \Rightarrow e_0 \mid \cdots \mid r - 1 \Rightarrow e_{r-1})\) with
\[
(0 \Rightarrow e_0 \mid \cdots \mid r - 1 \Rightarrow e_{r-1} \mid r \Rightarrow \bot \mid r + 1 \Rightarrow \bot \mid \ldots).
\]

We may now explain how a general meta-term \(T\) evaluates to a term \(\llangle T \rrangle\). This will in general be an infinite process, but we can capture the value of \(T\) as the limit of the finite portions that become visible at finite stages in the reduction. To this end, for any meta-term \(T\) we define
\[
\downarrow_{\text{fin}} T = \{ t \text{ finite} \mid \exists T'. T \Rightarrow^* T' \land t \subseteq T' \}.
\]

It is fairly easy to check that for any meta-term \(T\), the set \(\downarrow_{\text{fin}} T\) is directed with respect to \(\subseteq\), and also that each \(\text{SP}(\sigma)\) is directed-complete. We may therefore define \(\llangle T \rrangle\), the value of \(T\), to be the ordinary term
\[
\llangle T \rrangle = \bigsqcup (\downarrow_{\text{fin}} T).
\]

Note in passing that the value \(\llangle G \rrangle\) of a ground meta-term \(G\) may be either an expression or an application. In either case, it is certainly a ground meta-term. It is also easy to see that \(\llangle \lambda \vec{x}. E \rrangle = \lambda \vec{x}. \llangle E \rrangle\), and that if \(T \Rightarrow^* T'\) then \(\llangle T \rrangle = \llangle T' \rrangle\).

In the present paper, an important role will be played by the tracking of variable occurrences (and sometimes other subterms) through the course of evaluation. By inspection of the above rules for \(\Rightarrow\), it is easy to see that if \(T \Rightarrow T'\), then for any occurrence of a (free or bound) variable \(x\) within \(T'\), we can identify a unique occurrence of \(x\) within \(T\) from which it originates (we suppress the formal definition). The same therefore applies whenever \(T \Rightarrow^* T'\). In this situation, we may say that the occurrence within \(T'\) is a residual of the one within \(T\), or that the latter is the origin of the former. Note, however, that these relations are relative to a particular reduction path \(T \Rightarrow^* T'\): there may be other paths for which the origin-residual relation is different.

Likewise, for any occurrence of \(x\) within \(\llangle T \rrangle\), we may pick some finite \(t \subseteq \llangle T \rrangle\) containing this occurrence, and some \(T' \supseteq t\) with \(T \Rightarrow^* T'\); this allows us to identify a unique occurrence of \(x\) within \(T\) that originates the given occurrence in \(\llangle T \rrangle\). It is routine to check that this occurrence in \(T\) will be independent of the choice of \(t\) and \(T'\) and of the chosen reduction path \(T \Rightarrow^* T'\); we therefore have a robust origin-residual relationship between variable occurrences in \(T\) and those in \(\llangle T \rrangle\).

A fundamental result for NSPs is the evaluation theorem, which says broadly that the result of evaluating a meta-term is independent of the order of evaluating sub-expressions:

**Theorem 4 (Evaluation theorem)** If \(C[-^0, -^1, \ldots]\) is any meta-term context with countably many holes and \(C[T^0, T^1, \ldots]\) is well-formed, then
\[
\llangle C[T^0, T^1, \ldots] \rrangle = \llangle C[\llangle T^0 \rrangle, \llangle T^1 \rrangle, \ldots] \rrangle.
\]
The proof of this is non-trivial; see [3, Section 6.1.2]. As discussed in Section 6.1.5 of the book, the same proof is also applicable to untyped NSP meta-terms, and so certainly applies to the extended typed system that we are considering here.

Although not explicitly mentioned in [3], an inspection of the proof of Theorem 4 readily confirms that the notion of residual is again robust with respect to such variations in order of evaluation. Specifically, suppose \( \xi' \) denotes an occurrence of a variable \( x \) within \( \ll C[\ll T \gg] \), and \( \xi \) denotes its origin within \( C[\ll T \gg] \). Then

- if \( \xi \) lies within \( C[-] \), the corresponding occurrence within \( C[\ll \ll T \gg] \) is the origin of \( \xi' \) within \( \ll C[\ll \ll T \gg] \gg \);
- if \( \xi \) lies within some \( T_i \), there is an occurrence \( \xi'' \) of \( x \) within \( \ll T_i \gg \) such that \( \xi \) within \( T_i \) is the origin of \( \xi'' \) within \( \ll T_i \gg \), and \( \xi'' \) within \( C[\ll \ll T \gg] \) is the origin of \( \xi' \) within \( \ll C[\ll \ll T \gg] \gg \).

This will allow us to use the notions of origin and residual with confidence in complex situations involving many possible routes to evaluation.

One further piece of machinery will be useful: the notion of hereditary \( \eta \)-expansion, which enables us to convert a variable \( x \) into a procedure term (written \( x^\eta \)). The definition is by recursion on the type of \( x \): if \( x : \sigma_0, \sigma_1, \ldots \rightarrow \mathbb{N} \), then

\[
x^\eta = \lambda z_{\sigma_0}^0 \lambda z_{\sigma_1}^1 \cdots \text{case } xz_{\sigma_r}^r \text{ of } (i \Rightarrow i) .
\]

In particular, if \( x : \mathbb{N} \) then \( x^\eta = \lambda . \text{case } x \text{ of } (i \Rightarrow i) \). The following useful properties of \( \eta \)-expansion are proved in [3, Section 6.1.3] (we assume the terms in question here are well typed):

\[
\ll x^\eta \overline{q} \gg = \text{case } x\overline{q} \text{ of } (i \Rightarrow i) ,
\]

\[
\ll \lambda \overline{y} . p \overline{y}^\eta \gg = p .
\]

The sets \( \text{SP}(\sigma) \) may be made into a total applicative structure \( \text{SP} \) by defining

\[
(\lambda x_0 \cdots x_r . e) \cdot q = \lambda x_1 \cdots x_r . \ll e[x_0 \mapsto q] \gg .
\]

Clearly the sets \( \text{SP}^0(\sigma) \) are closed under this application operation, so we also obtain an applicative substructure \( \text{SP}^0 \). It is easy to check that application in \( \text{SP} \) is monotone and continuous with respect to \( \sqsubseteq \). It is also shown in [3, Section 6.1.3] that both \( \text{SP} \) and \( \text{SP}^0 \) are typed \( \lambda \)-algebras: that is, they admit a compositional interpretation of typed \( \lambda \)-terms that validates \( \beta \)-equality. (The relevant interpretation of pure \( \lambda \)-terms is in fact defined by three of the clauses from the interpretation of PCF\(^\Omega \) as defined below.)

2.3 Interpretation of PCF in \( \text{SP}^0 \)

A central role will be played by certain procedures \( Y_\sigma \in \text{SP}^0((\sigma \rightarrow \sigma) \rightarrow \sigma) \) which we use to interpret the PCF constants \( Y_\sigma \) (the overloading of notation
will do no harm in practice). If \( \sigma = \sigma_0 \to \cdots \to \sigma_{r-1} \to \mathbb{N} \), we define \( Y_\sigma = \lambda g^{\sigma \to \sigma}.F_\sigma[g] \), where \( F_\sigma[g] \) is defined corecursively by:
\[
F_\sigma[g] = \lambda x_{\sigma_0} \cdots x_{\sigma_{r-1}}. \text{case } g (F_\sigma[g]) x_{\sigma_0} \cdots x_{\sigma_{r-1}} \text{ of } (i \Rightarrow i)
\]

We may now give the standard interpretation of PCF in \( \text{SP}^0 \). To each PCF term \( \Gamma \vdash M : \sigma \) we associate a procedure-in-context \( \Gamma \vdash [M]_\Gamma : \sigma \) inductively as follows:
\[
\begin{align*}
[x^\sigma]_\Gamma &= x^{\sigma} \\
[n]_\Gamma &= \lambda.n \\
[suc]_\Gamma &= \lambda x. \text{case } x \text{ of } (i \Rightarrow i + 1) \\
[pre]_\Gamma &= \lambda x. \text{case } x \text{ of } (0 \Rightarrow 0 | i + 1 \Rightarrow i) \\
[ifzero]_\Gamma &= \lambda x y z. \text{case } x \text{ of } (0 \Rightarrow \text{case } y \text{ of } (j \Rightarrow j) | i + 1 \Rightarrow \text{case } z \text{ of } (j \Rightarrow j)) \\
[Y_\sigma]_\Gamma &= Y_\sigma \\
[C_f]_\Gamma &= \lambda x. \text{case } x \text{ of } (i \Rightarrow f(i)) \\
[\lambda x^\sigma. M]_\Gamma &= \lambda x^{\sigma}. [M]_\Gamma \cdot x^{\sigma} \\
[M N]_\Gamma &= [M]_\Gamma \cdot [N]_\Gamma
\end{align*}
\]

(In the clause for \( C_f \), we interpret \( f(i) \) as \( \bot \) whenever \( f(i) \) is undefined.)

As is shown in [3], this interpretation is \textit{adequate}, in the sense that \( M \cong^* \hat{n} \) iff \( [M] = \lambda.n \), and \textit{universal}, in the sense that every element of \( \text{SP}^0(\sigma) \) is the denotation of some closed \( M : \sigma \) in PCF. It follows from these facts that the structure SF is a quotient of \( \text{SP}^0 \), and it indeed its \textit{extensional collapse}; we shall write \( \cong \) for the equivalence relation on \( \text{SP}^0 \) induced by the quotient map. It is also routine to check that the canonical interpretation of PCF in SF factors through the above interpretation in \( \text{SP}^0 \) via this map.

Our proof of Theorem 2 will proceed via a detailed analysis of the model \( \text{SP}^0 \). Specifically, in Sections 3 and 4 we will show the following:

**Theorem 5** For any \( k \geq 1 \), the elements \( [Y_{k+1}] \) and \( [Y_{0 \Rightarrow 0(k+1)}] \) in \( \text{SP}^0 \) are not PCF\( _k^0 \)-definable.

For \( Y_{k+1} \), this is the best that we can currently achieve; but for \( Y_{0 \Rightarrow 0(k+1)} \), we will go on in Section 5 to show that no \( Z \cong [Y_{0 \Rightarrow 0(k+1)}] \) can be PCF\( _k^0 \)-definable, which will establish Theorem 2.

Finally, we note that each set SF(\( \sigma \)) naturally carries an \textit{observational ordering}, namely the partial order \( \preceq \) given by
\[
\begin{align*}
x \preceq x' \iff & \quad \forall f \in \text{SF}(\sigma \to \mathbb{N}), \ n \in \mathbb{N}, \ (f \cdot x = n) \Rightarrow (f \cdot x' = n).
\end{align*}
\]

Clearly, the application operations \( \cdot \) are monotone with respect to \( \preceq \); moreover, it follows from an inequational version of the context lemma that the observational ordering on any SF(\( \sigma_0, \ldots, \sigma_{r-1} \to \mathbb{N} \)) coincides with the pointwise
ordering induced by the usual partial order on \( \mathbb{N} \). We shall also use the symbol \( \preceq \) for the preorder on each \( SP^0(\sigma) \) induced by \( \leq \) on \( SF \); this also coincides with the observational preorder on \( SP^0(\sigma) \) defined by

\[
q \preceq q' \iff \forall p \in SP^0(\sigma \rightarrow \mathbb{N}), n \in \mathbb{N}, (p \cdot q = \lambda n. \Rightarrow p \cdot q' = \lambda n).
\]

Note too that \( q \approx q' \iff q \preceq q' \preceq q \). Finally, we shall allow the use of the notations \( \approx, \preceq \) for open terms (in the same environment) and indeed for metaterms: e.g. \( \vec{x} \vdash P \approx P' \iff \ll \lambda \vec{x}.P \gg \approx \ll \lambda \vec{x}.P' \gg \).

### 2.4 The embeddability hierarchy

The following result will play a central role in this paper; it is a mild generalization of Theorem 7.7.1 of [3].

**Theorem 6 (Strictness of embeddability hierarchy)** In \( SF \), no type \( k+1 \) can be a retract of any countably infinite product \( \prod_i \sigma_i \) where each \( \sigma_i \) is an extended type of level \( \leq k \). More formally, if \( z \) is a variable of type \( k+1 \), \( x_i \) a variable of type \( \sigma_i \) and \( \Gamma \) the set of the \( x_i \), there cannot exist procedures \( z \vdash t_i : \sigma_i, \Gamma \vdash r : k+1 \) such that

\[
\ll \lambda \vec{x}. \ll \lambda y.0 \gg \gg \ll \lambda y.0 \gg = 0.
\]

Here we have referred informally to an infinite product \( \prod_i \sigma_i \) which we have not precisely defined, although our formal statement gives everything that is necessary for our purposes. In fact, one may make precise sense of this notation as an infinite product within the Karoubi envelope \( K(SF) \) as studied in [3, Chapter 4]: for instance, if we inductively define types \( \tau_k \) by

\[
\tau_0 = \mathbb{N}, \quad \tau_{k+1} = (\mathbb{N} \rightarrow \tau_k) \rightarrow \mathbb{N},
\]

then it is clear that every extended type \( \sigma_i \) of level \( \leq k \) can be embedded in \( \tau_k \) (a type of level \( \leq k+1 \)), so that the countable product \( \Pi_i \sigma_i \) can be constructed as a retract of \( \mathbb{N} \rightarrow \tau_k \).

Since the proof of the above hierarchy theorem is slightly more elaborate in some respects than that of [3, Theorem 7.7.1], we include it in full here:

**Proof** By induction on \( k \). For our induction claim, we in fact work with the slightly stronger statement that \( k+1 \) is not even a pseudo-retract of any \( \Pi_i \sigma_i \) with \( \text{lv}(\sigma_i) \leq k \): that is, there do not exist procedures \( t_i \) and \( r \) of the above types such that \( \ll r[\vec{x} \mapsto \vec{t}] \gg \gg \geq z^n \).

For the case \( k = 0 \), we note that \( \mathbb{N} \) is the only extended type of level 0, so suppose for contradiction that we have \( z^{\mathbb{N} \rightarrow \mathbb{N}} \vdash t_i : \mathbb{N} \) and \( \Gamma \vdash r : \mathbb{N} \rightarrow \mathbb{N} \) such that \( \ll r[\vec{x} \mapsto \vec{t}] \gg \gg \geq z^n \). Then

\[
\ll (r[\vec{x} \mapsto \vec{t}]|z \mapsto \lambda y.0)](\lambda . \bot) \gg \geq \ll (\lambda y.0)(\lambda . \bot) \gg = 0.
\]

So writing \( r = \lambda y.e \), the path through \( e \) taken by the above computation may involve a sequence of ‘queries’ \textbf{case} \( x_i \) of \( (\cdot \cdot \cdot) \) (ending at a leaf 0), but no
queries \texttt{case y of \ldots}. At this point, we shall assume that our \(t_i\) and \(r\) are chosen so that the number \(n\) of queries along this path is as small as possible.

Clearly \(n \neq 0\), otherwise \(\langle r[x] \mapsto t \mid z \mapsto w\rangle\) \(\gg\) \(\bot\) for any \(w \in S^0(1)\) and \(s \in S^0(0)\). So suppose that \(e = \texttt{case } x_i \texttt{ of } (j \mapsto e_j)\). Then for any \(w \neq \lambda y.\bot\), we must have \(\langle t_i[z \mapsto w] \gg\) \(\bot\), otherwise taking any \(s\) with \(w\) \(\downarrow\) we have that \(\langle r[x] \mapsto t \mid z \mapsto w\rangle\) \(\gg\) \(\bot\). Hence also \(\langle t_i[z \mapsto \lambda y.\bot] \gg\) \(= m \in \mathbb{N}\) by stability of \(t_i\), since we may easily take \(w_0, w_1\) bounded above with \(w_0 \cap w_1 = \lambda y.\bot\). Thus \(t_i\) gives no information about \(z\), in that \(\langle t_i[z \mapsto w] \gg\) \(= m\) for all \(w\). This means that \(\lambda y.\, e_m\) can equally well play the role of \(r\), which contradicts the minimality of \(n\). This completes the proof for \(k = 0\).

