Lambda-Definable Order-3 Tree Functions are Well-Quasi-Ordered

Kazuyuki Asada
Tohoku University, Sendai, Japan
asada@riec.tohoku.ac.jp
https://orcid.org/0000-0001-8782-2119

Naoki Kobayashi
The University of Tokyo, Tokyo, Japan
koba@is.s.u-tokyo.ac.jp

Abstract

Asada and Kobayashi [ICALP 2017] conjectured a higher-order version of Kruskal’s tree theorem, and proved a pumping lemma for higher-order languages modulo the conjecture. The conjecture has been proved up to order-2, which implies that Asada and Kobayashi’s pumping lemma holds for order-2 tree languages, but remains open for order-3 or higher. In this paper, we prove a variation of the conjecture for order-3. This is sufficient for proving that a variation of the pumping lemma holds for order-3 tree languages (equivalently, for order-4 word languages).

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases higher-order grammar, pumping lemma, Kruskal’s tree theorem, well-quasi-ordering, simply-typed lambda calculus

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.14


Acknowledgements We would like to thank anonymous referees for useful comments. This work was supported by JSPS Kakenhi 15H05706 and 18K11156.

1 Introduction

Kruskal’s tree theorem [7] says that the homeomorphic embedding relation \( \preceq_{he} \) on finite trees is a well-quasi-ordering, i.e., for every infinite sequence of trees \( \pi_0, \pi_1, \pi_2, \ldots \), there exist \( i < j \) such that \( \pi_i \preceq_{he} \pi_j \). Here, \( \pi \preceq_{he} \pi' \) means that there exists an embedding of the nodes of \( \pi \) to those of \( \pi' \), preserving the labels and the ancestor/descendant relation. Asada and Kobayashi [2] considered a higher-order version \( \preceq_{he}^\kappa \) of \( \preceq_{he} \) on simply-typed \( \lambda \)-terms of type \( \kappa \), and conjectured that \( \preceq_{he}^\kappa \) is also a well-quasi-ordering, for every simple type \( \kappa \). Under the assumption that the conjecture (which we call AK-conjecture) is true, they proved a pumping lemma for higher-order languages (a la higher-order languages in Damm’s IO hierarchy [3]), which says that for any order-\( k \) tree grammar that generates an infinite language \( L \), there exists a strictly increasing infinite sequence \( \pi_0 \prec_{he} \pi_1 \prec_{he} \pi_2 \prec_{he} \cdots \) such that \( \pi_i \in L \) and \( |\pi| \leq \exp_k(c \cdot d) \), where \( \prec_{he} \) is the strict version of the homeomorphic embedding, \( c \) and \( d \) are constants that depend on the grammar, and \( \exp_k(x) \) is defined by \( \exp_0(x) = x \) and \( \exp_{k+1}(x) = 2^{\exp_k(x)} \). The pumping lemma can be used to prove that a certain language does not belong to the class of order-\( k \) languages. They also proved that the conjecture is
true up to order-2 types, and hence also the pumping lemma for order-2 tree languages and (by the correspondence between tree/word languages [1, 3]) order-3 word languages. The AK-conjecture is still open for order-3 or higher.

In the present paper, we consider a variation of the AK-conjecture (which we call nAK-conjecture), where the homeomorphic embedding relation is replaced by \( \preceq^\# \), defined by \( \pi_1 \preceq^\# \pi_2 \) if and only if, for every tree constructor \( a \), \( \#_a(\pi_1) \leq \#_a(\pi_2) \); here \( \#_a(\pi) \) denotes the number of occurrences of \( a \) in \( \pi \). The correctness of the nAK-conjecture would imply the following variation of the pumping lemma: for any order-\( k \) tree grammar that generates an infinite language \( L \), there exists a strictly increasing infinite sequence \( \pi_0 \prec^\# \pi_1 \prec^\# \pi_2 \prec^\# \cdots \) such that \( \pi_i \in L \) and \( |\pi_i| \leq \exp_k(i + d) \). We prove that the nAK-conjecture is true for the order-3 case, i.e., that \( \preceq^N \) (the logical relation on simply-typed \( \lambda \)-terms of type \( \kappa \), obtained from \( \preceq^\# \)) is a well-quasi-ordering for any type \( \kappa \) of order up to 3. The variation of the pumping lemma above is thus obtained for order-3 tree languages and order-4 word languages. To our knowledge, pumping lemmas were known only for tree (word, resp.) languages of order up to 2 (3, resp.) [2]. A reduction from the well-quasi-orderedness of order-3 \( \lambda \)-terms to that of order-2 numeric \( \lambda \)-terms as (higher-order) tree \( \lambda \)-terms to numeric functions (that are also represented by \( \lambda \)-terms of type \( \kappa \), obtained from \( \preceq^\# \)) is a well-quasi-ordering. We prove that the nAK-conjecture is true for the order-3 case, i.e., that \( \preceq^N \) (the logical relation on simply-typed \( \lambda \)-terms of type \( \kappa \), obtained from \( \preceq^\# \)) is a well-quasi-ordering.

Related work. We are not aware of directly related work, besides our own previous work [2]. Our reduction from the well-quasi-orderedness of order-3 \( \lambda \)-terms to that of order-2 numeric functions relies on the inexpressiveness of simply-typed \( \lambda \)-terms as (higher-order) tree functions. Zaionc [11, 12, 13] studied the expressive power of simply-typed \( \lambda \)-terms of order up to 2 (3, resp.) [2]. We have further improved the result to obtain a pumping lemma for order-4 word (or, order-3 tree) languages. For clarity, we use the word order for this notion, and ordering for relations such as \( \preceq, \preceq^{\text{box}} \), etc.

## 2 Preliminaries

We give basic definitions on \( \lambda \)-terms and quasi-orderings.