Now assume the result for \(k - 1\), and suppose for contradiction that \(t_i, r\) exhibit \(k + 1\) as a pseudo-retract of \(\Pi \sigma_i\), where each \(\sigma_i\) is of level \(\leq k\). Let \(v = \langle r[x] \mapsto t \rangle \gg\), so that \(\langle v[z \mapsto w] \gg\) \(\supseteq w\) for any \(w \in S^0(k + 1)\). We first check that any \(v\) with this latter property must have the syntactic form \(\lambda f^k.\texttt{case } zp \texttt{ of } \ldots\) for some \(p\) of type \(\mathcal{E}\). Indeed, it is clear that \(v\) does not have the form \(\lambda f.\, n\) or \(\lambda f.\bot\), and the only other alternative form is \(\lambda f.\texttt{case } fp' \texttt{ of } \ldots\). In that case, however, we would have

\[
\langle v[z \mapsto \lambda x.0] \gg \cdot (\lambda y^{k-1}.\bot) = \bot,
\]

contradicting \(\langle v[z \mapsto \lambda x.0] \gg \cdot (\lambda y^{k-1}.\bot) \gg (\lambda x.0)(\lambda y.\bot) = 0\).

We now focus on the subterm \(p\) in \(v = \lambda f.\texttt{case } zp \texttt{ of } \ldots\). The general direction of our argument will now be to show that \(\lambda f^k.\, p\) represents a function of type \(\mathcal{K} \rightarrow \mathcal{K}\) that dominates the identity, and that moreover our two-stage construction of \(v\) via \(u\) can be used to split this into morphisms \(\mathcal{K} \rightarrow \Pi \rho_j\) and \(\Pi \rho_j \rightarrow \mathcal{E}\) where the \(\rho_j\) are of level \(\leq k - 1\), contradicting the induction hypothesis. An apparent obstacle to this plan is that \(z\) as well as \(f\) may appear free in \(p\); however, it turns out that we still obtain all the properties we need if we specialize \(z\) (somewhat arbitrarily) to \(\lambda x.0\).

Specifically, we claim that \(\langle p[f \mapsto q, z \mapsto \lambda x.0] \gg\supseteq q\) for any \(q \in S^0(k)\). For suppose that \(q \cdot s = n \in \mathbb{N}\) whereas \(\langle p[f \mapsto q, z \mapsto \lambda x.0] \gg\supseteq s\neq n\) for some \(s \in S^0(k - 1)\). Take \(w = \lambda q.\texttt{case } gs \texttt{ of } \ldots\) \(\gg\) \(0\), so \(w \cdot q' = \bot\) whenever \(q' \cdot s \neq n\). Then \(w \leq \lambda x.0\) pointwise, so we have \(\langle p[f \mapsto q, z \mapsto w] \gg\supseteq s\neq n\) by Exercise ???. By the definition of \(w\), it follows that \(\langle (zp)[f \mapsto q, z \mapsto w] \gg\supseteq \bot\), and hence that \(\langle v[z \mapsto w] \gg \cdot q = \bot\), whereas \(w \cdot q = 0\), contradicting \(\langle v[z \mapsto w] \gg\supseteq w\). We have thus shown that \(\lambda f.\, \langle p[z \mapsto \lambda x.0] \gg\supseteq id_k\).

We next show how to split the function represented by this procedure so as to go through some \(\Pi \rho_j\). As above. Since \(\langle r[x] \mapsto t \mid \rangle \gg\) \(\lambda f.\texttt{case } zp \texttt{ of } \ldots\), we have that \(r[x] \mapsto t\) reduces in finitely many steps to a head normal form \(\lambda f.\texttt{case } zP \texttt{ of } \ldots\) where \(\langle P \gg\) \(= p\). By working backward through this reduction sequence, we may locate the ancestor within \(r[x] \mapsto t\) of this head occurrence of \(z\). Since \(r\) is closed, this occurs within some \(t_i\), and clearly it must appear as the head of some subterm \(\texttt{case } zP' \texttt{ of } \ldots\) where \(P'\) is a substitution instance of \(P\). Now since \(t_i\) has type \(\sigma_i\) of level \(\leq k\), and \(z : \overline{k + 1}\) is its only free variable, it is easy to see that all \textit{bound} variables within \(t_i\) have pure types of level \(< k\). (There may be infinitely many of these, since \(\sigma_i\) is an extended
type.) Let \( h_0, h_1, \ldots \) denote the bound variables that are in scope at the relevant occurrence of \( z^P \); and suppose each \( h_j \) has type \( \rho_j \) of level \( < k \). By considering the form of the head reduction sequence \( r[\vec{x} \mapsto \vec{t}] \rightsquigarrow_h^* \lambda f. \text{case } z^P \text{ of } (\cdots) \), we see that \( P = P'[\vec{h} \mapsto \vec{H}] \) where each \( H_j : \rho_j \) contains at most \( f \) and \( z \) free.

Again taking advantage of product types in \( K(\mathbf{SP}^0) \), and writing \( \ast \) for the substitution \( \left[ z \mapsto \lambda x.0 \right] \) and \( \Delta \) for the set of variables \( h_j \), define procedures

\[
f^k \vdash t'_j = H_j^* : \rho_j, \quad \Delta \vdash r' = P^* : \kappa.
\]

Then \( r' \circ t' \) coincides with the term \( \ll \lambda f. P^* \gg = \lambda f. \ll p^* \gg \), which dominates the identity as shown above. Thus \( \mathcal{K} \) is a pseudo-retract of \( \Pi_j \rho_j \), which contradicts the induction hypothesis. So \( k + 1 \) is not a pseudo-retract of \( \Pi_i \sigma_i \) after all, and the proof is complete. □

\[\text{2.5 The computational power of PCF}_1\]

Our main results will in effect present a hierarchy of sublanguages of PCF with PCF\(_1\) as the bottom rung. However, there are also other known sublanguages that are either weaker than or incomparable with PCF\(_1\). We here offer a brief summary of known results in order to situate our theorems more fully within this broader picture.

That PCF\(_1\) surpasses the power of PCF\(_0\) is true but uninteresting, since the latter is an extremely weak language that does not even define addition. More representative is that PCF\(_1\) is strictly stronger than the language \( T_0 + \min \), where \( T_0 \) (a fragment of Gödel’s System T) is the \( \lambda \)-calculus with constants

\[
\hat{0} : \mathbb{N}, \quad \text{suc} : \mathbb{N} \rightarrow \mathbb{N}, \quad \text{rec}_0 : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N}),
\]

(the last of these being the standard operator for primitive recursion), and \( \min \) is the classical minimization operator of type \( (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \). On the one hand, it is an easy exercise to define both \( \text{rec}_0 \) and \( \min \) in PCF\(_1\); on the other hand, Berger [1] shows that the functional \( F : (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N}) \) recursively defined by

\[
F h n = h n (F h (n + 1)),
\]

whilst readily expressible in PCF\(_1\), is not definable in \( T_0 + \min \). This situation is revisited in [3] from the perspective of substructures of \( \mathbf{SP}^0 \); it is shown that the language \( T + \min \) can be modelled within the substructure of \( \text{left-well-founded} \) procedures, but that the above functional \( F \) is not representable by such a procedure, whence \( F \) is not definable even in \( T + \min \). (This in a sense offers a prototype for the method adopted in this paper, in that we shall identify a structure of \( \mathbf{SP}^0 \) that models PCF\(_k\) but not PCF\(_{k+1}\).) At third order, there are even ‘hereditarily total’ functionals definable in PCF\(_1\) but not in \( T + \min \), one example being the well-known \textit{bar recursion} operator (see [2]).
3 A substructure of $\text{SP}^0$ for $\text{PCF}_k$

For the rest of the paper, we take $k$ to be some fixed natural number greater than 0.

We shall define a certain substructure $A^0_k$ of $\text{SP}^0$, whose elements we call the $k$-acceptable procedures, in such a way that $A^0_k$ provides a model for $\text{PCF}_k^\Omega$ but excludes $Y_{k+1}$. The substructure $A^0_k$ will be constructed inductively by means of a system of term formation operations that generate these procedures, somewhat similar character to the formation operations used in [3] to generate left-well-founded procedures (i.e. those containing no infinite descending chains of application subterms). However, since the NSPs for the $Y_l$ are not LWF, the definition of $A^0_k$ will need to contain some provision for the generation of non-LWF procedures—albeit in a carefully controlled way, since we wish to generate $Y_k$ but not $Y_{k+1}$.

To motivate the definition of $A^0_k$, let us try to explain informally what is the characteristic of $Y_{k+1}$ that $A^0_k$ is designed to exclude. Recall that for any $l > 0$, the NSP $Y_l$ is defined as $\lambda g : l \to l. F_l[g]$, where $F_l[g]$ is the procedure defined corecursively by

$$F_l[g] = \lambda x^{l-1}. \text{case } g(F_l[g]) x^n \text{ of } (i \Rightarrow i) .$$

The manifest difference between $Y_{k+1}$ and $Y_k$, then, is that $Y_{k+1}$ involves an infinite sequence of nested calls to a variable $g$ of type level $k+2$, whereas $Y_k$ does not.

One’s first thought might therefore be to try and show that no procedure involving an infinite nesting of this kind can be constructed via $\text{PCF}_k$ alone. As it stands, however, this is not the case. Suppose, for example, that $up_k : k \to k+1$ and $down_k : k+1 \to k$ are PCF$_1$ terms defining a standard retraction $k < k+1$. Specifically, let us inductively define

$$up_0 = \lambda x^0. \lambda z^0. x , \quad down_0 = \lambda y^1. \lambda \bar{y}^0 ,$$

$$up_{k+1} = \lambda x^{k+1}. \lambda z^{k+1}. x(down_k z), \quad down_{k+1} = \lambda y^{k+2}. \lambda w^{k}. y(up_k w) .$$

Now consider the PCF$_k$ program

$$Z_{k+1} = \lambda g : (k+1) \to (k+1). up_k (Y_k(down_k \circ g \circ up_k)) .$$

A simple calculation shows that the NSPs for $Y_{k+1}$ and $Z_{k+1}$ are superficially very similar in form, both involving an infinite sequence of nested calls to $g : (k+1) \to (k+1)$. (These NSPs are shown schematically in Figure 3 for the case $k = 2$.) We will therefore need to identify some more subtle property of NSPs that differentiates between $Y_{k+1}$ and $Z_{k+1}$.

The intuitive idea will be that in the NSP for $Z_{k+1}$, the full potency of $g$ as a variable of type $k+1 \to k+1$ is not exploited, since both the input and output to $g$ are ‘funneled’ through the simpler type $k$. (The force of Theorem 6 is that the type $k$ cannot fully represent the structure of the type $k+1$.) Broadly speaking, we shall construct a model $A^0_k$ which admits infinite
Figure 1: The NSPs for $Y_3$ and $Z_3$. Here $\lambda . z$ abbreviates $\lambda . \text{case } z \text{ of } (i => i)$. 
nesting for variables of high type provided that the resulting information is 
funneled sufficiently often through a type of level $\leq k$, but not otherwise. This 
model will then contain the NSP for $Z_{k+1}$, but not that for $Y_{k+1}$.

We shall define $A^0_k$ by first defining a larger set $A_k$ of meta-procedures-
in-context $\Gamma \vdash P : \sigma$, then taking $A^0_k$ to consist of the closed normal forms 
in $A_k$. Our construction will make use the following notion of $k$-plugging for 
meta-terms. Throughout the paper, we shall use Greek capitals $\Gamma, \Delta$ for arbi-
trary environments, and Roman capitals $Z, V, X$ for sets of variables that are 
constrained to be of type level $\leq k$.

**Definition 7 (k-plugging)** Suppose given the following data:

- a possibly infinite variable environment $\Gamma$ (i.e. a set of typed variables),
- a possibly infinite set $Z$ of ‘plugging variables’ $z$ of level $\leq k$, disjoint from 
$\Gamma$,
- a root meta-term $U$ in context $\Gamma, Z$,
- for each variable $z^\sigma \in Z$, a meta-term $\xi(z)$ in context $\Gamma, Z$.

We may assume here by $\alpha$-conversion that no variables in $\Gamma, Z$ appear 
bound by a $\lambda$-abstraction in $U$ or in any of the $\xi(z)$.

In this situation, we define the $k$-plugging $\Pi(U, \xi)$ (also written as $\Pi_{\Gamma, Z}(U, \xi)$) 
to be the meta-term obtained from $U$ by repeatedly expanding variables $z \in Z$ to 
$\xi(z)$. More formally, writing $T^\circ$ for the meta-term obtained from $T$ by replacing 
all subterms $z^Q$ with $z \in Z$ by $\bot$, we may define

\[
\Pi^0(U, \xi) = U,
\]
\[
\Pi^{n+1}(U, \xi) = \Pi^n(U, \xi)[z \mapsto \Pi^n(\xi(z)) \text{ for all } z \in Z],
\]
\[
\Pi(U, \xi) = \bigcup_n \Pi^n(U, \xi)^\circ,
\]

where $\bigcup$ denotes supremum with respect to the syntactic order on meta-terms.

It is easy to see that $\Pi_{\Gamma, Z}(U, \xi)$ is well-typed in environment $\Gamma$.

**Lemma 8** Under the conditions of the above definition, we have

\[
\preceq \Pi_{\Gamma, Z}(U, \xi) \succeq = \preceq \Pi_{\Gamma, Z}(\preceq U \succeq, \preceq \xi \succeq) \succeq,
\]

where $\preceq \xi \succeq$ denotes the mapping $z \mapsto \preceq \xi(z) \succeq$.

**Proof** Since $\preceq \cdot \succeq$ is continuous, this will follow from the general fact 
that $\preceq \Pi^m(U, \xi)^\circ \succeq = \preceq \Pi^m(\preceq U \succeq, \preceq \xi \succeq)^\circ \succeq$ for each $m \in \mathbb{N}$ and all $U, \xi$. 
First, it follows easily from Definition 7 that

\[
\Pi^0(U, \xi)^\circ = U^\circ,
\]
\[
\Pi^{m+1}(U, \xi)^\circ = U[z \mapsto \Pi^m(\xi(z), \xi)^\circ] \text{ for all } z \in Z.
\]
We now argue by induction on \( m \). When \( m = 0 \), the desired equation holds because \( \ll U^0 \gg = \ll U \gg^0 \). For the induction step, we have

\[
\ll \Pi^{m+1}(U, \xi)^0 \gg = \ll U[Z \mapsto \Pi^m(\xi(z), \xi)^0] \gg \\
= \ll \ll U \gg[Z \mapsto \ll \Pi^m(\xi(z), \xi)^0] \gg \text{ by the evaluation theorem} \\
= \ll \ll U \gg[Z \mapsto \ll \Pi^m(\ll \xi(z) \gg, \ll \xi \gg)^0] \gg \text{ by the induction hypothesis} \\
= \ll \ll \Pi^{m+1}(\ll U \gg, \ll \xi \gg)^0 \gg ,
\]

which completes the proof. \( \square \)

We now give our central definition specifying the class \( A_k \) of interest. Unlike the definition of (say) the class of left-well-founded meta-terms which is framed directly in terms of a manifest structural property of syntax trees (see [3], Section 6.3), the present definition has an inductive character, consisting of a set of rules which suffice to generate all the meta-terms of \( A_k \). This enables us to capture the rather subtle structural difference between the above procedures \( Y_{k+1} \) and \( Z_{k+1} \), though it also means that some non-trivial effort will be needed to show that \( Y_{k+1} \) is not present in \( A_k \) (see Section 4). As in the grammar above, we let \( P, Q, R \) range over meta-procedures, \( D, E \) over meta-expressions, and \( G \) over ground meta-terms.