### 2.1 \( \lambda \)-terms and higher-order languages

**Definition 1 (types and terms).** The set of simple types, ranged over by \( \kappa \), is given by:

\[
\kappa ::= \varnothing \mid \kappa_1 \to \kappa_2.
\]

The order \( 1 \) of a simple type \( \kappa \), written \( \text{order}(\kappa) \), is defined by \( \text{order}(\varnothing) = 0 \) and \( \text{order}(\kappa_1 \to \kappa_2) = \max(\text{order}(\kappa_1) + 1, \text{order}(\kappa_2)) \). The type \( \varnothing \) describes trees, and \( \kappa_1 \to \kappa_2 \) describes functions from \( \kappa_1 \) to \( \kappa_2 \). A ranked alphabet \( \Sigma \) is a map from a finite set of constants (that represent tree constructors) to the set of natural numbers called arities. The set of \( \lambda \)-terms, ranged over by \( s, t, u, v \), is defined by:

\[
t ::= x \mid a t_1 \cdots t_k \mid t_1 t_2 \mid \lambda x : \kappa. t \mid Y_{\kappa} t \mid t_1 \oplus t_2
\]

\[1\text{ For clarity, we use the word order for this notion, and ordering for relations such as } \preceq, \preceq^{\text{box}}, \text{etc.\]
Here, \(x, y, \ldots\) ranges over variables, and \(a\) over \(\text{dom}(\Sigma)\). The term \(a \ t_1 \cdots t_k\) (where we require \(\Sigma(a) = k\)) constructs a tree that has \(a\) as the root and (the values of) \(t_1, \ldots, t_k\) as children. \(Y_e\) and \(\oplus\) represent a fixed-point combinator and a non-deterministic choice, respectively. We often omit the type annotation and just write \(\lambda x. t\) and \(Y t\) for \(\lambda x : \kappa t\) and \(Y_e t\). A \(\Lambda Y\) term is called: (i) a \(\Lambda^\kappa\) term if it does not contain \(Y\); (ii) a \(\Lambda^\kappa\) term if it contains neither \(Y\) nor \(\oplus\); and (iii) an applicative term if it contains none of \(\lambda\)-abstractions, \(Y\), and \(\oplus\). We often call a \(\Lambda^\kappa\) term just a term. As usual, we identify \(\Lambda Y\) terms up to the \(\alpha\)-equivalence, and implicitly apply \(\alpha\)-conversions.

A **type environment** \(\Gamma\) is a sequence of type bindings of the form \(x : \kappa\) such that \(\Gamma\) contains at most one binding for each variable \(x\). A \(\Lambda Y\) term \(t\) has type \(\kappa\) under \(\Gamma\) if \(\Gamma \vdash_{ST} t : \kappa\) is derivable from the following typing rules.

\[
\begin{align*}
\Gamma, x : \kappa, \Gamma' \vdash_{ST} x : \kappa & \quad \Sigma(a) = k \quad \Gamma \vdash_{ST} t_i : \odot \text{ (for each } i \in \{1, \ldots, k\}\text{)} \quad \Gamma \vdash_{ST} t : \kappa \rightarrow \kappa \\
\Gamma \vdash_{ST} t_1 : \kappa_2 \rightarrow \kappa, \Gamma' \vdash_{ST} t_2 : \kappa_2 & \quad \Gamma, x : \kappa_1 \vdash_{ST} t : \kappa_2 \quad \Gamma \vdash_{ST} t_1 : \odot \quad \Gamma \vdash_{ST} t_2 : \odot \\
\Gamma \vdash_{ST} t_1 t_2 : \kappa & \quad \Gamma \vdash_{ST} \lambda x : \kappa_1 t : \kappa_2 \quad \Gamma \vdash_{ST} t_1 \odot t_2 : \odot
\end{align*}
\]

We consider below only well-typed \(\Lambda Y\) terms. Note that given \(\Gamma\) and \(t\), there exists at most one type \(\kappa\) such that \(\Gamma \vdash_{ST} t : \kappa\). We call \(\kappa\) the type of \(t\) (with respect to \(\Gamma\)). We often omit “with respect to \(\Gamma\)” if \(\Gamma\) is clear from context. Given a judgment \(\Gamma \vdash t : \kappa\), we define \(\Lambda \Gamma.t\) by: \(\lambda \kappa t := t\) and \(\Lambda (\Gamma', x : \kappa')t := \Lambda \Gamma, \lambda x.t\). Also we define \(\Gamma \rightarrow \kappa\) by: \(\emptyset \rightarrow \kappa := \kappa\) and \((\Gamma, x : \kappa') \rightarrow \kappa := \Gamma \rightarrow (\kappa' \rightarrow \kappa)\); thus we have \(\Gamma \vdash \Lambda \Gamma.t : \kappa\) if \(\Gamma \vdash t : \kappa\). Given an alphabet \(\Sigma\), we write \(\Lambda^{\Sigma}_{\kappa}\) for the set of \(\Lambda^\kappa\) terms whose constants are taken from \(\Sigma\). Also we define \(\Lambda_{\emptyset, \kappa}^{\Sigma} := \{ t \in \Lambda^{\Sigma} \mid \Gamma \vdash t : \kappa\} \) and \(\Lambda_{\emptyset}^{\Sigma} := \Lambda_{\emptyset, \kappa}^{\Sigma}\).