**Definition 9** (i) The class \( A_k \) of \( k \)-acceptable meta-terms-in-context is generated inductively by means of the following clauses:

1. \( n, \bot \in A_k \) for any \( n \in \mathbb{N} \).
2. If \( \Gamma, x_0, \ldots, x_{r-1} \vdash E \in A_k \), then \( \Gamma \vdash (\lambda x_0 \ldots x_{r-1}. E) \in A_k \).
3. If \( x : \sigma_0, \ldots, \sigma_{r-1} \rightarrow \mathbb{N} \in \Gamma \) and \( \Gamma \vdash Q_i : \sigma_i \in A_k \) for each \( i < r \), then \( \Gamma \vdash xQ_0 \ldots Q_{r-1} \in A_k \).
4. If \( \Gamma \vdash G \in A_k \) and \( \Gamma \vdash E_i \in A_k \) for each \( i \), then \( \Gamma \vdash \text{case} \; G \; \text{of} \; (i \mapsto E_i) \in A_k \).
5. If \( \Gamma \vdash P : \sigma_0, \ldots, \sigma_{r-1} \rightarrow \mathbb{N} \in A_k \) and \( \Gamma \vdash Q_i : \sigma_i \in A_k \) for each \( i < r \), then \( \Gamma \vdash PQ_0 \ldots Q_{r-1} \in A_k \).
6. Plugging rule: If \( \Gamma, Z, U, \xi \) are as in Definition 7 (with respect to \( k \)), where \( U \) is some normal-form expression \( \Gamma, Z \vdash e \in A_k \), and each \( \xi(z) \) is a normal form procedure \( \Gamma, Z \vdash \xi(z) \in A_k \), then \( \Gamma \vdash \ll \Pi_\Gamma Z(e, \xi) \gg \in A_k \).

(ii) For each type \( \sigma \) and environment \( \Gamma \), let \( A_k(\Gamma, \sigma) \) consist of all (normal-form) procedures \( p \in \text{SP}(\sigma) \) such that \( \Gamma \vdash p \in A_k \), and let \( A_k^\bot(\sigma) = A_k(\emptyset, \sigma) \), so that \( A_k^\bot(\sigma) = A_k(\sigma) \cap \text{SP}^\bot(\sigma) \).
Note that the $k$-acceptable meta-terms that can be constructed without recourse to the plugging rule are exactly the \textit{well-founded} meta-terms as studied in Section 6.3 of [3]. Note too that the generation of \textit{normal-form} $k$-acceptable terms is self-contained, in that if $t$ is a normal form, the derivation of $\Gamma \vdash t \in A_k$ does not involve any non-normal forms $\Gamma' \vdash T \in A_k$ (and also does not require rule 5).

To give some intuition for the above definition, the idea is that if $M$ is a closed term of $\text{PCF}_k^\Omega$, then the NSP $[M]$ will be an element of $A_k$ whose construction via the clauses of Definition 9 broadly parallels the syntactic structure of $M$. In particular, for every occurrence of an operator $Y_{\sigma}$ within $M$, say at a position in which the variables in scope are those of $\Gamma$, the construction of $[M]$ will involve an application of the plugging rule in environment $\Gamma$. Thus, if $M$ involves multiple occurrences of operators $Y_{\sigma}$, then multiple applications of the plugging rule (perhaps with different $\Gamma$) may be needed to construct $[M]$.

On the other hand, if the inductive definition of our class $A_k$ were to mirror the syntactic structure of $\text{PCF}_k^\Omega$ terms too closely, it may become too hard to show that this class excludes $Y_{k+1}$. Thus, as an extremal case, one might consider the substructure of $\text{SP}^0$ that consists by definition of the $\text{PCF}_k^\Omega$-denotable procedures, but clearly this would afford no reduction of our original problem.

We now proceed to show that the sets $A_k(\sigma)$ constitute a well-behaved substructure of $\text{SP}$ that suffices for the interpretation of $\text{PCF}_k^\Omega$. We begin with a simple analysis of the leaf structure of meta-terms that will play a role in our treatment of \textit{case} expressions.

\textbf{Definition 10} (i) The set of rightward (occurrences of) numeral leaves within a meta-term $T$ is defined inductively by means of the following clauses:

1. A meta-term $n$ is a rightward numeral leaf within itself.
2. Every rightward numeral leaf within $E$ is also one within $\lambda \vec{x}.E$.
3. Every rightward numeral leaf in each $E_i$ is also one in \textit{case} $G$ of $(i \Rightarrow E_i)$.
4. Every rightward numeral leaf within $P$ is also one within $P\vec{Q}$.

(Note that there are no rightward numeral leaves within a meta-term $x\vec{Q}$. Note too that clause 4 is not required for computing the rightward leaves of a meta-procedure $P$ or meta-expression $E$.)

(ii) If $T$ is a meta-term and $E_i$ is a meta-expression for each $i$, we write $T[i \mapsto E_i]$ for the result replacing each rightward leaf occurrence $i$ in $T$ by $E_i$.

\textbf{Proposition 11} $\ll \text{case} d \text{ of } (i \Rightarrow e_i) \gg = d[i \mapsto e_i]$ for any expressions $d, e_i$.

\textbf{Proof} Define a ‘truncation’ operation $-(c)$ on normal-form expressions for each $c \in \mathbb{N}$ as follows:

\[ n^{(c)} = n, \quad \bot^{(c)} = \bot, \]
\[ \text{case} g \text{ of } (i \Rightarrow e_i)^{(0)} = \bot, \]
\[ \text{case} g \text{ of } (i \Rightarrow e_i)^{(c+1)} = \text{case} g \text{ of } ((i \Rightarrow e_i)^{(c)}). \]
Then clearly $d = \bigsqcup_c d^{(c)}$ and $d[i \mapsto e_i] = \bigsqcup_c d^{(c)}[i \mapsto e_i]$. Moreover, we may show by induction on $c$ that

$$\ll \text{case } d^{(c)} \text{ of } (i \mapsto e_i) \gg = d^{(c)}[i \mapsto e_i].$$

The case $c = 0$ is trivial since $d^{(0)}$ can only have the form $n$ or $\bot$. For the induction step, the situation for $d = n, \bot$ is trivial, so let us suppose $d = \text{case } g \text{ of } (j \mapsto f_j)$. Then

$$\ll \text{case } d^{(c+1)} \text{ of } (i \mapsto e_i) \gg$$

$$= \ll \text{case } (\text{case } g \text{ of } (j \mapsto f_j^{(c)})) \text{ of } (i \mapsto e_i) \gg$$

$$= \text{case } g \text{ of } (j \mapsto \ll \text{case } f_j^{(c)} \text{ of } (i \mapsto e_i) \gg)$$

$$= \text{case } g \text{ of } (j \mapsto (f_j^{(c)}[i \mapsto e_i])) \text{ by induction hypothesis}$$

$$= (\text{case } g \text{ of } (j \mapsto f_j^{(c)}))[i \mapsto e_i]$$

$$= d^{(c+1)}[i \mapsto e_i].$$

Since $\ll - \gg$ is continuous, the proposition follows by taking the supremum over $c$. □

**Lemma 12**

(i) If $T \rightsquigarrow T'$ then the rightward numeral leaves in $T'$ are exactly the residuals in $T'$ of rightward numeral leaves in $T$.

(ii) The rightward numeral leaves in $\ll T \gg$ are precisely the residuals of rightward numeral leaves in $T$.

(iii) $\ll T[n \mapsto E_n] \gg = \ll T \gg [n \mapsto \ll E_n \gg]$.

**Proof** (i) Easy by induction on the generation of the one-step reduction relation $\rightsquigarrow$ as given in Section 2.

(ii) If $n$ is any rightward numeral leaf in $\ll T \gg$, then by the definition of $\ll T \gg$, we may take a finite normal form $t \sqsubseteq \ll T \gg$ containing $n$ and a finite reduction sequence $T \rightsquigarrow^* T'$ such that $t \sqsubseteq T'$. By considering the chain of subterms witnessing that $n$ is a rightward leaf in $\ll T \gg$, we see that the corresponding occurrence of $n$ is also a rightward leaf in $t$ and in $T'$. So by (i), this occurrence is a residual in $T'$ of a rightward leaf $n$ in $T$; and since this occurrence falls within $t$, this is to say that it is a residual in $\ll T \gg$ of this leaf in $T$. Moreover, this argument is easily reversible, so that any residual in $\ll T \gg$ of a rightward leaf $n$ in $T$ is itself a rightward leaf in $\ll T \gg$.

(iii) Let $e_n = \ll E_n \gg$ for each $n$; then by the evaluation theorem we have $\ll T[n \mapsto E_n] \gg = \ll T[n \mapsto e_n] \gg$. Let us write $\bullet$ for the operation $[n \mapsto e_n]$. As in (i), we see that any finite reduction $T \rightsquigarrow T_1 \rightsquigarrow \cdots \rightsquigarrow T_s$ instantiates to a reduction $T^* \rightsquigarrow T_1^* \rightsquigarrow \cdots \rightsquigarrow T_s^*$; in particular, the operation $\bullet$ can never ‘block’ one of these reduction steps, since no residual of a rightward leaf $n$ can occur as the branching condition in a subterm $\text{case } n \text{ of } (\cdots)$. Conversely, any redex in $T^*$ is an instantiation of a redex in $T$, and similarly for each $T_i$, so any finite reduction sequence for $T^*$ is the $\bullet$-instantiation of a reduction sequence for $T$.

20
We now see that if \( T \leadsto^\delta T' \) and \( t \subseteq T' \) is a normal form, then \( T^* \leadsto^\delta T'^* \) and \( t^* \subseteq T'^* \) is a normal form; and since \( \cdot \) commutes with \( \cup \), this shows that \( \ll T \gg T^* \ll T' \gg \). Conversely, if \( T^* \leadsto^\delta U \) and \( u \subseteq U \) is a normal form, then \( U = T'^* \) for some \( T \leadsto^\delta T' \), and there is some \( t \subseteq T' \) with \( u \subseteq t^* \); this shows that \( \ll T^* \gg \ll T \gg \). Thus \( \ll T[n \mapsto E_n] \gg = \ll T^* \gg = \ll T \gg \) as required. □

**Lemma 13** If \( \Gamma \vdash T \in \mathcal{A}_k \) and \( \Gamma \vdash e_n \in \mathcal{A}_k \) for each \( n \), where the \( e_n \) are normal-form expressions, then \( \Gamma \vdash T[n \mapsto e_n] \in \mathcal{A}_k \).

**Proof** By induction on the generation of \( T \in \mathcal{A}_k \) as in Definition 9. The cases for clauses 1–5 are easy. For clause 6, suppose \( \ll D \gg = \ll \Pi_{\Gamma,Z}(e,\xi) \gg \) for some \( \Gamma, Z, e, \xi \) satisfying the conditions of the plugging rule, so that \( \Gamma \vdash D \in \mathcal{A}_k \); we wish to show that \( \Gamma \vdash D[n \mapsto e_n] \in \mathcal{A}_k \). But by Lemma 12(iii) we have
\[
\ll D[n \mapsto e_n] \gg = \ll D \gg [n \mapsto e_n] = \ll \Pi_{\Gamma,Z}(e,\xi) \gg [n \mapsto e_n] = \ll \Pi_{\Gamma,Z}(e,\xi) [n \mapsto e_n] \gg,
\]
and it is easy to see that \( \Pi_{\Gamma,Z}(e,\xi)[n \mapsto e_n] = \Pi_{\Gamma,Z}(e[n \mapsto e_n],\xi) \), since all rightward leaves in \( \Pi_{\Gamma,Z}(e,\xi) \) must originate from \( e \). (Specifically, every occurrence of a plugging variable \( z \) within \( e \) must be at the head of some subexpression case \( zQ \) of \( \cdots \), so that the expansion of \( z \) does not contribute to the set of rightward leaf nodes.) But \( e[n \mapsto e_n] \) is a normal form and \( \Gamma, Z \vdash e[n \mapsto e_n] \in \mathcal{A}_k \) by the induction hypothesis; hence the plugging \( \Pi_{\Gamma,Z}(e[n \mapsto e_n],\xi) \) itself satisfies the conditions of rule 6, and this plugging now witnesses that \( \Gamma \vdash D[n \mapsto e_n] \in \mathcal{A}_k \). □

The core of our analysis is the proof of the following lemma. As in [3], we say a meta-term \( T \) is of order \( \delta \in \mathbb{N} \) if for every redex \( PQ \) within \( T, P \) has type level \( \leq \delta \) (so that the \( Q_i \) each have type level \( < \delta \)). Thus, a meta-term of order \( 0 \) may contain only nullary redexes \( \langle \lambda . E \rangle \); contracting these to \( E \) immediately yields a normal form.

**Lemma 14** If \( T \) is of finite order and \( \Gamma \vdash T \in \mathcal{A}_k \), then \( \Gamma \vdash \ll T \gg \in \mathcal{A}_k \).

**Proof** We will establish the following two claims by a simultaneous induction on \( \delta \in \mathbb{N} \):

1. For all \( T \) of order \( \delta \), if \( \Gamma \vdash T \in \mathcal{A}_k \) then \( \Gamma \vdash \ll T \gg \in \mathcal{A}_k \).

2. For all normal-form \( t \) and all \( \vec{q} \) of type level \( \leq \delta \), if \( \Gamma, \vec{x} \vdash t \in \mathcal{A}_k \) and \( \Gamma \vdash \vec{q} \in \mathcal{A}_k \) then \( \ll \Gamma \vdash t[\vec{x} \mapsto \vec{q}] \gg \in \mathcal{A}_k \).

We will show that both claims hold for \( \delta \) assuming they hold for all \( \delta' < \delta \), so that there is no need for a separate base case. For each \( \delta \), we first establish claim 1, then use this to help establish claim 2.

For claim 1, we argue by an inner induction on the generation of \( \Gamma \vdash T \in \mathcal{A}_k \) via the clauses of Definition 9. The cases for meta-terms \( n \) and \( \bot \) are trivial, and
those for meta-terms \( \lambda \vec{x}.E \) and \( xQ \) are straightforward, bearing in mind that \( \ll \lambda \vec{x}.E \gg = \lambda \vec{x}. \ll E \gg \) and \( \ll xQ_0 \cdots Q_{r-1} \gg = x \ll Q_0 \gg \cdots \ll Q_{r-1} \gg \). For meta-terms \( \Gamma \vdash \text{case } G \text{ of } (n \Rightarrow E_n) \), let \( g = \ll G \gg \) and \( e_n = \ll E_n \gg \) for each \( n \), so that \( \Gamma \vdash g, e_n \in \mathcal{A}_k \) by the induction hypothesis. If \( g \) is an application \( xq \) then \( \ll \text{case } G \text{ of } (n \Rightarrow E_n) \gg = \ll \text{case } g \text{ of } (n \Rightarrow e_n) \gg \in \mathcal{A}_k \) by clause 4 of Definition 9. Otherwise, \( g \) is a normal-form expression, and

\[
\ll \text{case } G \text{ of } (n \Rightarrow E_n) \gg = \ll \text{case } g \text{ of } (n \Rightarrow e_n) \gg = g[n \mapsto e_n]
\]

by Proposition 11. But \( g[n \mapsto e_n] \in \mathcal{A}_k \) by Lemma 13.