For a \(\Lambda Y\) term \(t\) with a type environment \(\Gamma\), the **(internal)** order of \(t\) (with respect to \(\Gamma\)), written \(\text{order}_{\Gamma}(t)\), is the largest order of the types of subterms of \(\Lambda \Gamma.t\), and the **external order** of \(t\) (with respect to \(\Gamma\)), written \(\text{order}_{\emptyset}(t)\), is the order of the type of \(t\) with respect to \(\Gamma\). We often omit \(\Gamma\) when it is clear from context. For example, for \(t = (\lambda x : \odot x) e\), \(\text{order}_{\emptyset}(t) = 1\) and \(\text{order}_{\emptyset}(t) = 0\). We define the size \(|t|\) of a \(\Lambda Y\) term \(t\) by: \(|x| := 1\), \(|a t_1 \cdots t_k| := 1 + |t_1| + \cdots + |t_k|\), \(|s t| := |s| + |t| + 1\), \(|\lambda x.t| := |t| + 1\), \(|Y e t| := |t| + 1\), and \(|s \oplus t| := |s| + |t| + 1\). We call a \(\Lambda Y\) term \(t\) ground (with respect to \(\Gamma\)) if \(\Gamma \vdash_{ST} t : \odot\). We call \(t\) a (finite, \(\Sigma\)-ranked) tree if \(t\) is a ground closed applicative term (consisting of only constants). We write \(\text{Tree}_{\Gamma}\) for the set of \(\Sigma\)-ranked trees, and use the meta-variable \(\pi\) for a tree. We often write \(\rightsquigarrow\) to denote a sequence (possibly with a condition on the range of the sequence in the superscript). For example, \(\overrightarrow{t_1 \cdots t_m}\) denotes the sequence \(t_1, \ldots, t_m\) of terms, and \(\overrightarrow{[t_1/x_1, \ldots, t_m/x_m]}\) denotes the substitution \([t_1/x_1, \ldots, t_m/x_m]\).

We sometimes identify a ranked alphabet \(\Sigma = \{a_1 \rightarrow r_1, \ldots, a_k \rightarrow r_k\}\) with the first-order environment \(\Sigma = \{a_1 : \sigma^{a_1} \rightarrow \odot, \ldots, a_k : \sigma^{a_k} \rightarrow \odot\}\) (assuming an arbitrary fixed linear ordering on \(\Sigma\)).

**Definition 2** (reduction and language). The set of *(call-by-name)* evaluation contexts is defined by:

\[
E ::= [\underline{\mid}] t_1 \cdots t_k | a \ \underline{\mid} \pi_i \rightarrow \underline{\mid} E t_1 \cdots t_k
\]

and the call-by-name reduction for (possibly open) ground \(\Lambda Y\) terms is defined by:

\[
E[(\lambda x.t)t'] \rightarrow E[t'[t/x]] \quad E[Y t] \rightarrow E[t(Y t)] \quad E[t_1 \odot t_2] \rightarrow E[t_i] \quad (i = 1, 2)
\]

where \(t'[t/x]\) is the usual capture-avoiding substitution. We write \(\rightarrow^*\) for the reflexive transitive closure of \(\rightarrow\). A *call-by-name normal form* is a ground \(\Lambda Y\) term \(t\) such that
t \not\rightarrow t' for any t'. For a ground closed $\lambda Y^\text{nd}$-term $t$, we define the tree language $\mathcal{L}(t)$ generated by $t$ by $\mathcal{L}(t) := \{ \pi \mid t \rightarrow^* \pi \}$. For a ground closed $\lambda^\gamma$-term $t$, $\mathcal{L}(t)$ is a singleton set $\{ \pi \}$; we write $\mathcal{T}(t)$ for such $\pi$ and call it the tree of $t$.

In the previous paper [2] we stated the pumping lemma for the notion of a higher-order grammar; in this paper, following [8, 9], we use only the formalism by $\lambda Y^\text{nd}$-terms for simplicity. Since there exist well-known order-preserving and language-preserving transformations between higher-order grammars and ground closed $\lambda Y^\text{nd}$-terms, we obtain corresponding results on higher-order grammars immediately.

The notion of a word can be seen as a special case of that of a tree:

Definition 3 (word alphabet). We call a ranked alphabet $\Sigma$ a word alphabet if it has a special nullary constant $\epsilon$ and all the other constants have arity 1. For a tree $\pi = a_1(\cdots(a_n\epsilon)\cdots)$ of a word alphabet, we define word$(\pi) := a_1\cdots a_n$, and we define utree as the inverse function of word, i.e., utree$(a_1\cdots a_n) := a_1(\cdots(a_n\epsilon))$. The word language generated by a ground closed $\lambda Y^\text{nd}$-term $t$ over a word alphabet, written $\mathcal{L}_u(t)$, is defined as $\{ \text{word}(\pi) \mid \pi \in \mathcal{L}(t) \}$.

A tree language (word language, resp.) over an alphabet (word alphabet, resp.) $\Sigma$ is called order-$n$ if it is generated by some order-$n$ ground closed $\lambda Y^\text{nd}$-term of $\Sigma$; we note that the classes of order-0, order-1, and order-2 word languages coincide with those of regular, context-free, and indexed languages, respectively [10].

2.2 Some quasi-orderings and their logical relation extension

Definition 4 ((well-)quasi-ordering). A quasi-ordering (a.k.a. preorder) on a set $A$ is a binary relation on $A$ that is reflexive and transitive. A well-quasi-ordering (wqo for short) on a set $S$ is a quasi-ordering $\preceq$ on $S$ such that for any infinite sequence $(s_i)_i$ of elements in $S$ there exist $j$ and $k$ such that $j < k$ and $s_j \preceq s_k$.

As a general notation, for a quasi-ordering denoted by $\preceq$, we write $\approx$ for the induced equivalence relation (i.e., $x \approx y$ if $x \preceq y$ and $y \preceq x$), and write $\prec$ for the strict version (i.e., $x \prec y$ if $x \preceq y$ and $y \npreceq x$). Also, for a quasi-ordering denoted by $\preceq$, we write $\sim$ for the induced equivalence relation and $\prec$ for the strict version. We apply these conventions also to notations with superscript/subscript such as $\preceq^a$, $\preceq^b$, $\preceq_1^a$, $\preceq_2^a$, $\preceq_3^b$, and $\preceq_4^b$. Further, for any quasi-ordering on the set of trees of a word alphabet, we use the same notation also for the quasi-ordering on the set of words induced through utree.