The case for applications \( PQ \) follows a line of argument familiar from [3, Section 6.3]. Suppose \( \Gamma \vdash PQ_0 \cdots Q_{r-1} \in \mathcal{A}_k \) is of order \( \delta \), and let \( p = \ll P \gg \), \( q_i = \ll Q_i \gg \in \mathcal{A}_k \) so that \( \Gamma \vdash p, q_i \in \mathcal{A}_k \) by the inner induction hypothesis. We may assume \( r > 0 \), otherwise the conclusion \( \Gamma \vdash \ll PQ \gg \in \mathcal{A}_k \) is trivial; this also implies that \( \delta > 0 \). Write \( p \) as \( \lambda \vec{x}.e \); then \( \Gamma, \vec{x} \vdash e \in \mathcal{A}_k \), since clause 2 of Definition 9(i) provides the only possible way to generate \( \Gamma \vdash p \in \mathcal{A}_k \). Note too that each \( q_i \) has type level \( < \delta \) since \( PQ \) is of order \( \delta \), so that \( \Gamma \vdash e[\vec{x} \mapsto \vec{q}] \gg \in \mathcal{A}_k \) by claim 2 for \( \delta - 1 \). But \( \ll e[\vec{x} \mapsto \vec{q}] \gg = \ll (\lambda \vec{x}.e)\vec{q} \gg \ll PQ \gg \) using the evaluation theorem, so \( \Gamma \vdash \ll PQ \gg \in \mathcal{A}_k \) as required.

The case for the plugging rule itself is trivial, since this rule only generates normal forms in \( \mathcal{A}_k \). This completes the proof of claim 1 for \( \delta \).

To show claim 2 for \( \delta \), we again argue by an inner induction on the generation of \( \Gamma, \vec{x} \vdash t \in \mathcal{A}_k \), writing \( ^* \) for the substitution \( [\vec{x} \mapsto \vec{q}] \). The cases for clauses 1–5 are easy; for clause 3, notice that \( (x\vec{Q})^* = q_i(Q^*) \) which belongs to \( \mathcal{A}_k \) via clause 5. For clause 6, suppose \( t = \ll \Pi(\Gamma, \vec{x}), Z(e, \xi) \gg \) where \( (\Gamma, \vec{x}), Z, e, \xi \) satisfy the conditions of the plugging rule. Then

\[
\ll t^* \gg = \ll \ll \Pi(e, \xi) \gg^* \gg = \ll (\lambda \vec{x}. \ll \Pi(e, \xi) \gg)\vec{q}^* \gg = \ll (\lambda \vec{x}. \ll \Pi(e, \xi) \gg)\vec{q}^* \gg = \ll \Pi(e, \xi)^* \gg ,
\]

where the third equality holds by the evaluation theorem. But it is easy to see from Definition 7 that the relevant plugging commutes with the substitution \( ^* \), so that

\[
\ll \Pi(\Gamma, \vec{x}), Z(e, \xi)^* \gg = \ll \Pi(\Gamma, Z(e^*, \xi^*)) \gg
\]

where \( \xi^*(z) = \xi(z)^* \). Furthermore, as an instance of Lemma 8, we have that

\[
\ll \Pi(\Gamma, Z(e^*, \xi^*)) \gg = \ll \Pi(\Gamma, Z(e^1, \xi^1)) \gg
\]

where \( e^1 = \ll e^* \gg \) and \( \xi^1(z) = \ll \xi(z) \gg \). We have thus shown that

\[
\ll t^* \gg = \ll \Pi(\Gamma, Z(e^1, \xi^1)) \gg .
\]

It remains to check that this plugging conforms to the requirements of the plugging rule, and so witnesses \( \Gamma \vdash \ll t^* \gg \in \mathcal{A} \) as required. By the inner induction hypothesis we have \( \Gamma \vdash e^* \in \mathcal{A}_k \); moreover, \( e^* \) is of order \( \delta \) since \( e \) is a normal form and the \( q_i \) are of level \( \leq \delta \), so by claim 1 for \( \delta \) (as established above) we have that \( \Gamma \vdash e^1 = \ll e^* \gg \in \mathcal{A}_k \). The same reasoning also shows
that \( \Gamma \vdash \xi^1(z) \in \mathcal{A}_k \) for each \( z \in Z \). Since also \( e^i \) and each \( \xi^1(z) \) are normal forms, the plugging \( \Pi_{\Gamma,Z}(e^1,\xi^1) \) satisfies the conditions of the plugging rule. This concludes the proof of claim 2 for \( \delta \), and the whole of the outer induction is now complete. \( \square \)

**Corollary 15** The sets \( \mathcal{A}_k(\sigma) \) form a substructure \( \mathcal{A}_k \subseteq \text{SP} \) closed under application. Likewise, the sets \( \mathcal{A}_k^0(\sigma) \) form an applicatively closed substructure \( \mathcal{A}_k^0 \subseteq \text{SP}^0 \).

**Proof** Suppose \( p \in \mathcal{A}_k(\sigma \to \tau) \) and \( q \in \mathcal{A}_k(\sigma) \). Then for a suitable \( \bar{x} \) we have \( \lambda \bar{x}.pq\bar{x}^n \in \mathcal{A}_k(\tau) \) using Definition 9(i); moreover, this term contains just a single redex so is of finite order. Thus \( p \cdot q = \ll \lambda \bar{x}.pq\bar{x}^n \gg \in \mathcal{A}_k \) by Lemma 14, so \( p \cdot q \in \mathcal{A}_k(\tau) \). Moreover, if \( p \) and \( q \) are closed then so is \( p \cdot q \). \( \square \)

**Proposition 16** If \( \text{lv}(\sigma) \leq k \), then \( Y_\sigma \in \mathcal{A}_k^0 \).

**Proof** Suppose \( \sigma = \sigma_0 \to \cdots \to \sigma_{r-1} \to \mathbb{N} \) has level \( \leq k \). Let \( g \) be a variable of type \( \sigma \to \sigma \), and for each \( i < r \), let \( x_i, x_i' \) be variables of type \( \sigma_i \). We also let \( z \) be a plugging variable of type \( \sigma \). Now let

\[
\begin{align*}
p &= \lambda x_0' \cdots x_{r-1}' \cdot \text{case } gz^n \cdot x_0' \cdots x_{r-1}' & \text{of } (i \Rightarrow i) , \\
e &= \text{case } g\bar{z}^nx_0 \cdots x_{r-1} & \text{of } (i \Rightarrow i) ,
\end{align*}
\]

and let \( \Gamma = \{ g, x_0, \ldots, x_{r-1} \} \), \( Z = \{ z \} \), and \( \xi(z) = p \). Then the plugging \( \Pi_{\Gamma,Z}(e,\xi) \) conforms to the requirements of the plugging rule since \( e \) and \( p \) are well-founded and in normal form, and clearly \( Y_\sigma = \lambda \bar{z} \bar{\bar{x}}. \ll \Pi_{\Gamma,Z}(e,\xi) \gg \). Thus \( Y_\sigma \in \mathcal{A}_k^0 \) via clauses 6 and 2 of Definition 9. (Note that we use two sets of variables \( \bar{x} \) and \( \bar{\bar{x}} \) here simply to conform to the stipulation on bound variables in Definition 7.) \( \square \)

We are now ready to consider the interpretation of \( \text{PCF}_k^\Omega \) in \( \text{SP}^0 \) as defined by the clauses in Section 2:

**Theorem 17** If \( \Gamma \vdash M : \sigma \) is any term of \( \text{PCF}_k^\Omega \), then \( [M]_\Gamma \in \mathcal{A}_k(\Gamma,\sigma) \).

**Proof** By induction on the structure of \( M \). For the terms \( x^\sigma, \bar{n}, \text{succ}, \text{pre}, \text{iszero} \) and \( C_f \), it suffices to note that the corresponding NSPs are well-founded, and the case of \( Y_\sigma \) where \( \text{lv}(\sigma) \leq k \) is handled by Proposition 16. The induction case for \( \lambda \)-abstraction is trivial using clause 2 of Definition 9, and the case for application is given by Corollary 15. \( \square \)

### 4 Acceptable procedures are gremlin-free

In this section, we will prove that the element \( Y_{k+1} \in \text{SP}^0 \) is not \( \text{PCF}_k^\Omega \)-definable, by showing that \( Y_{k+1} \notin \mathcal{A}_k \). Recall that the procedure \( Y_{k+1} \) is defined as \( \lambda \bar{g}(k+1) \cdot (k+1) \cdot x^k.C[g,x] \), where \( C[g,x] \) may be given co-recursively (up to \( \alpha \)-equivalence) by:

\[
C[g,x] = \text{case } g(\lambda x^k.C[g,x]) \cdot x^\eta \text{ of } (i \Rightarrow i) .
\]
Note that $C[g, x]$ contains an ‘infinite spine’ consisting of nested applications of $g$. Since the only way to generate $Y_{k+1} \in A_k$ is via rule 2 of Definition 9, it will suffice to show that $g, x \vdash C[g, x] \notin A_k$. We will regard $g : (k + 1) \to (k + 1)$ as a ‘global variable’ whose identity remains fixed throughout the subsequent development.

We will actually prove a more general claim than those outlined above, to the effect that no term in $A_k$ can contain a gremlin (roughly speaking, a certain complex of material that enables one to create a structure broadly similar to that of $C[g, x]$ or $C'[g, y, x]$). The notion of a gremlin will be defined formally in Definition 22, once we have introduced some necessary technical prerequisites. We will also explain how the same theory, with one tiny modification, provides what we will need to show in Section 5 that $Y_{0 \to (k+1)}$ is not PCF$^\Omega_k$-definable in SF.

First, we wish to generalize the form of terms illustrated by $C[g, x]$ above to allow the crucial application of $g$ to occur at certain positions other than the head of the term. However, we wish to exercise a certain control over what these positions might be. This is the purpose of the following definition:

**Definition 18** Suppose $\Gamma \vdash K[E]$ is a meta-term, with $E$ a meta-expression occurring at a position $K[-]$ (here $K[-]$ is a context with a single hole occurrence). We say that $K[-]$ is an exposed position if:

1. $K[-]$ does not contain any abstraction $\lambda x$ where $\text{lv}(x) > k$, and
2. the hole does not lie within any subterm $P_i$ of type level $\geq k$ occurring as an argument in an application $wP$, where $w \in \Gamma$ is of $\text{lv}(w) > k$.

By the local environment of the exposed position, we mean the set $\Delta$ of bound variables $x$ whose scope includes the hole, so that the environment in force at the hole is $\Gamma, \Delta$.

Condition 1 above means that the local environment $\Delta$ consists entirely of variables of level $\leq k$; note that the formulation of this condition also means that the hole can never enter the scope of such a variable when $T$ is evaluated. (For this reason, we shall often denote a local environment by $X$ to emphasize that it is of level $\leq k$.) Condition 2 (which suggests the term ‘exposed’) allows us to improve on this slightly in the case of normal-form contexts or similar:

**Lemma 19** Suppose $\Gamma \vdash T = K[E]$ is a meta-expression or meta-procedure of type level $\leq k$, where $K[E]$ is exposed and contains no $\beta$-redex with an operator of level $\geq k + 2$. Then the local environment $\Delta$ for $K$ consists entirely of variables of level $< k$.

**Proof** Suppose for contradiction that the hole in $K[-]$ appears within some $\lambda w. E'$ where $\text{lv}(w) \geq k$, and assume that $\lambda w$ is the outermost such binding. By the $\beta$-redex condition on $K[-]$, $\lambda w. E'$ does not appear as an argument to another $\lambda$-abstraction, and so is either the whole of $T$ or appears as an argument to some variable $v$. The former is impossible since $T$, if it is a procedure, is of
level \( \leq k \). In the latter case, \( v \) has level \( \geq k + 2 \); hence \( v \) cannot be bound within \( T \) since its binding would be outside that of \( w \). But by condition 2 of Definition 18, neither can we have \( \lambda w.E' \) (of level \( \geq k + 1 \) so certainly \( \geq k \)) as an argument to \( v \in \Gamma \), so we have a contradiction. (Note in passing that a variable \( w \) of level \( k - 1 \) can only arise when \( \lambda w.E' \) is the whole of \( T \).) \( \Box \)

The above lemma will be invoked at a crucial step in the proof of Lemma 25, and this is the reason why the machinery of exposed positions is necessary.

With the above notions in place, we may now introduce the notion of a *spinal term*, slightly generalizing the structure exhibited by the terms \( C[g,x] \).

**Definition 20** We coinductively declare an expression \( e \) to be spinal with respect to a variable \( x^k \) if \( e \) has the form

\[
\text{case } g(\lambda x'.E[e'])o \text{ of } (\cdots)
\]

where \( E[-] \) is a normal-form exposed position, \( e' \) is spinal with respect to \( x' \), and for some specialization \( ^o \) of the free variables of \( o \) apart from \( x \), we have \( o^o \approx x^0 \). (That is, we take the predicate 'e is spinal w.r.t. \( x' \) to be the largest predicate that satisfies the above statement.)

In the above setting, we may also refer to the application \( g(\lambda x'.E[e'])o \) as a spinal term, and may refer to the procedure term \( \lambda x'.E[e'] \) as pre-spinal.

At this point, we mention a small modification of this concept that we will need for the results of Section 5. Here we will work in a setting where the global variable \( g \) has the slightly different type \( 0 \rightarrow (k + 1) \rightarrow (k + 1) \). In this modified setting, we may vary the above definition by declaring \( e \) to be spinal w.r.t. \( x \) if \( e \) has the form

\[
\text{case } gb(\lambda x'.E[e'])o' o \text{ of } (\cdots)
\]

where \( b \) is a procedure term of type \( \overrightarrow{0} \) and the other conditions above are also satisfied. Subject to this adjustment, all the results and proofs of the present section go through in this modified setting, and the extra argument \( b \) plays no active role. For notational simplicity, therefore, we shall work here in the setting of a global variable \( g : (k + 1) \rightarrow (k + 1) \), on the understanding that the extra arguments \( b \) can be inserted where appropriate to make formal sense of the material in the modified setting. We do not expect that any confusion will arise in practice from this convention.

For either of our notions of ‘spinal’, we have the following simple lemma:

**Lemma 21** If \( \ll \text{case } G \text{ of } (i \Rightarrow E_i) \gg \) is spinal, then either \( \ll G \gg \) is a numeral or \( \ll G \gg \) is spinal.

**Proof** Working in the standard setting, if \( \ll \text{case } G \text{ of } (i \Rightarrow E_i) \gg \) is spinal then it has the form \( \text{case } g(\lambda x'.E[e'])o \text{ of } (\cdots) \) with \( E[-], e', o \) as in Definition 20(ii). Clearly \( \ll G \gg \neq \perp \), so \( \ll G \gg \) is either a numeral or has one of the forms

\[
g(\lambda x'.E[e'])o \quad \text{or} \quad \text{case } g(\lambda x'.E[e'])o \text{ of } (\cdots).
\]
In either of these cases, $\langle G \rangle$ is manifestly spinal. (The same proof clearly works in the modified setting with trivial adjustments.) □

We now combine these ingredients as follows. Once again, the following definition makes sense relative to either the standard or the modified notion of ‘spinal term’. We write $\Gamma^-$ for $\Gamma - \{g\}$.