Definition 5 (logical relation extension). Let $\Sigma$ be a ranked alphabet. We call $\preceq$ a base quasi-ordering (with respect to $\Sigma$) if $\preceq$ is a quasi-ordering on the set $\Lambda^\Sigma_\kappa$ modulo $\beta\eta$-equivalence and every constant in $\Sigma$ is monotonic on $\preceq$. We define the logical relation extension of $\preceq$ as the family $(\preceq_\kappa)_\kappa$ of relations $\preceq_\kappa$ on the set $\Lambda^\Sigma_\kappa$ modulo $\beta\eta$-equivalence indexed by simple types $\kappa$ where $\preceq_\kappa$’s are defined by induction on $\kappa$ as follows:

For any $t_1', t_2', t_1, t_2$,

\[
t_1 \preceq_\kappa t_2 \quad \text{if} \quad t_1 \preceq t_2,
\]

\[
t_1 \preceq_{\kappa \rightarrow \kappa'} t_2 \quad \text{if} \quad \text{for any } t_1', t_2', t_1' \preceq_{\kappa'} t_2' \Rightarrow t_1 t_1' \preceq_{\kappa'} t_2 t_2'.
\]

Furthermore we extend the relation to open terms: for $t_1, t_2 \in \Lambda^\Sigma_{\Gamma, \kappa}$, we define $t_1 \preceq_{\Gamma, \kappa} t_2$ if $\lambda \Gamma. t_1 \preceq_{\Gamma \rightarrow \kappa} \lambda \Gamma. t_2$. We omit the subscripts of $\preceq_\kappa$ and $\preceq_{\Gamma, \kappa}$ if there is no confusion.

The next lemma follows immediately from the basic lemma (a.k.a. the abstraction theorem) of logical relations (see the full version for details).
Lemma 6. Let \( \leq \) be a base quasi-ordering. Each component \( \leq_\kappa \) of the logical relation extension of \( \leq \) is a quasi-ordering. Further, \( \leq_\kappa \) is the point-wise quasi-ordering:
\[
t_1 \leq_\kappa \rightarrow_\kappa t_2 \quad \text{if and only if} \quad \text{for any } t' \in \Lambda_\kappa, \quad t_1 t' \leq_\kappa t_2 t'.
\]

Every quasi-ordering for higher-order terms used in this paper is a logical relation extension (of some base quasi-ordering). The next ordering is used in the previous paper [2].

Definition 7 (homeomorphic embedding). Let \( \Sigma \) be a ranked alphabet. The homeomorphic embedding ordering \( \leq_{he, \Sigma} \) between \( \Sigma \)-ranked trees\(^2\) is inductively defined by the following rules:
\[
\begin{align*}
\pi_i & \leq_{he, \Sigma} \pi'_i \quad (\text{for all } i \leq k) & k = \Sigma(a) & \pi \leq_{he, \Sigma} \pi & k = \Sigma(a) > 0 & 1 \leq i \leq k
\end{align*}
\]

We extend the above ordering to a base ordering by:
\[
t_1 \leq_{he, \Sigma} t_2 \quad \text{if} \quad T(t_1) \leq_{he, \Sigma} T(t_2).
\]

For example, \( \text{br a b} \leq_{he} \text{br (br a c) b} \). The homeomorphic embedding on words is nothing but the (scattered) subsequence ordering. The following is a fundamental result on the homeomorphic embedding:

Proposition 8 (Kruskal’s tree theorem [7]). For any (finite) ranked alphabet \( \Sigma \), the homeomorphic embedding \( \leq_{he} \) on \( \Sigma \)-ranked trees is a well-quasi-ordering.

Also, we often use the Dickson’s theorem [6] which says that the product quasi-ordering (component-wise quasi-ordering) of a finite number of wqo’s is a wqo.

The next is the quasi-ordering that is used in the theorems in this paper.

Definition 9 (occurrence-number quasi-ordering). Let \( \Sigma \) be a ranked alphabet. For \( a \in \Sigma \) and a \( \Sigma \)-tree \( \pi \), we define \( #_a(\pi) \) as the number of occurrences of \( a \) in \( \pi \), and extend this to a ground closed \( \lambda \)-term \( t \) by \( #_a(t) := #_a(T(t)) \). Then we define a base quasi-ordering \( \leq_{#, \Sigma, \cdot} \) by:
\[
t_1 \leq_{#, \Sigma, \cdot} t_2 \quad \text{if} \quad #_a(t_1) \leq #_a(t_2).
\]

Also we define a base quasi-ordering \( \leq_{#, \cdot, \Sigma} \) by:
\[
t_1 \leq_{#, \cdot, \Sigma} t_2 \quad \text{if} \quad \text{for every } a \in \Sigma, \quad t_1 \leq_{#, \Sigma, \cdot} t_2.
\]

Note that \( \pi \leq_{he} \pi' \) implies \( \pi \leq_{#, \Sigma, \cdot} \pi' \), shown by induction on the rule of \( \leq_{he} \); and further \( \pi \leq_{he} \pi' \) implies \( \pi \leq_{#, \cdot, \Sigma} \pi' \) for any \( \kappa \) since \( \leq_{he} \) and \( \leq_{#, \Sigma} \) are point-wise quasi-ordering. Also note that \( \leq_{#, \cdot, \Sigma} = \cap_{\kappa \in \Sigma} (\leq_{#, \Sigma, \cdot}) \) for any \( \kappa \).