**Definition 22** Suppose that $\Gamma, V \vdash t$ is a normal-form term, where $\text{lv}(V) \leq k$. By a *gremlin* within $t$ (with respect to $\Gamma$ and $V$), we shall mean a sub-expression $c$ of $t$ at an exposed position $K[-]$ with local environment $X$ (so that $t = K[c]$ and $\Gamma, V, X \vdash c$), such that for some specialization $[\vec{v} \mapsto \vec{s}]$ of the variables $v_i$ of $V, X$ to terms $\Gamma^- \vdash s_i$ (necessarily all of level $\leq k$), the term $\Gamma \vdash \langle c[\vec{v} \mapsto \vec{s}] \rangle$ is spinal (and necessarily a case-expression).

Besides the restriction on type levels, an important feature of this definition is that the terms $\vec{s}$ exist in the environment $\Gamma^-$ rather than $\Gamma$. This effectively means that all occurrences of $g$ within the spinal term $\langle c[\vec{v} \mapsto \vec{s}] \rangle$ must originate from $c$, since $g$ cannot appear in the $s_i$. Intuitively, we think of a gremlin as a structure that contains everything needed to manufacture a spinal subterm modulo the addition of relatively innocuous ingredients. By the leading occurrence of $g$ within $c$, we shall mean the one that gives rise to the head occurrence of $g$ within $\langle c[\vec{v} \mapsto \vec{s}] \rangle$.

Clearly $C[g, x]$, since it is spinal, is a gremlin within itself with respect to $\Gamma = g, x$ and $V = \emptyset$. As a generalization of the claim that $g, x \vdash C[g, x] \not\in A_k$, our main result in this section will be that no normal-form term $\Gamma, V \vdash t \in A_k$ (where $\text{lv}(V) \leq k$) can contain a gremlin (Theorem 27 below).

Our strategy will be broadly as follows. Suppose for contradiction that $t$ is some term of minimal ordinal rank in the generation of $A_k$ containing a gremlin. It will not be hard to show that the final rule in the generation of $t \in A_k$ must be either rule 3 or rule 6 from Definition 9. Using some technical machinery, we will show that even in these cases, a gremlin must already be present in one of the simpler terms in $A_k$ from which $t$ is constructed—intuitively because there is no way in which a gremlin can be ‘assembled’ from ingredients coming from different subterms. Since these simpler terms have a lower ordinal rank than $t$ itself, we obtain a contradiction.

The bulk of the proof will consist of a series of lemmas developing the technical machinery for tackling rules 3 and 6. First, a very general lemma about evaluation and the tracking of subterms:

**Lemma 23** Suppose that

\[\Gamma \vdash \langle C[\text{case } gpq \text{ of } (i \Rightarrow e_i)] \rangle = C'[\text{case } gp'q' \text{ of } (i \Rightarrow e'_i)],\]

where both sides are procedures or expressions, and the indicated occurrence of $g$ on the right hand side originates from the one on the left. Suppose too that $\Gamma, \vec{y}$ is the environment in force at the hole position in $C[-]$, and $\Gamma, \vec{y}'$ is that in
force at the hole in $C'[-]$. Then there exist a substitution $\Gamma \vdash \vec{y} \mapsto \vec{u}$ (where $\vec{y} \vdash \vec{u}$) and expressions $\Gamma, \vec{y} \vdash h_j$ such that

$$g_{\vec{p}'} \vec{q}' = \ll (gpq) \gg,$$

$$c_i' = \ll \text{case } c_i \text{ of } (j \Rightarrow h_j) \gg.$$

Proof. We formulate a suitable property of terms that is preserved under all individual reduction steps. Let $C[-], \vec{p}, \vec{q}, c_i$ and $\vec{y}$ be fixed as above, and suppose that

$$C^0[\text{case } gP^0Q^0 \text{ of } (i \Rightarrow E^0_i)] \rightsquigarrow C^1[\text{case } gP^1Q^1 \text{ of } (i \Rightarrow E^1_i)]$$

via a single reduction step, where the $g$ on the right originates from the one on the left, and moreover $C^0, P^0, Q^0, E^0_i$ enjoy the following property (here $\vec{y}^0$ denotes the list of local variables in scope at the position $C^0[-]$):

- There exist a substitution $\vec{y}^0 = [\vec{y} \mapsto \vec{u}^0]$ (where $\vec{y} \vdash \vec{u}^0$) and expressions $\Gamma, \vec{y} \vdash h_j^0$, such that

$$\ll gP^0Q^0 \gg = \ll (gpq)^{10} \gg,$$

$$\ll E^0_i \gg = \ll \text{case } c_i^{10} \text{ of } (j \Rightarrow h_j^0) \gg.$$

We claim that $C^1, P^1, Q^1, E^1_i$ enjoy the same property with respect to the list $\vec{y}^1$ of local variables in scope at $C^1[-]$: that is, there exist a substitution $\vec{y}^1 = [\vec{y} \mapsto \vec{u}^1]$ (where $\vec{y} \vdash \vec{u}^1$) and expressions $\Gamma, \vec{y} \vdash h_j^1$ for which the corresponding equations hold.

We show this by cases on the nature of the reduction step in question:

- If the subexpression $\text{case } gP^0Q^0 \text{ of } (i \Rightarrow E^0_i)$ is unaffected by the reduction (that is, $P^0 = P^1$, $Q^0 = Q^1$ and $E^0_i = E^1_i$ for all $i$), then the conclusion is immediate.

- If the reduction is internal to $P^0, Q^0$ or one of the $E^0_i$ (for example, if $P^0 \rightsquigarrow Q^0$ and $E^0_i = E^1_i$ for each $i$ then again the conclusion follows immediately, taking $h_j^i = h_j^{10}$ for all $j$.

- If the reduction is for a $\beta$-redex $(\lambda \vec{x}. E) \vec{R}$ where the indicated subexpression $\text{case } gP^0Q^0 \text{ of } (i \Rightarrow E^0_i)$ lies within $E$, then $P^1 = P^0[\vec{x} \mapsto \vec{R}]$ and similarly for $Q^1, E^1_i$, so the conclusion follows if we take $h_j^i = h_j^{10}$ for all $j$.

- If the reduction has the form

$$\text{case } (\text{case } gP^0Q^0 \text{ of } (i \Rightarrow E^0_i)) \text{ of } (j \Rightarrow F_j) \rightsquigarrow \text{case } gP^0Q^0 \text{ of } (i \Rightarrow \text{case } E^0_i \text{ of } (j \Rightarrow F_j))$$

then $P^1 = P^0, Q^1 = Q_0$ and $E^1_i = \text{case } E^0_i \text{ of } (j \Rightarrow F_j)$ for each $i$, so the conclusion follows if we take $h_j^i = h_j^{10}$ and $h_j^1 = \ll \text{case } h_j^0 \text{ of } (k \Rightarrow F_k) \gg.$
To complete the proof, in the situation of the lemma we will have a finite reduction sequence

\[ C[\textit{case } gpq \text{ of } (i \Rightarrow e_i)] \leadsto^* C''[\textit{case } gP'Q' \text{ of } (i \Rightarrow E'_i)] \]

where \( \ll P' \gg = p', \ll Q' \gg = q', \ll E'_i \gg = e'_i \) for each \( i \), \( \ll C''[-] \gg = C''[-] \), and there is a finite normal-form context \( t[-] \sqsubseteq C''[-] \) containing the hole in \( C''[-] \) such that \( t[-] \sqsubseteq C''[-] \). Since \( C, p, q, e_i \) themselves trivially satisfy the above invariant, by applying the above to each step of the reduction we see that \( C'', P', Q', E'_i \) also satisfy it with respect to some \( \dagger \) and \( h_j \), and this gives the properties of \( p', q', e'_i \) required by the lemma. \( \square \)

We now apply the above lemma to develop some more specialized machinery. In accordance with our earlier proof sketch, our intention is to show that under appropriate assumptions, a gremlin cannot be ‘assembled’ from separate ingredients that do not individually contain a gremlin—this will enable us to prove inductively that all terms in \( A_k \) are gremlin-free. As a specific manifestation of this idea, we will show (under certain conditions) that if a meta-term \( C[T] \) evaluates to a term containing a gremlin whose leading occurrence of \( g \) originates from \( T \), then the entire gremlin structure originates from \( T \), with no essential contribution coming from \( C[-] \).

We work in the setting of a two-part environment \( \Gamma, V \) with \( g \in \Gamma \) and \( \text{lv}(V) \leq k \). We shall also, at this point, adopt the standing assumption (which we will later show to be innocuous) that \( \Gamma \) contains no variables of level greater than \( k+2 \) (the level of \( g \)). This property will then be shared by all other environments we consider, since we will never refer to variables of level \( k+3 \) or higher. By the argument of Lemma 19, it is easy to see that no normal-form expression or procedure of level \( \leq k+1 \) in such an environment can contain a binding \( \lambda \vec{x} \) of level \( > k \).

In the situation of a meta-term \( C[T] \) as outlined above, the following technical lemma goes much of the way towards establishing that a gremlin is present in \( T \). (The remaining gap will be discussed after the proof of the lemma, and will be addressed by Lemma 25.) Our formulation here identifies three conditions that are required to hold, then introduces various pieces of local notation needed to articulate the conclusions.

**Lemma 24** Suppose \( \Gamma, V \vdash \ll C[T] \gg = t' \) satisfies the following:

1. \( C[T] \) contains no abstractions \( \lambda \vec{x} \) with \( \text{lv}(\vec{x}) > k \).

2. \( t' \) contains a gremlin \( c \) w.r.t. \( \Gamma \) and \( V \) at some position \( K'[-] \) (so that \( t' = K'[c] \)).

3. The leading occurrence of \( g \) in \( c \) originates from the head of a subterm \( d = \textit{case } gpq \text{ of } (i \Rightarrow e_i) \) at position \( K[-] \) within \( T \) (so that \( T = K[d] \)).

Let \( [\vec{v} \mapsto \vec{s}] \) be a suitable substitution of level \( \leq k \) covering \( V \) and the local variables for \( K'[-] \) such that \( \Gamma^- \vdash \vec{s} \) and \( \ll c[\vec{v} \mapsto \vec{s}] \gg \) is spinal. Then the following hold:
(i) The positions \( C[K[-]] \) and \( K[-] \) are exposed.

(ii) For a certain substitution \( \bar{v}' \mapsto \bar{s}' \) of level \( \leq k \), where \( \bar{v}' \) consists of the variables of \( V \) plus the local variables for \( C[K[-]] \), and \( \Gamma \vdash \bar{s}' \), the term \( \Gamma \vdash \ll \bar{d} \bar{v}' \mapsto \bar{s}' \gg \) is spinal, and its head occurrence of \( g \) originates from the head occurrence of \( g \) in \( d \).

(iii) If \( \ll c \bar{v} \mapsto \bar{s} \gg = \text{case } gFo \ (\cdots) \) then \( \ll d \bar{v}' \mapsto \bar{s}' \gg \) also has the form \( \text{case } gFo \ of \ (\cdots) \).

In the statement of this lemma, the enclosing context \( C[-] \) might seem to play very little role; however, this feature is worth maintaining for the sake of our formulation of Corollary 26 below.

**Proof (i)** Condition 1 of Definition 18 (for both \( C[K[-]] \) and \( K[-] \)) is given by the hypothesis that \( C[T] \) contains no bindings of level \( > k \). For condition 2, it will suffice to consider \( C[K[-]] \). Suppose for contradiction that the hole in \( C[K[-]] \) appears within a subterm \( P \) of level \( \leq k \) appearing as an argument in \( wP_v \), where \( w \in \Gamma \) and \( lv(w) > k \). Then within \( C[T] = C[K[d]] \), the relevant application \( gpq \) of the global variable \( g \) occurs within an application of the global variable \( w \). It is now easy to check that whenever \( C[T] \vdash T' \), every residual in \( T' \) of the relevant \( g \) in \( C[T] \) lies within a level \( \leq k \) argument in some application of \( w \). Hence the relevant occurrence of \( g \) in \( c \) falls within a level \( \leq k \) argument in some application of \( w \) in \( t' \). Since the position \( K'[c] \) of \( c \) within \( t' \) is exposed (because \( c \) is a gremlin within \( t' \)), this application of \( w \) must occur inside \( c \). But this implies that the relevant residual of \( g \) within \( \ll c \bar{v} \mapsto \bar{s} \gg \) (that is, the occurrence of \( g \) at the head of this spinal term) also falls within an application of \( w \), which it does not. This is a contradiction, so condition 2 of Definition 18 is established.

(ii),(iii) Consider the leading occurrence of \( g \) within \( c \); this occurs as the head of some expression \( d' = \text{case } gpq' \ of \ (i \Rightarrow e_i') \) at some position \( J[-] \). Thus \( \ll C[K[d]] \gg = K'[J[d']] \), where the head \( g \) in \( d' \) originates from that in \( d \). So by Lemma 23, for some substitution \( \bar{1} = \{\bar{y} \mapsto \bar{u}\} \) and some expressions \( \Gamma, V, \bar{y}, \bar{y}' \vdash h_j \), where \( \bar{y}, \bar{y}' \) are the local variables associated with \( C[K[-]] \) and \( K'[J[-]] \) respectively (so that \( X = \{\bar{y}\} \)) and \( \Gamma, V, \bar{y}' \vdash \bar{u} \), we have

\[
\begin{align*}
gp'q' &= \ll (gpq)^1 \gg, \\
e_i' &= \ll \text{case } e_i^1 \ of \ (j \Rightarrow h_j) \gg.
\end{align*}
\]

Recall too that, by assumption 1 on \( C[T] \), the variables \( \bar{y} \) all have level \( \leq k \).

Next, consider the evaluation of \( c \bar{v} \mapsto \bar{s} \) to a spinal term. Abbreviating \( \bar{v} \mapsto \bar{s} \) to \( \ast \), we have \( c^\ast = J^* [\text{case } gpq'^* q'^* \ of \ (i \Rightarrow e_i'^* \Rightarrow) \gg] \gg = \text{case } gFo \ of \ (\cdots) \)

for some suitable pre-spinal \( F \) and \( o \) as in the definition of spinal terms. Moreover, since \( J[-] \) and \( d' \) were chosen so that the head \( g \) of \( d' \) was the leading occurrence of \( c \), the indicated occurrence of \( g \) on the left-hand side here is the origin of the head \( g \) on the right-hand side. So by Lemma 23 again, for some
substitution \( + = [\vec{y}'\mapsto\vec{u}']\) and some expressions \( \Gamma \vdash f_j \), where \( \vec{y}' \) are the local variables associated with \( J^*[-] \) (or equivalently with \( J[-] \)) and \( \Gamma \vdash \vec{u}' \), we have
\[
gFo = \ll (gpq^*q'^*)^+ \gg .
\]
Putting all this together, we deduce that
\[
\ll \text{case (case gpq of (i \Rightarrow e_i))}^{+\ast+} \text{ of (j \Rightarrow h_j)}^{+\ast+} \gg = \ll \text{case (case gpq of (i \Rightarrow e_i))}^{+\ast+} \text{ of (j \Rightarrow h_j)}^{+\ast+} \gg = \ll \text{case (case gpq of (i \Rightarrow e_i))}^{+\ast+} \text{ of (j \Rightarrow h_j)}^{+\ast+} \gg = \text{case gFo of (\cdots)}
\]
which is spinal. It follows by Lemma 21 that \( d^{\ast+} = (\text{case gpq of (i \Rightarrow e_i))}^{+\ast+} \) is spinal, and we also see that \( (gpq)^{+\ast+} = gFo \).

It remains to see that the multiple substitution \( ^{+\ast+} \) is equivalent to a substitution of the form \( [\vec{v}' \mapsto \vec{s}'] \) as required by the lemma. Noting that \( \vec{y}' = X', \vec{y}'' \), we recall that
\[
\begin{align*}
^+ &= [\vec{y} \mapsto \vec{u}] \quad \text{where } \vec{y} \text{ covers } X \quad \text{and } \Gamma, V, X', \vec{y} \vdash \vec{u}, \\
\ast &= [\vec{v} \mapsto \vec{s}] \quad \text{where } \vec{v} \text{ covers } V, X' \quad \text{and } \Gamma^\ast \vdash \vec{s}, \\
\ast\ast &= [\vec{y}'' \mapsto \vec{u}'] \quad \text{where } \Gamma \vdash \vec{u}''.
\end{align*}
\]
It follows that we may define
\[
[\vec{v}' \mapsto \vec{s}'] = [\vec{y} \mapsto \vec{u}^+, \vec{v} \mapsto \vec{s}^-]
\]
where \( \vec{v}^- \mapsto \vec{s}^- \) denotes the restriction of \( \vec{v} \mapsto \vec{s} \) to \( V \). We will then have that \( \vec{v}' \) consists of the variables of \( V, X \) (which are all of level \( \leq k \)), and \( \Gamma \vdash \vec{s}' \). \( \square \)

We have thus shown that \( d \) occurs at an exposed position within \( T \) and can be specialized and evaluated to yield a spinal term via a substitution \( [\vec{v}' \mapsto \vec{s}'] \) of level \( \leq k \). If \( T \) were a normal form, this would mean that \( d \) was a gremlin within \( T \), were it not that the terms \( \vec{s}' \) exist in the environment \( \Gamma \) rather than \( \Gamma^- \). Thus, our analysis so far leaves open the possibility that the terms \( \vec{s}' \) contain occurrences of \( g \) (for instance) that contribute crucially to the spinal structure of \( [\vec{v}' \mapsto \vec{s}'] \). To complete the proof, we therefore need to show that we can replace \( \vec{s}' \) in this substitution by terms in \( \Gamma^- \), and still obtain a spinal term—thus, the gremlin is in essence already present in \( d \), rather than being assembled from ingredients in \( d \) and \( \vec{s}' \). This is achieved by the proof of the next lemma, which forms the most demanding part of our argument.

**Lemma 25** Suppose that \( \Gamma \vdash [\vec{v}' \mapsto \vec{s}'] \gg = t \) is spinal, where \( d = \text{case gpq of (i \Rightarrow e_i)} \), \( \Gamma, \vec{v}' \vdash d, \Gamma \vdash \vec{s}' \), \( \text{lv}(\vec{s}') \leq k \).

Then there are terms \( \Gamma^- \vdash \vec{s}'^+ \) such that \( [\vec{v}' \mapsto \vec{s}'] \gg \) is spinal.

**Proof** We begin with some informal intuition. The spinal term \( t \) will feature various local variables \( x \) of type \( k \), which occur within procedures \( o \) (where \( o^\circ \approx x^n \) for some \( o \)) that appear as arguments to \( g \). Since these variables are
local to \( t \), they do not appear in \( \Gamma \) and so originate from \( d \) rather than from \( s' \). Suppose, however, that one of the occurrences of \( g \) necessary for the spinal structure originated from some \( s'_i \) rather than \( d \). In order to form the application of this \( g \) to the relevant \( \phi^0 \approx x^0 \), the whole extension of \( x^0 \) would in essence need to be ‘passed in’ to \( s'_i \) when \( d \) and \( s' \) are combined. But this is impossible, since the arguments to \( s'_i \) are of level \( < k \), so by Theorem 6, the whole of \( x^0 \) cannot be funnelled through them. Informally, the interface between \( d \) and \( s' \) is too narrow (in terms of type levels) for the necessary communication to occur.

We now proceed to the formal proof. Let \( \uparrow = [\vec{v}' \mapsto \vec{s}' \cdot \vec{u}] \). Consider the subterm \( p = \lambda x'.e \) within \( d \). Then \( \ll e \gg \) is some spinal term

\[
\Gamma, x' \vdash e' = E[c], \text{ where } c = \text{case } gFo \text{ of } (\cdots).
\]

We wish to show, first, that the head \( g \) here (that is, the second occurrence of \( g \) on the spine of \( t \)) originates from \( e \) rather than from \( \vec{s}' \). We will later show that the same argument can be repeated for lower spinal occurrences of \( g \).

Suppose for contradiction that the head \( g \) of \( c \) originates from some substituted occurrence of an \( s'_i \) within \( e^1 \), say as indicated by \( e^1 = D[s'_i] \) and \( s'_i = L[d'] \) where \( \Gamma, x' \vdash D[-], \Gamma \vdash s'_i, \Gamma, X \vdash d' \) and \( d' = \text{case } gp'q' \text{ of } (i \Rightarrow e'_i) \). (Here \( X \) is the local variable environment for \( L[-] \).

Then

\[
\Gamma, x' \vdash \ll D[L[d']] \gg = e',
\]

where \( d' = \text{case } gp'q' \text{ of } (i \Rightarrow e'_i), e' = E[\text{case } gFo \text{ of } (\cdots)], \) and the head occurrence of \( g \) in \( d' \) is the origin of the identified occurrence of \( g \) in \( e' \). Our first objective will be to apply Lemmas 24 and 19 to show that a spinal term may be obtained from \( d' \) via a substitution of relatively low type—this will provide the bottleneck through which \( x'^0 \) is unable to pass.

Since \( c \) is spinal, it is clearly a gremlin within \( e' \) w.r.t. \( \Gamma, x' \) and \( \emptyset \), occurring at the exposed normal-form position \( E[-] \). Moreover, because \( \Gamma, x' \) contains no variables of level \( > k + 2 \) by assumption, and \( \Gamma, x' \vdash e \) and \( \Gamma \vdash s' \) are normal forms, \( e \) and hence \( e^1 \) involve no bound variables of level \( > k \). We are thus in the situation of Lemma 24, taking \( \Gamma, V, C, T, t', c, K, K' \) and \( [\vec{v} \mapsto \vec{s} \cdot \vec{u}] \) of the lemma to be respectively \( (\Gamma, x'), \emptyset, D, s'_i, e', c, L, E \) and \( \emptyset \). We conclude, first, that the position \( D[L[-]] \) is exposed. Moreover, \( D[L[d']] = e^1 \) contains no \( \beta \)-redexes except those introduced by the substitution \( \uparrow \) (for which the operators are of level \( \leq k \)). Since \( e^1 \) is an expression, we therefore conclude by Lemma 19 that the local variables associated with \( D[L[-]] \) (call them \( \vec{y} \)) are all of level \( < k \).

We also conclude from Lemma 24 that there is a substitution \( [\vec{v} \mapsto \vec{s}] \) such that \( \ll d'[\vec{v} \mapsto \vec{s}] \gg \) is spinal, and indeed of the form \( \text{case } gFo \text{ of } (\cdots) \). Moreover, since the ‘\( \vec{s} \)’ of Lemma 24 is empty in this case, the substitution variables \( \vec{v} \) are precisely the local variables \( \vec{y} \) associated with \( D[L[-]] \). We henceforth write this substitution \( [\vec{v} \mapsto \vec{s}] \) as \( [\vec{y} \mapsto \vec{u}] \), where \( \Gamma, x' \vdash \vec{u} \) and \( \vec{y}, \vec{u} \) are of level \( < k \).

Summarizing, we have that

\[
\ll d'[\vec{y} \mapsto \vec{u}] \gg = \text{case } gFo \text{ of } (\cdots),
\]

31
where \( d' = \text{case } gp'q' \text{ of } (i \Rightarrow e'_i), \) the \( \vec{u} \) have level \(< k, \) and the right-hand side is spinal. From this we may read off that

\[
\ll q'[\vec{y} \mapsto \vec{u}] \gg = o.
\]

Moreover, \( x' \) does not occur in \( q' \) since \( \Gamma, X \vdash q' \) (recall that the application \( gp'q' \) is assumed to come from \( s_i' \) rather than \( e' \); thus each occurrence of \( x' \) in \( o \) above must originate from some \( u_j \), with some type \( \rho_j \) of level \(< k). \) Note that \( \vec{u} \) may consist of infinitely many \( u_j \) in general.

We may exploit this to obtain a retraction \( K < \Pi_j \rho_j \), contradicting Theorem 6. Specifically, we have procedures \( \Gamma \) which contradict Theorem 6 as claimed.

To greater depth, suppose that this originating occurrence is as indicated by \( d \) is spinal. From this we may read off that

\[
\text{Moreover, } x' \vdash \ll q'[\vec{y} \mapsto \vec{u}] \gg \Rightarrow \ll o \gg \approx x'' \tilde{\varepsilon},
\]

which contradicts Theorem 6 as claimed.

We have thus shown, as promised, that the head \( g \) in \( c \) (a subterm of \( E[c] = \ll e' \gg \)) originates from \( e \) rather than \( \dagger. \) In order to continue our analysis to greater depth, suppose that this originating occurrence is as indicated by \( e = C[d'] \) where \( d' = \text{case } gp'q' \text{ of } (i \Rightarrow e'_i) \). Then

\[
\Gamma, x' \vdash (\lambda \vec{v}' . C[d'])s'' \gg = e'
\]
is spinal, where the displayed occurrence of \( g \) is the origin of the head \( g \) in \( t''. \)

(Since \( \vec{v}' \) may be infinite, extended NSP terms may enter into the picture here.) Moreover, we see that \( (\lambda \vec{v}' . C[d'])s'' \) involves no bound variables of order \( \geq k, \) since neither \( C[d'] = e \) nor \( s'' \) do so (see the remark preceding Lemma 24). So applying Lemma 24 again, there is a substitution \( [\vec{v}'' \mapsto s''] \) (with \( lv(\vec{v}'') \leq k, \) \( \Gamma, x', \vec{v}'' \vdash d' \) and \( \Gamma, x' + s'' \)) such that \( \Gamma, x' \vdash [d''[\vec{v}'' \mapsto s'']] \gg \) is spinal, and indeed of the form \( \text{case } gpFo \text{ of } (\cdot \cdot \cdot) \). It follows that \( \ll p''[\vec{v}'' \mapsto s''] \gg = F. \)

We are now back precisely where we started, in the sense that \( d', \vec{v}'', s''' \) satisfy the hypotheses of the lemma, with \( \Gamma, x' \) now playing the role of \( \Gamma \). Applying the same argument again, we conclude that the designated \( g \) in the pre-spinal term \( \ll p''[\vec{v}'' \mapsto s''] \gg = F \) (which coincides with a subterm of the original \( t \) headed by the third spinal occurrence of \( g \)) comes from the one headning a subterm \( d''' = \text{case } gp''q'' \text{ of } (i \Rightarrow e'_i) \) within \( p'' \), itself a part of \( d' \), which is a subterm of \( d. \)

One now naturally expects that the occurrence of \( g \) within \( d \) that we have identified is indeed the origin of the third spinal \( g \) in \( \ll d'' \gg = t; \) however, a little bureaucracy is needed to confirm this. First note that the substitution \( [\vec{v}'' \mapsto s''] \) in effect arose via Lemma 24(ii) from the evaluation of \( e'' \) to a spinal term \( \text{case } gpFo \text{ of } (\cdot \cdot \cdot). \) In the notation of the proof of Lemma 24(ii), we see that \( [\vec{v}'' \mapsto s''] \) is constructed as a certain substitution \( [\vec{y} \mapsto \vec{u}^{++}, \vec{v}^- \mapsto \vec{s}^-]; \) however, in this instance of the lemma, the set \( V \) is empty and the term \( t \) is actually spinal; thus \( K'[\cdot] \) and \( J[\cdot] \) are both \( \cdot \) and this substitution simplifies
to \([\bar{y} \mapsto \bar{u}]\). But this substitution itself arose via Lemma 23 from the evaluation of \(e^1\) to a spinal term; this means that it is possible to reduce \(e^1\) to a term \(d^1\) of the form \(d'[\bar{v}'' \mapsto \bar{s}''']\) or \textbf{case} \(d'[\bar{v}'' \mapsto \bar{s}''']\) of \((\cdots)\), where the subterm \(d'\) within \(e\) is the origin of the instantiated subterm \(d''\) within \(d^1\). Thus, \(d^1\) itself can be reduced to a term \textbf{case} \(g(\lambda x'.d^1)q^1\) of \((\cdots)\), with the identified occurrence of \(g\) in \(d\) giving rise to its obvious correspondent in \(d^1\). But the latter falls within \(p'[\bar{v}'' \mapsto \bar{s}''']\), and we have already seen that it is the originator of the designated \(g\) in \(F\). Since \(d' = \textbf{case} \ gp'q'\) of \((\cdots)\), it follows that the identified occurrence of \(g\) in \(d^1\) yields the second spinal occurrence in \textbf{case} \(gFo\) of \((\cdots)\). Putting all this together, we see that the third spinal \(g\) in \(t\) does indeed originate from \(d\) in the expected way.

Continuing inductively in this fashion, we now see that all spinal occurrences of \(g\) in our original spinal term \(\ll [d[\bar{v}'' \mapsto \bar{s}']] \gg\) come from \(d\) rather than \(\bar{s}'\).

Since there are no occurrences of \(g\) in \(\ll [d'[\bar{v}'' \mapsto \bar{s}'''] \gg\) other than these spinal ones, it is easy to see that any occurrences of \(g\) in \(\bar{s}'\) play no role in the evaluation of \(\ll d'[\bar{v}'' \mapsto \bar{s}'] \gg\) to a spinal term, so that if we define \(s_i'^{-1} = \ll s_i'[g \mapsto \bot] \gg\) for each \(i\), we will have that \(\Gamma^{-1} \vdash \bar{s}'^{-1}\) and \(\ll d'[\bar{v}'' \mapsto \bar{s}'''] \gg\) is spinal. This completes the proof. \(\square\)

Combining the previous two lemmas in the obvious way yields the following:

**Corollary 26** In the situation of Lemma 24, if \(T\) is a normal form, then \(d\) is a gremlin within \(T\) with respect to \(\Gamma\) and \((V, X)\), where \(X\) is the local variable environment for \(C[\cdot]\).

**Proof** Immediate from Lemma 24, Lemma 25 and the intervening discussion. \(\square\)

This corollary will be used below to handle terms constructed via rule 3 of Definition 9. To handle terms constructed via rule 6, a slightly different way of combining Lemmas 24 and 25 will be needed.

We are now ready to prove the main result of this section:

**Theorem 27** No normal-form term \(\Gamma, V \vdash t \in A_k\) (where \(\text{lv}(V) \leq k\)) can contain a gremlin.

**Proof** Suppose for contradiction that \(\Gamma, V \vdash t\) were a term of minimal ordinal rank \(\alpha\) in the inductive generation of \(A_k\) such that \(t\) contains a gremlin w.r.t. \(\Gamma, V\), and let \(c, K[\cdot], X\) and \([\bar{v} \mapsto \bar{s}]\) be the relevant data as in Definition 22. As an easy first step, we may assume without loss of generality that \(\Gamma\) contains no variables of level \(> k + 2\) (as postulated before Lemma 24), since it is an easy exercise to show that for any such variables \(w\), we may replace all expressions \textbf{case} \(w\bar{p}\) of \((\cdots)\) within \(t\) by \(\bot\) to obtain another term of rank at most \(\alpha\) containing a gremlin. As noted earlier, this will mean that in \(\Gamma, V\) and all other contexts that we shall consider, a normal-form term of type level \(\leq k + 1\) will not contain any bound variables of level \(> k\).