The next quasi-ordering is used just in proofs. We write \( \Sigma_\mathbb{N} \) for the ranked alphabet \( \{0 \mapsto 0, 1 \mapsto 0, + \mapsto 2, \times \mapsto 2\} \); we write \( + t t' \) as \( t + t' \) and \( \times t t' \) as \( t \times t' \). We define a set-theoretical denotational interpretation \( [\cdot] \) of \( \Lambda_\Sigma^\mathbb{N} \) by:
\[
[\{0\}] := 0, \quad [\{1\}] := 1, \quad [+](m) := n + m, \quad \text{and} \quad [\times](m) := n \times m.
\]

For \( t_1, t_2 \in \Lambda_\Sigma^\mathbb{N} \), we write \( t_1 =_{\mathbb{N}} t_2 \) if \([t_1] = [t_2] \).

Definition 10 (natural number quasi-ordering). We define a base quasi-ordering \( \leq_{\mathbb{N}} \) on the set \( \Lambda_\Sigma^\mathbb{N} \) by:
\[
t_1 \leq_{\mathbb{N}} t_2 \quad \text{if} \quad [t_1] \leq [t_2].
\]

\(^2\) In the usual definition, a quasi-ordering on labels (tree constructors) is assumed. Here we fix the quasi-order on labels to the identity relation.
3 Numeric Pumping Lemma for Higher-order Tree Languages

Here we explain the nAK-conjecture and the pumping lemma for higher-order tree languages with respect to $\preceq_{\#}^{\Sigma}$.

▶ **Conjecture 11** (nAK-conjecture). For any $\Sigma$ and $\kappa$, $\preceq_{\#}^{\Sigma}$ is a well quasi-ordering.

Our main theorem (Theorem 14) is to show the above conjecture for $\kappa$ of order up to 3. The above conjecture (and Theorem 14) can be used for the following pumping lemma:

▶ **Theorem 12** (pumping lemma). Assume that Conjecture 11 holds. Then, for any order-$n$ ground closed $\lambda Y^{nd}$-term $t$ of a ranked alphabet $\Sigma$ such that $L(t)$ is infinite, there exist an infinite sequence of trees $\pi_0, \pi_1, \pi_2, \ldots \in L(t)$, and constants $c, d$ such that:

(i) $\pi_0 \prec_{\#}^{\Sigma} \pi_1 \prec_{\#}^{\Sigma} \pi_2 \prec_{\#}^{\Sigma} \ldots$, and

(ii) $|\pi_i| \leq \exp_n (ci + d)$ for each $i \geq 0$.

Furthermore, we can drop the assumption on Conjecture 11 when $n \leq 3$.

The proof of the above theorem is obtained as a simple modification of the proof of the pumping lemma in [2]: see the full version.

▶ **Remark.** The theorem we prove in the full version is actually slightly stronger than Theorem 12 above, in the following three points (see the full version for details):

(i) As in [2], we relax the assumption of nAK conjecture, so that $\preceq_{\#}^{\Sigma}$ need not be the logical relation; any higher-order extension of the base quasi-ordering that is closed under application suffices.

(ii) As in [2], we use actually a weaker conjecture, called the periodicity, which requires that, for any $\vdash_{ST} t : \kappa \rightarrow \kappa$ and $\vdash_{ST} s : \kappa$, there exist $i, j > 0$ such that $t^i s \preceq_{\#}^{\Sigma} t^{i+j} s \preceq_{\#}^{\Sigma} \ldots$.

(iii) Whilst Theorem 12 states a pumping lemma on $\preceq_{\#}^{\Sigma}$, the generalized theorem states a pumping lemma on arbitrary base quasi-ordering with certain conditions, which includes $\preceq_{\#}^{\Sigma}$ and $\preceq_{ho}$ as instances.

By the correspondence between order-$n$ tree grammars and order-$(n+1)$ word grammars [3, 1], we also have:

▶ **Corollary 13** (pumping lemma for word languages). Assume that Conjecture 11 holds. Then, for any order-$n$ ground closed $\lambda Y^{nd}$-term $t$ of a word alphabet $\Sigma$ (where $n \geq 1$) such that $L_w(t)$ is infinite, there exist an infinite sequence of words $w_0, w_1, w_2, \ldots \in L_w(t)$, and constants $c, d$ such that:

(i) $w_0 \prec_{\#}^{\Sigma} w_1 \prec_{\#}^{\Sigma} w_2 \prec_{\#}^{\Sigma} \ldots$, and

(ii) $|w_i| \leq \exp_{n-1}(ci + d)$ for each $i \geq 0$.

Furthermore, we can drop the assumption on Conjecture 11 when $n \leq 4$.

4 Numeric Version of Order-3 Kruskal’s Tree Theorem

Here we prove the main theorem (Theorem 14 below), which states that the nAK-conjecture (Conjecture 11) holds for order-3 types. In this whole section, by a term, we mean a $\lambda \rightarrow$-term, and we never consider a fixed-point combinator nor non-determinism.
4.1 Main theorem

Theorem 14. For any alphabet $\Sigma$ and any type $\kappa$ of order up to 3, $\preceq^{\#}[\Sigma, \kappa]$ on $\Lambda^{\Sigma}_{\kappa}$ is a wqo.

The theorem above is obtained as a corollary of the following lemma.

Lemma 15. For any alphabet $\Sigma$, any $a \in \Sigma$, and any order-2 type environment $\Gamma$ (i.e., a type environment whose codomain consists of types of order up to 2), the quasi-ordering $\preceq^{\#}[\Sigma, \kappa, a]$ on $\Lambda^{\Sigma}_{\kappa, a}$ is a wqo.

Proof sketch of Theorem 14.