We consider in turn the possibilities for the last rule from Definition 9 applied in the construction of \(\Gamma, V \vdash t \in A_k\). This cannot be rule 1 because \(n, \bot\) contain
no gremlins, and cannot be rule 5 because \( t \) is normal. Moreover, rules 2 and 4 can be eliminated as follows:

- Suppose \( t \) is constructed by rule 2, say as \( \lambda \vec{x}.c \); then the \( \vec{x} \) are of level \( \leq k \) since \( t \) contains a gremlin \( c \) at an exposed position. Since \( c \) is an expression, it appears within \( e \), and it is easy to check that \( c \) is a gremlin in \( e \) with respect to \( \Gamma \) and \((V, \vec{x})\). We thus have that \( \Gamma, V, \vec{x} \vdash e \) has rank \(< \alpha \) and contains a gremlin, contradicting the minimality of \( \alpha \).

- Suppose \( t \) is constructed via rule 4, say as \textbf{case} \( a \text{ of } (i \Rightarrow e_i) \). If a gremlin for \( t \) appears within \( a \) or some \( e_i \), then as before, this subterm contains a gremlin and is of lower rank. If \( t \) itself is the gremlin, then for a suitable \([\vec{v} \mapsto \vec{s}]\) covering \( V \) we have that \( \ll \textbf{case} \ a[\vec{v} \mapsto \vec{s}] \text{ of } (i \Rightarrow e_i[\vec{v} \mapsto \vec{s}]) \gg \) is a spinal expression. So by Lemma 21, either \( \ll a[\vec{v} \mapsto \vec{s}] \gg \) is a numeral \( n \), in which case \( e_n \) is a gremlin within itself, or \( \ll a[\vec{v} \mapsto \vec{s}] \gg \) is spinal, in which case \( a \) is a gremlin within itself. Either way, the minimality of \( \alpha \) is contradicted.

It remains to consider rules 3 and 6, and for these, the machinery of Lemmas 24 and 25 comes into play. Suppose first that \( t \) is constructed by rule 3, say as \( x_{p_0} \ldots p_{r-1} \). If a gremlin \( c \) for \( t \) exists within some \( p_i \), then clearly \( c \) is also a gremlin in \( p_i \) which is of rank \(< \alpha \), a contradiction. Otherwise, \( t = x\vec{p} \) is a gremlin within itself, say with \( \Gamma \vdash t' = \ll t[\vec{v} \mapsto \vec{s}] \gg \) a spinal term (where \( \vec{v} \) covers \( V \) and \( \Gamma \vdash \vec{s} \)). Here \( x \notin \Gamma \), since otherwise \( t' \) would have head variable \( x \neq g \). There are thus two possibilities:

- \( x = g \), \( r = 2 \) and \( \ll p_0[\vec{v} \mapsto \vec{s}] \gg = F \) is pre-spinal. Suppose \( F = \lambda \vec{x}'.E'[c] \), with \( E'[\ldots] \) a normal-form exposed position and \( c \) spinal. Then the head \( g \) of \( c \) originates from \( p_0 \), say as indicated by \( p_0 = K[d] \) where \( d = \textbf{case} \ gp'q' \text{ of } (\cdots) \). Setting \( C[-] = (\lambda \vec{v}. -)\vec{s} \), we now have
  \[
  \Gamma \vdash \ll C[K[d]] \gg = \ll (\lambda \vec{v}.p_0)\vec{s} \gg = F = (\lambda \vec{x}'.E'[c]) .
  \]
  (Again, extended terms may enter the picture here.) Since \( c \) is spinal and \( (\lambda \vec{x}'.E[-]) \) is exposed, \( c \) is a gremlin within \( F \). We claim that this is an instance of the situation of Lemma 24, where we take \( C[-] = C[-], T = p_0, c = c, K' = (\lambda \vec{x}'.E[-]), K[-] = K[-], d = d \) and \( V = \emptyset \). For condition 1, we require that \( C[p_0] \) contains no variable bindings of level \( > k \); but this holds because (in the present notation) \( \vec{v} \) is of level \( \leq k \) and \( \Gamma, V \vdash \vec{p} \) and \( \Gamma \vdash \vec{s} \) are normal forms of level \( \leq k \). Condition 2 is already established above, and condition 3 holds by choice of \( d \) and \( K[-] \). We may therefore apply Corollary 26 to conclude that the subterm \( d \) is a gremlin within \( p_0 \), which gives a contradiction.

- \( x = v_i \in V \), and the head \( g \) in \( t' = \ll (x\vec{p})[\vec{v} \mapsto \vec{s}] \gg \) originates from some \( p_i \), say as indicated by \( p_i = K[d] \) where \( d = \textbf{case} \ gp'q' \text{ of } (i \Rightarrow e_i) \). We may also set \( C[-] = (\lambda \vec{v}. x_{p_0} \ldots p_{i-1}[\ldots]p_{i+1} \ldots p_{r-1})\vec{s} \), so that
  \[
  \Gamma \vdash \ll C[K[d]] \gg = \ll (\lambda \vec{v}. x\vec{p})\vec{s} \gg = t' .
  \]
We claim that this is an instance of the situation of Lemma 24, where we take \( C[-] = C[-], T = p_i, c = t', K'[-] = -, K[-] = K[-], d = d \), and take the \( V \) and \( X' \) of the lemma to be empty, so that the \([\vec{v} \mapsto \vec{s}]\) of the lemma is also empty. For condition 1 of the lemma, we require that \( C[p_i] \) contains no variable bindings of level \( > k \), and this can be verified by inspection just as in the previous case. For conditions 2 and 3, it suffices to recall that \( t' \) is spinal and its head \( g \) originates from \( d \) by choice of \( d \) of \( K[-] \). We may therefore conclude by Corollary 26 that the subterm \( d \) is a gremlin within \( \Gamma, V \vdash p_i \), again giving a contradiction.

Finally, suppose that \( t \) is constructed via rule 6 (the plugging rule): say as \( t = \ll \Pi(\Gamma, V), z(e, \xi) \gg \) where \( \Gamma, V, Z \vdash e \in \mathcal{A}_k \) and each \( \Gamma, V, Z \vdash \xi(z) \in \mathcal{A}_k \) have rank \( < \alpha \). For later convenience, to each \( z_i \in Z \) let us associate the procedure \( \Gamma, V \vdash r_i = \ll \Pi(\Gamma, V), z(\xi(z_i), \xi) \gg \): it is then clear from the definition of plugging and the evaluation theorem that \( t = \ll e[\vec{z} \mapsto \vec{r}] \gg \) and that \( r_i = \ll \xi(z_i)[\vec{z} \mapsto \vec{r}] \gg \) for each \( i \). We also have by hypothesis that \( c \) is a gremlin in \( t \) at the exposed position \( K[-] \), and in particular that \( \Gamma \vdash \ll c[\vec{v} \mapsto \vec{s}] \gg \) is a spinal case expression, where \( \vec{v} \) covers \( V \) and \( \Gamma \vdash \vec{s} \). We shall focus on the head occurrence of \( g \) within \( \ll c[\vec{v} \mapsto \vec{s}] \gg \).

Since the \( \vec{s} \) cannot contain \( g \) free, this occurrence of \( g \) must originate from \( c \), which is a subterm of \( t \), and hence ultimately from some normal-form term fragment \( \Gamma, V, Z \vdash t_0 \) involved in the above plugging (either \( e \) or some \( \xi(z) \)). Suppose that this originating occurrence of \( g \) in \( t_0 \) is as indicated by

\[
t_0 = L[d], \quad d = \text{case gpq of } (i \mapsto e_i). \]

Writing \( * \) for \([\vec{z} \mapsto \vec{r}]\), we have \( \ll \Pi(t_0, \xi) \gg = \ll t_0^* \gg \); and if \( C[-] \) is the context encapsulating the remainder of the plugging \( \Pi(e, \xi) \) then we may write

\[
\Gamma, V \vdash t = \ll C[\Pi(t_0, \xi)] \gg = \ll C[t_0^*] \gg = \ll C[L'[\ll d^* \gg]] \gg ,
\]

where

\[
\ll d^* \gg = \text{case } g \ll p^* \gg \ll q^* \gg \text{ of } (i \mapsto \ll e_i^* \gg) .
\]

Again, we claim that we are in the situation of Lemma 24: the \( C, T, t', d \) and \( K \) of the lemma are \( C, t_0^*, t, \ll d^* \gg, L^* \) respectively, and the gremlin \( c \) and position \( K' \) of the lemma are \( c, K \) respectively. Here, condition 1 of the lemma holds because \( C[t_0^*] \) is constructed by substitution from normal-form terms of level \( \leq k \), and conditions 2 and 3 are immediate given the present setup.

By Lemma 24, we may therefore conclude that \( L^* \) is exposed, and that for a suitable substitution \([\vec{v}' \mapsto \vec{s}]\) (where \( \Gamma \vdash \vec{s} \)), \( \Gamma \vdash \ll \ll d^* \gg [\vec{v}' \mapsto \vec{s}] \gg \gg \) is spinal. We may rephrase this as saying that \( \ll d[\vec{v}'' \mapsto \vec{s}'' \gg \gg \) is spinal, where

\[
[\vec{v}'' \mapsto \vec{s}''] = [\vec{v}' \mapsto \vec{s}', \vec{z} \mapsto \ll \vec{v}'[\vec{v}' \mapsto \vec{s}] \gg] ,
\]

so that \( \Gamma \vdash \vec{s}'' \) and \( lv(\vec{s}'') \leq k \). We are therefore in the situation of Lemma 25, and so may replace \( \Gamma \vdash \vec{s}'' \) by some \( \Gamma^- \vdash \vec{s}''^- \) such that \( \ll d[\vec{v}'' \mapsto \vec{s}''^- \gg \gg \) is spinal. Finally, the fact that \( L^* \) is exposed clearly implies that \( L \) is exposed;
but $t_0 = L[d]$, so $d$ is a gremlin within $t_0$ with respect to $\Gamma$ and $V, Z$. Once again, this contradicts the minimality of $\alpha$, since $\Gamma, V, Z \vdash t_0 \in A_k$ has rank $< \alpha$. This completes the proof. $\square$

Since $Y_{k+1} \in SP^0$ trivially contains a gremlin, and likewise $Y_{0\rightarrow(k+1)}$ contains a gremlin in the modified sense, we may conclude that $Y_{k+1}, Y_{0\rightarrow(k+1)} \not\in A_k$. Since by Theorem 17 every $PCF^{k^2}_k$-definable procedure lives in $A_k$, this establishes Theorem 5.

5 The extensional model

We conjecture that if $\tau = k + 1$ then every closed procedure $Z \approx Y_{\tau} \in SP^0$ must contain a gremlin; it would follow easily from this that the fixed point operator $Y_{k+1}$ within $SF$ was not definable in $PCF^\Omega_k$. However, the syntactic analysis of such procedures $Z$ appears to present considerable technical difficulties; and even in the case of $\tau = 0 \rightarrow (k + 1)$, it seems easiest to consider not $Y_{\tau}$ itself but a certain functional $\Phi$ that is readily definable from it.

Specifically, within $PCF_{k+1}$, let us define

$$\Phi : (0 \rightarrow (k + 1) \rightarrow (k + 1)) \rightarrow (0 \rightarrow (k + 1))$$

$$\Phi g = Y_{0\rightarrow(k+1)} (\lambda f. \lambda n. gn (f(suc n))) ,$$

so that informally

$$\Phi gn = g(n, g(n + 1, g(n + 2, \cdots ))) .$$

(This generalizes the definition of the functional $F$ mentioned at the end of Section 2.) For each $n \in \mathbb{N}$, let $\Phi_n[g] = \Phi g \bar{n} : k + 1$, and let $p_n$ denote the canonical NSP for $\Phi_n[g]$ (that is, the one arising from the above $PCF$ definition via the standard interpretation in $SP^0$). These procedures may be explicitly defined coinductively by:

$$g \vdash p_n = \lambda x^k. \text{case } g (\lambda n) p_{n+1} x^\eta \text{ of } (i \Rightarrow i) .$$

By a syntactic analysis of the possible forms of procedures $q \approx p_n$, we will show that any such $q$ is necessarily spinal. (For the purpose of this section, we allow ourselves to say that $\lambda x. e$ is spinal if $e$ is.) Here we have in mind the modified notion of spinal term that is applicable to terms involving a global variable $g : 0 \rightarrow (k + 1) \rightarrow (k + 1)$ (see the explanation following Definition 20). Using Theorem 27 (understood relative to this modified setting), it will then be easy to conclude that within $SF$, the element $[\lambda g. \Phi_0[g]]$, and hence $Y_{0\rightarrow(k+1)}$ itself, is not $PCF^\Omega_k$ definable in $SF$.

As a prelude to our analysis of the procedures $q$, we present a theorem saying that no functional in $SF(k \rightarrow k)$ can extensionally ‘improve on’ the identity function. For the purpose of our main proofs, this result could in principle be bypassed, although it is of some interest in its own right, and its use leads to a somewhat more complete and satisfying picture. Recall that $\preceq$ denotes the
extensional order on SF, as well as the associated preorder on SP₀. Within SF,
we will write \( f \prec f' \) to mean \( f \preceq f' \) but \( f \neq f' \); we also write \( f \parallel f' \) to mean
that \( f, f' \) have no upper bound with respect to \( \preceq \).

**Theorem 28** (i) If \( f \in SF^{\text{fin}}(k) \) and \( f \prec f' \), then there exists \( f'' \parallel f' \) with
\( f \prec f'' \).

(ii) If \( \Phi \in SF(k \to k) \) and \( \Phi \succeq \text{id} \), there can be no \( f \in SF(k) \) with \( \Phi(f) > f \);
hence \( \Phi = \text{id} \).

**Proof** Both statements are trivial for \( k = 0, 1 \), so we will assume \( k \geq 2 \).

(i) It clearly suffices to show this for \( f' \) finite. So suppose \( f \prec f' \) in \( SF^{\text{fin}}(k) \);
and take \( g \in SF^{\text{fin}}(k - 1) \) \( \preceq \)-minimal such that \( f(g) = \bot \) but \( f'(g) \downarrow \); say \( f'(g) = n \in \mathbb{N} \). (Note that a basis for \( f' \) will contain at least one such \( g \).) Take \( p, p', q \in SP^{0, \text{fin}} \) representing \( f, f', g \) respectively; we may assume that \( p, p', q \)
are ‘pruned’ so that every subtree containing no leaves \( m \in \mathbb{N} \) must itself be a leaf \( \bot \).

**Case 1:** \( g(\bot^{k-2}) = a \in \mathbb{N} \). In this case, we may suppose that \( q = \lambda x.a \).
Consider the computation of \( p \cdot q \). Since all calls to \( q \) trivially evaluate to \( a \),
this computation follows the rightward path through \( p \) consisting of branches
\( a \Rightarrow \cdots \). But since \( p \) is finite and \( p \cdot q = \bot \) since \( f(g) = \bot \), this path must end
in a leaf occurrence of \( \bot \) anywhere within \( p \). Now extend \( p \) to a procedure \( p'' \) by replacing
this leaf occurrence by some \( n' \neq n \); then clearly \( p'' \cdot q = n' \). Taking \( f'' \) to be
the function represented by \( p'' \), we then have \( f \preceq f'' \) and \( f''(g) = n' \neq f'(g) \),
so \( f'' \parallel f' \) (whence \( f'' \neq f \)).

**Case 2:** \( g(\bot^{k-2}) = \bot \). Take \( N \) larger than all numbers appearing in \( p, p', q \);
in particular, \( N > n \). Define \( q' \supseteq q \) as follows: if \( q = \lambda x.\bot \), take \( q' = \lambda x.N \),
otherwise obtain \( q' \) from \( q \) by replacing each case branch \( j \Rightarrow \bot \) anywhere within
\( q \) by \( j \Rightarrow N \) whenever \( j \leq N \). Likewise, extend \( p \) to \( p'' \) in the same way.

Now consider the computation of \( p \cdot q \). Since \( p, q \) are finite and \( f(g) = \bot \), this
evaluates to an occurrence of \( \bot \) which originates from \( p \) or \( q \). Since no numbers
\( > N \) ever arise in the computation, this occurrence of \( \bot \) will have been replaced
by \( N \) in \( p'' \) or \( q' \). We now claim that \( p'' \cdot q' = N \). Informally, this is because
the abovementioned occurrence of \( N \) will be propagated to the top level by the
case branches \( N \Rightarrow N \) within both \( p'' \) and \( q' \).

More formally, let us define the set of meta-expressions led by \( N \) inductively
as follows:

- \( N \) is led by \( N \),
- if \( E \) is led by \( N \), then so is \textbf{case} \( E \) of \( (i \Rightarrow D_1) \).

We now say that an NSP meta-term \( T \) is \textit{saturated at} \( N \) if every \textbf{case} subterm
within \( T \) has a branch \( N \Rightarrow E \) where \( E \) is led by \( N \). Clearly \( p''q' \) is saturated
at \( N \), and it is easy to check that the terms saturated at \( N \) are closed under
head reductions. Furthermore, the previous discussion makes clear that \( p''q' \)
head-reduces in finitely many steps to a meta-term \( T \) with an occurrence of \( N \)
in head position, so that \( T \) is itself led by \( N \), and an easy induction on term
size shows that every finite such meta-term that is saturated at \( N \) evaluates to \( N \) itself. This shows that \( p'' \cdot q' = N \).

To conclude, let \( g', f'' \) be the functions represented by \( q', p'' \) respectively, so that \( g \leq g' \) and \( f \leq f'' \). Then \( f'(g') = n \), but \( p'' \cdot q = N \) so \( f''(g') = N \), whence \( f'' \not\approx f' \), and as before this implies \( f'' \not\approx f \).

(ii) Suppose \( \Phi \succeq id \) and \( \Phi(f) \not\succ f \). Then by (i), we may take \( f'' \succ f \) such that \( f'' \not\approx \Phi(f) \). But this is impossible because \( \Phi(f'') \succeq f'' \) and \( \Phi(f'') \succeq \Phi(f) \).

Thus \( \Phi = id \). \( \square \)

Returning to our main task, we now present our syntactic analysis of a procedure \( q \) based on the premise that it has the same extensional behaviour as \( p_n \). We recall that

\[
g \vdash p_n = \lambda x^k. \text{case } g(\lambda.n) p_{n+1} x^n \text{ of } (i \Rightarrow i) : k + 1,
\]

where \( g : \rho = 0 \rightarrow (k + 1) \rightarrow (k + 1) \).

If \( n \leq n' \), we define the masking operator \( \mu_{n,n'} \) to be the term

\[
\mu_{n,n'} = \lambda g^n. \text{case } i \text{ of } (n \Rightarrow 0) \cdots (n' \Rightarrow 0) \Rightarrow gizx.
\]

We write \( \mu_{n,n} \) simply as \( \mu_n \). Clearly \( \mu_n(\mu_{n+1}(\cdots(\mu_{n'}(g)\cdots))) = \mu_{n,n'}g \).

Contrary to our usual convention, we will here use the uppercase letters \( G, X \) to range over normal-form procedures that may be substituted for \( g, x \) respectively. We say that \( G \in \text{SP}^0(\rho) \) is trivial at \( n \) if \( G(\lambda.n) \not\Rightarrow \perp \) for some \( m \in N \). Note that \( \mu_{n,n}G \) is trivial at each of \( n, \ldots, n' \).

**Lemma 29** Suppose \( q \succeq p_n \). Then \( q \) has the form \( \lambda x^k. \text{case garo of } (\cdots) \), where:

1. \( a[g \rightarrow G, x \rightarrow X] \approx \lambda n \) for any \( X \in \text{SP}^0(k) \) and \( G \in \text{SP}^0(\rho) \) such that \( G \) is trivial at \( n \),
2. \( o[Gi] \approx x^n \) whenever \( G \) is trivial at \( n \),
3. \( r[g \rightarrow \mu_n g, x \rightarrow X] \succeq p_{n+1} \) for any \( X \).

**Proof** Suppose \( q = \lambda x^k.e \). Clearly \( e \) is not constant since \( q \succeq p_n \); and if \( e \) had head variable \( x \), we would have \( q[g \rightarrow \lambdaizx.0](\lambda w.\perp) = \perp \), whereas \( p_n[g \rightarrow \lambdaizx.0](\lambda w.\perp) = 0 \). So \( e \) has the form \( \text{case garo of } (\cdots) \).

For claim 1, suppose we have \( X, G \) with \( G(\lambda.n) \not\Rightarrow \perp \Rightarrow \perp \) in \( N \), and define

\[
G' = \lambdaizx. \text{case } i \text{ of } (n \Rightarrow Gizz | - \Rightarrow \perp).
\]

Then \( G' \preceq G \), so if \( a[G, X] \not\preceq \lambda.n \) then \( a[G', X] \not\preceq \lambda.n \), so that \( G'a^*r^*o^* \not\Rightarrow \perp \) where \( * = [G', X] \); thus \( q[G']X \not\Rightarrow \perp \). On the other hand, we have

\[
p_n[G']X \succeq \text{case } G'(\lambda.n)p_{n+1}^*X \text{ of } (i \Rightarrow i) \approx G(\lambda.n)p_{n+1}^*X \approx m,
\]

contradicting \( q \succeq p_n \). Thus \( a[G, X] \succeq \lambda.n \).
For claim 2, it will suffice by Theorem 28 to show that \( o[G] \succeq x^n \) when \( G(\lambda n)\bot \bot \approx m \in \mathbb{N} \). Suppose not; then we may take \( X \in \text{SP}^0(k) \) and \( u \in \text{SP}^0(k-1) \) such that \( Xu \approx l \in \mathbb{N} \) but \( o[G, Xu] \neq l \). Now define

\[
G' = \lambda i.z.x. \text{case } xu \text{ of } (l \Rightarrow Gixz : - \Rightarrow \bot).
\]

Then again \( G' \preceq G \), so \( o^*u \neq l \) where \( ^* = [G', X] \); hence \( G'a^*r^*o^* \approx \bot \) and so \( q[G']X \approx \bot \). On the other hand, we have

\[
p_n[G']X \approx \text{ case } G'(\lambda n)p_{n+1}X \text{ of } (i \Rightarrow i)
\]

\[
\approx \text{ case } Xu \text{ of } (l \Rightarrow \text{ case } G(\lambda n)p_{n+1}X \text{ of } (i \Rightarrow i)) \approx m,
\]

contradicting \( q \approx p_n \). Thus \( o[G] \approx x^n \).

For claim 3, suppose that \( p_{n+1}[G]X' \approx l \) for some \( G \in \text{SP}^0(\rho) \) and \( X' \in \text{SP}^0(k) \). We wish to show that \( r[\mu_n G, X]X' \approx l \) for any \( X \). Suppose not, and consider

\[
G' = \lambda i.z.x. \text{ case } i \text{ of } (n \Rightarrow \text{ case } zX' \text{ of } (l \Rightarrow 0) \Rightarrow \bot) \Rightarrow Gixz \cdot
\]

Then \( G' \preceq \mu_n G \), so \( r[G', X]X' \neq l \). Moreover, by claim 1 we have that \( a[\mu_n G, X] \approx \lambda n, a[\mu_n G, X] \approx \bot \) or \( \lambda n \). In either case, we see that \( G'a^*r^*o^* \approx \bot \), where \( ^* = [G', X], \) so that \( q[G']X \approx \bot \). On the other hand, we have

\[
p_n[G']X \approx \text{ case } G'(\lambda n)p_{n+1}X \text{ of } (i \Rightarrow i)
\]

\[
\approx \text{ case } p_{n+1}X' \text{ of } (l \Rightarrow 0).
\]

Since \( p_{n+1} \) does not contain \( x \) free, we have \( p_{n+1} = p_{n+1}[G'] \). But it is easy to see that \( p_{n+1}[G'] \approx p_{n+1}[G] \), since every occurrence of \( g \) within \( p_{n+1} \) is applied to \( \lambda n' \) for some \( n' > n \), and for all such \( n' \) we have \( G'(\lambda n') \approx \mu[G, X] \). But \( p_{n+1}[G]X' \approx l \) by assumption; thus \( p_{n+1}X' \approx l \), completing the proof that \( p_{n+1}[G']X \approx 0 \). Once again, this contradicts \( q \approx p_n \), so claim 3 is established.

We have now completed a circle, in the sense that claim 3 tells us that \( \ll r[g \mapsto \mu_n g, x \mapsto X] \gg \) itself satisfies the hypothesis for \( q \) (with \( n+1 \) in place of \( n \)), and so can be iteratively subjected to the same analysis. However, it still remains to see what this analysis tells us about the syntactic form of \( r \) as distinct from \( \ll r[g \mapsto \mu_n g] \gg \). (In the light of claim 3, the variable \( x \) may be safely ignored here.)

Applying the first part of the above lemma to \( q' = \ll r[g \mapsto \mu_n g] \gg \), we see that this term has the form \( \lambda x'. \text{ case } ga'x'o' \text{ of } (\cdots) \). We may now ask what is the origin of the head occurrence of \( g \) here within \( r[g \mapsto \mu_n g] \). Clearly, there will be an occurrence of \( g \) in \( r \), say as indicated by \( r = \lambda x'. E[d] \) where \( d = \text{ case } ga''r''o'' \text{ of } (\cdots) \), that originates the head \( g \) of \( q' \) via the substitution \( g \mapsto \mu_n g \). (There will also be a particular bound occurrence of \( g \) within \( \mu_n \) that is responsible for this head \( g \), but this need not concern us.) Since we are wishing to show that \( r \) is pre-spinal, the next step should be to show that \( E[\ldots] \) is exposed. A mild generalization of this (suitable for our analysis at arbitrary depth) is given by the following:

39
Lemma 30 Suppose that $E[d] \gg = e = \text{case } g \cdots \text{ of } (\cdots)$, where $\dagger$ denotes $[g \mapsto \mu_{n,n}g]$, and $d$ is the expression whose head $g$ originates the indicated $g$ on the right-hand side. Then the position $E[-]$ is exposed, and its associated local variable environment is empty.

Proof We will show that one of the following must apply:

1. $E[-] = -$.
2. $E[-]$ has the form $\text{case } \cdots \text{ of } (j \Rightarrow E'[-] | \cdots)$ for some $j$ and $E'[-]$, where $\ll E'[d] \gg$ has head $g$ arising from the head $g$ of $d$.
3. $E[-]$ has the form $\text{case } g(\lambda.E''[-])r''o'' \text{ of } (\cdots)$ for some $E''[-]$, where $\ll E''[d] \gg$ has head $g$ arising from the head $g$ of $d$.

To see this, suppose that $E[-] \not= -$. Since $E[-]$ is also not constant, we have $E[d] = \text{case } b \text{ of } (j \Rightarrow e_j)$ (where one of $b, e_j$ contains the critical occurrence of $d$). There are two possibilities for the evaluation of $E[d] \dagger$:

- $b \dagger$ evaluates to some numeral $j$, and $e_j \dagger$ evaluates to $e$. Then the critical occurrence of $d$ lies within $e_j$, so taking $E'[-]$ to be the position in $e_j$ of this occurrence, the conditions of case 2 above are satisfied.

- $b \dagger$ evaluates to a case expression with head $g$. In this case, $b$ itself cannot have head $x$, so $b$ is of the form $ga''r''o''$, and

$$b \dagger \approx (\mu_{n}g)a''r''o'' \approx \text{case } a'' \text{ of } (n, \ldots, n' \Rightarrow 0 | - \Rightarrow ga''r''o'') \ .$$

Since $b$ is not itself a case expression, the critical occurrence of $d$ must lie within $a''$, $r''$ or $o''$. Here again, there are two subcases:

- The head $g$ of $b \dagger$ originates from $a'' \dagger$. Then taking $\lambda.E''[-]$ to be the position in $d''$ of the critical occurrence of $d$, the conditions of case 3 above are satisfied.

- $a'' \dagger$ evaluates to a numeral, and the $g$ of $ga''r''o'' \dagger$ becomes the head $g$ of $b \dagger$. In this case, it is an occurrence of $g$ in $E[-]$ itself (namely the head of $ga''r''o''$), rather than one in $d$, that originates the head $g$ in $e$, contradicting our assumption.

From this, we see inductively that $E[-]$ is a finite composition of context fragments of the above three kinds. Since the first argument to $g$ is of type 0, this means that $E[-]$ is exposed. For the same reason, its local variable environment is empty, since only empty $\lambda$-abstractions are involved. \(\square\)

The next lemma now gives what we need in order to iterate our analysis of $r$ to arbitrary depth:

Lemma 31 Suppose that $r \dagger \succeq p_{n+1}$, where $\dagger = [g \mapsto \mu_{n,n}g]$. Then $r$ has the form $\lambda x'. E[\text{case } ga''r''o'' \text{ of } (\cdots)]$, where:
1. $E[-]$ is exposed,

2. $o''^o \approx x''^o$ for a suitable substitution $\circ$,

3. $r''^\dagger \succeq p_{n'+2}$, where $\dagger = [g \mapsto \mu_{n,n'+1}g]$.

**Proof** Applying Lemma 29 (with $n = n'+1$) to $r''^\dagger$, we see that $r''^\dagger$ has the form $q''^\dagger = \lambda x\cdot \text{case } ga''^o\circ r''^o\text{ of } (\cdots)$, where $o''^o \approx x''^o$ for a suitable substitution $\circ$, and $r''[g \mapsto \mu_{n'+1}g] \succeq p_{n'+2}$. Write $r$ as $\lambda x'. E[d]$ where the head $g$ of $d = \text{case } ga''^o\circ r''^o\text{ of } (\cdots)$ is the origin of that in $q'$ via the substitution $g \mapsto \mu_{n,n'+1}g$. By Lemma 30, $E[-]$ is exposed, giving claim 1. Furthermore, since the local variable environment for $E[-]$ is empty, we actually have $a''^\dagger \approx a'$, $r''^\dagger \approx r'$, $o''^\dagger \approx o'$. Since $\dagger$ covers only $g$, taking $\circ = \dagger$ gives us claim 2. Finally, since $r' \approx r''^\dagger$, we have $r'[g \mapsto \mu_{n'+1}g] \approx r''^\dagger[g \mapsto \mu_{n'+1}g] \approx r''[g \mapsto \mu_{n,n'+1}]$, which gives claim 3. □

Having reached this point, we may immediately read off the following:

**Theorem 32** If $q \succeq p_n$, then $q$ is spinal.

**Proof** By Lemma 29, $q$ has the form $\lambda x. \text{case } g \text{aro } \text{ of } (\cdots)$ in which $r'[g \mapsto \mu_{n,n'+1}g] \succeq p_{n+1}$. Applying Lemma 31 iteratively to this readily yields the desired spinal structure as in Definition 20. □

Thus, if $t \approx \lambda G. p_0$ (for example), $t$ is spinal and so trivially contains a gremlin (in the modified sense appropriate to the type $0 \to (k+1) \to (k+1)$). By Theorem 27 (understood relative to this modified setting), we conclude that no such $t$ is present in $A_k^0$, and by Theorem 17, this means that no such $t$ can be definable in $\text{PCF}_k^0$. Since the interpretation of $\text{PCF}_k^0$ in $\text{SF}$ factors through $\text{SP}_k^0$, this in turn means that within the model $\text{SF}$, the element $[\lambda g, \Phi_0 | g]$ is not definable in $\text{PCF}_k^0$. However, this element is clearly definable relative to $Y_{0 \to (k+1)} \in \text{SF}$ even in $\text{PCF}_1$, so the proof of Theorem 2 is complete.

**References**