For Theorem 14, it is sufficient that $\preceq^{\#}[\Sigma, \kappa, a]$ on $\Lambda^{\Sigma}_{\kappa, a}$ is a wqo for every $a \in \Sigma$ and $\kappa$ with $\text{order}(\kappa) \leq 3$, because $\preceq^{\#}[\Sigma, \kappa, a] = \cap_{a \in \Sigma}(\preceq^{\#}[\Sigma, \kappa])$ and well-quasi-orderings are closed under finite intersection.

For $\preceq^{\#}[\Sigma, \kappa, a]$ to be a wqo for every order-3 type $\kappa$, it is sufficient that the restriction of $\preceq^{\#}[\Sigma, \kappa, a]$ to $\Lambda^{\Sigma}_{\kappa}$ (i.e., $\preceq^{\#}[\Sigma, \kappa, a] \cap (\Lambda^{\Sigma}_{\kappa} \times \Lambda^{\Sigma}_{\kappa})$) is a wqo for every order-3 type $\kappa$, because $t_{1} \preceq^{\#}[\Sigma, \kappa, a] t_{2}$ holds if $\lambda \Sigma.t_{1}(\preceq^{\#}[\Sigma, \kappa, a] \cap (\Lambda^{\Sigma}_{\kappa} \times \Lambda^{\Sigma}_{\kappa}))\lambda \Sigma.t_{2}$, and $\text{order}(\Sigma) \rightarrow \kappa \leq 3$.

For $\preceq^{\#}[\Sigma, \kappa, a] \cap (\Lambda^{\Sigma}_{\kappa} \times \Lambda^{\Sigma}_{\kappa})$ to be a wqo, Lemma 15 is sufficient, because $t_{1}(\preceq^{\#}[\Sigma, \kappa, a] \cap (\Lambda^{\Sigma}_{\kappa} \times \Lambda^{\Sigma}_{\kappa})) t_{2}$ holds if $t_{1} z_{1} \cdots z_{k} \preceq^{\#}[\Sigma, \kappa, a] t_{2} z_{1} \cdots z_{k}$, where $\kappa = \kappa_{1} \to \cdots \to \kappa_{k} \to \circ$ and $\Gamma = \kappa_{1}, \ldots, z_{k} : \kappa_{k}$.

See the full version for details.

4.2 Transformation from order-3 terms to order-2 terms

The key observation behind the transformation $(\cdot)^{3}$ is as follows. Let $s$ be a closed term of type $\circ^{m} \rightarrow \circ$ and $t_{1}, \ldots, t_{m}$ be closed terms of type $\circ$. Then, we have:

$$\#_{\circ}(s t_{1} \cdots t_{m}) = c_{1} \times \#_{\circ}(t_{1}) + \cdots + c_{m} \times \#_{\circ}(t_{m}) + d$$

for some numbers $c_{1}, \ldots, c_{m}, d$ that do not depend on $t_{1}, \ldots, t_{m}$. This is because the order-1 function $s$ representable as a $\lambda^{m}$-term can copy only arguments, and the number of copies cannot depend on the arguments. Thus, if we are interested only in the number of occurrences of a constant, information about an order-1 function can be represented by a tuple $(c_{1}, \ldots, c_{m}, d)$ of numbers (order-0 values, in other words). By lifting this representation to order-3 terms in $\Lambda^{\circ}_{\Gamma_{c}, a}$, we obtain order-2 terms in $\Lambda^{\circ}_{\Gamma_{c}, a}$.

The actual transformation is non-trivial. Let us first fix $\Gamma = \varphi_{1} : \kappa_{1}, \ldots, \varphi_{m} : \kappa_{m}, f_{1} : \circ^{m} \rightarrow \circ, \ldots, f_{t} : \circ^{m} \rightarrow \circ$. Here, $\varphi_{i}$’s are order-2 variables and $f_{j}$’s are variables of order up to 1. Every element of $\Lambda^{\circ}_{\Gamma_{c}, a}$ can be normalized to a term generated by the following syntax (which we call an order-3 normal form):

$$t ::= y \mid t_{1} t_{2} \mid \varphi_{i} t_{1} \cdots t_{k} \mid \lambda y.t.$$

Here, $y$ is a local variable of order 0. We require that the order of $\varphi t_{1} \cdots t_{k}$ is at most 1. For example, $\varphi : (\circ \rightarrow \circ) \rightarrow \circ \rightarrow \circ \rightarrow \circ$, $f : \circ \rightarrow \circ \rightarrow \circ$, $x : \circ \rightarrow \lambda y : \circ. \varphi(f x)((\lambda y' : \circ. f y' y') y) : \circ \rightarrow \circ \rightarrow \circ$ is an order-3 normal form. It can be checked by induction that for any order-3 normal form $t$, $\text{order}_{\Gamma}(t) \leq 1$ (with a suitable environment $\Gamma$). Since any
Lambda-Definable Order-3 Tree Functions are Well-Quasi-Ordered

Let $\varphi_1, \varphi_m : \kappa, f_1 : \sigma^m \to \sigma, \ldots, f_0 : \sigma^\ell \to \sigma; y_1 : \sigma, \ldots, y_n : \sigma \vdash t : \sigma^\ell \to \sigma$
to a term $e$ with some suitable type environment. Here, $y_1, \ldots, y_n$ are order-0 variables that
are bound inside $t_0$ (rather than $t$), order($\kappa_i$) = 2 for $i \leq m$, and $\kappa_i \geq 0$ for $i \leq \ell$. We call
$f_i$ and $\varphi_i$ external variables and $y_i$ an internal variable. Note that an external variable $f_i$
can be order-0.

We first explain how variables and environments are transformed.

The variables $y_1, \ldots, y_n$ will just disappear after the transformation.

For each order-1 variable $f_i$ of type $\sigma^\ell \to \sigma$, we prepare a tuple of variables $(c_{f_i}, \ldots, c_{f_i}, q_i, d_{f_i})$. Each $c_{f_i,j}$ expresses how often $f_i$ copies the $j$-th argument, and $d_{f_i}$ expresses how
often $a_{f_i}$ occurs in the value of $f_i$, so that the number of $a_{f_i}$ in $f_i t_1, \ldots, t_q$, can be represented by
$c_{f_i,1} \times \# a_{f_i}(t_1) + \cdots + c_{f_i,q_i} \times \# a_{f_i}(t_q) + d_{f_i}$ (recall the observation given
at the beginning of this subsection).

For each order-2 variable $\varphi_i$ of type $\kappa_i = (\sigma^\ell \to \sigma) \to \cdots \to (\sigma^\ell \to \sigma) \to (\sigma^\ell \to \sigma)$
(where $q_i > 0$), we prepare a tuple of order-1 variables $(g_{\varphi_i,1}, \ldots, g_{\varphi_i, q_i}, h_{\varphi_i}, h_{\varphi_i})$. Basically,
$g_{\varphi_i, j}$ and $h_{\varphi_i}$ are analogous to $c_{f_i, j}$ and $d_{f_i}$, respectively. Given order-1 functions $t_1, \ldots, t_k$
whose values are $\tilde{u}_1, \ldots, \tilde{u}_k$ (where each $\tilde{u}_i$ is a tuple of size $q_i + 1$), for each $j \leq q$, the
function $\varphi_i t_1 \cdots t_k$ copies the $j$-th order-0 argument $g_{\varphi_i,j}(\tilde{u}_1, \ldots, \tilde{u}_k)$ times, and creates $h_{\varphi_i}(\tilde{u}_1, \ldots, \tilde{u}_k)$ copies of the constant $a_{f_i}$. The other function variable $h_{\varphi_i}$ is similar
to $h_{\varphi_i}$, but used for counting an internal variable $y_j$ rather than $a_{f_i}$.

For a type environment

$\Gamma = \varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa, f_1 : \sigma^m \to \sigma, \ldots, f_0 : \sigma^\ell \to \sigma$

where $\kappa_i = (\sigma^l \to \sigma) \to \cdots \to (\sigma^l \to \sigma) \to (\sigma^l \to \sigma)$ ($q_i > 0$, $i = 1, \ldots, k$), we define:

$\Gamma^\ell := g_{\varphi_i, m}^{j \leq q}, h_{\varphi_i}, h_{\varphi_i} : \sigma^{l+1} \to \cdots \to \sigma^{l+1} \to \sigma$

We now define the transformation of terms. A term $t$ such that

$\varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa, f_1 : \sigma^m \to \sigma, \ldots, f_0 : \sigma^\ell \to \sigma; y_1 : \sigma, \ldots, y_n : \sigma \vdash t : \sigma^\ell \to \sigma$

is transformed to a tuple $(v_1, \ldots, v_n; w_1, \ldots, w_r; e)$, using the transformation relation

$\varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa, f_1 : \sigma^m \to \sigma, \ldots, f_0 : \sigma^\ell \to \sigma; y_1 : \sigma, \ldots, y_n : \sigma \vdash t : \sigma^\ell \to \sigma$
defined below. Here, each component is constructed from variables $c_{f_i, j}, d_{f_i}, g_{\varphi_i, j}, h_{\varphi_i}, h_{\varphi_i}$
above and $\times, +, 0, 1$. The output of the transformation consists of three parts, separated by
semicolons: a (possibly empty) sequence $v_1, \ldots, v_n$, a (possibly empty) sequence $w_1, \ldots, w_r$, and
a single element $e$. The term $v_j$ represents how often $y_j$ is copied, $w_j$ represents how
often the $j$-th argument of $t$ is copied, and $e$ represents how often the constant $a_{f_i}$ is copied.
The terms $v_j$ and $w_j$ are auxiliary ones for this transformation, and $e$ plays the role of $t^2$
explained in Section 4.1.
The transformation relation is defined by the following rules, where \( \Gamma = \varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa_m, f_1 : \sigma^{\varphi_1} \to o, \ldots, f_{i} : \sigma^{\varphi_i} \to o \) is fixed.

\[
\Gamma; y_1 : o, \ldots, y_n : o \vdash y_j \triangleright (0, \ldots, 0, 1, 0, \ldots, 0 ; 0) \\
\quad \text{(IVAR)}
\]

\[
\Gamma; y_1 : o, \ldots, y_n : o \vdash f_i \triangleright (0, \ldots, 0; c_{f_i, 1}, \ldots, c_{f_i, q_i}; d_{f_i}) \\
\quad \text{(VAR)}
\]

\[
\Gamma; y_1 : o, \ldots, y_n : o \vdash t_1 \triangleright (v_1, \ldots, v_n; w_1, \ldots, w_r; e) \\
\Gamma; y_1 : o, \ldots, y_n : o \vdash t_2 \triangleright (v'_1, \ldots, v'_{n';} e') \\
\Gamma; y_1 : o, \ldots, y_n : o \vdash t_1 t_2 \triangleright (v_1 + w_1 v'_1, \ldots, v_n + w_n v'_{n';} w_2, \ldots, w_r; e + w_r e') \\
\quad \text{(APP0)}
\]

\[
\Gamma; y_1 : o, \ldots, y_n : o \vdash t_j \triangleright (\vec{v}_j; \vec{w}_j; e_j) \\
\quad \text{for each } j \in \{1, \ldots, k\} \\
\bar{u}_{j, j'} = (\vec{w}_j; \vec{v}_{j'}) \\
\quad \text{for each } j \in \{1, \ldots, k\} \text{ and } j' \in \{1, \ldots, n\} \\
\bar{u} = (\bar{u}_{j, j'}) \\
\quad \text{for each } k \geq 1 \text{ and the type of } t_k \text{ is order-1} \\
\quad \text{(APP1)}
\]

\[
\Gamma; y_1 : o, \ldots, y_n : o, y_{n+1} : o \vdash t \triangleright (v_1, \ldots, v_n, v_{n+1}; w_1, \ldots, w_r; e) \\
\quad \text{for order-1 variables}, \\
\Gamma; y_1 : o, \ldots, y_n : o \vdash \lambda y_{n+1}. t \triangleright (v_1, \ldots, v_n, v_{n+1}; w_1, \ldots, w_r; e) \\
\quad \text{for order-2 functions} \\
\quad \text{(LAM)}
\]

Rules (IVAR) (for internal variables of type \( o \)) (VAR) (for order-1 variables), and (LAM) should be obvious from the intuition on the tuple and the translation of an environment. Rules (APP0) and (APP1) are for applications of order-1 and order-2 functions respectively. (Note however that in (APP0), \( t_1 \) itself may be an application of order-2 function, of the form \( \varphi t_{1,1} \ldots t_{1,k} \).) In (APP0), note that \( t_1 t_2 \) creates \( w_1 \) copies of \( t_2 \), so that the number of copies of \( y_i \) can be calculated by \( v_i + w_i v'_i \), where \( v_i \) and \( v'_i \) are the numbers of copies created by \( t_1 \) and \( t_2 \) respectively. Rule (APP1) is based on the intuition explained above about the translation of order-2 variables. Note that the same function \( \hat{h}_{\varphi_i} \) is used for counting \( y_1, \ldots, y_n \); this is because \( \varphi_i \) does not know \( y_j \) (in other words, \( \varphi_i \) cannot be instantiated to a term containing \( y_j \) as a free variable), so that the information for counting \( y_j \) can only be passed through arguments \( \vec{w}_{j, j'} \).

It should be clear that if \( \Gamma; y_1 : o, \ldots, y_n : o \vdash t \triangleright (v_1, \ldots, v_n; w_1, \ldots, w_r; e) \) then \( v_j, w_j, e \in A^{\Sigma_r}_{\Gamma^o} \) and the order of \( \Gamma^o \to o \) is no greater than 2.

**Example 16.** Let \( \Gamma = \varphi : (o \to o) \to o, f : o \to o \). Then, we have

\[ \Gamma^2 = g_{\varphi, 1}, h_{\varphi}, \hat{h}_{\varphi} : \sigma^2 \to o, c_{f, 1}, d_f : o \]

and \( t := \lambda y. \varphi f y \) is transformed to

\[ t^2 = h_{\varphi}(g_{\varphi, 1}(c_{f, 1}, d_f), h_{\varphi}(c_{f, 1}, d_f)) + g_{\varphi, 1}(g_{\varphi, 1}(c_{f, 1}, d_f), h_{\varphi}(c_{f, 1}, d_f)) \times 0 \]
by the following derivation:

\[
\begin{align*}
\Gamma; y : o &\vdash f \triangleright (0; c_{f,1}; d_f) \quad (\text{VAR}) \\
\Gamma; y : o &\vdash \varphi f \triangleright (\hat{h}_\varphi(c_{f,1}, 0); g_{\varphi,1}(c_{f,1}, d_f); h_\varphi(c_{f,1}, d_f)) \quad (\text{APP1}) \\
\Gamma; y : o &\vdash \varphi f \triangleright (\hat{h}_\varphi(u); g_{\varphi,1}(u); h_\varphi(u)) \quad (\text{APP1}) \\
\Gamma; y : o &\vdash y \triangleright (1; 0) \quad (\text{IVAR}) \\
\Gamma; y : o &\vdash \varphi f \triangleright (\hat{h}_\varphi(u); g_{\varphi,1}(u); h_\varphi(u)) \quad (\text{APP1}) \\
\Gamma; y : o &\vdash y \triangleright (1; 0) \quad (\text{IVAR}) \\
\end{align*}
\]

where \( u = g_{\varphi,1}(c_{f,1}, d_f), h_\varphi(c_{f,1}, d_f) \) and \( u' = g_{\varphi,1}(c_{f,1}, d_f), h_\varphi(c_{f,1}, 0) \). The terms in the bottom line of the derivation, \( \hat{h}_\varphi(u') + g_{\varphi,1}(u') \times 1 \) and \( t' = h_\varphi(u') + g_{\varphi,1}(u') \times 0 \), have type \( o \) under the environment \( \Gamma' \), and \( \text{eorder}(\lambda \Gamma' t') = \text{order}(\Gamma' \rightarrow o) = 2 \).

The next example is a slightly modified one involving an external variable \( x : o \) instead of the internal variable \( y : o \). We have

\[(\Gamma, x : o)^2 = \Gamma^3, d_x : o\]

and \( t' \coloneqq \varphi f x \) is transformed to

\[t'^2 = h_\varphi(g_{\varphi,1}(c_{f,1}, d_f), h_\varphi(c_{f,1}, d_f)) + g_{\varphi,1}(g_{\varphi,1}(c_{f,1}, d_f), h_\varphi(c_{f,1}, d_f)) \times d_x\]

by the following derivation:

\[
\begin{align*}
\Gamma; x : o &\vdash f \triangleright (0; c_{f,1}; d_f) \quad (\text{VAR}) \\
\Gamma; x : o &\vdash \varphi f \triangleright (\hat{h}_\varphi(c_{f,1}, 0); g_{\varphi,1}(c_{f,1}, d_f); h_\varphi(c_{f,1}, d_f)) \quad (\text{APP1}) \\
\Gamma; x : o &\vdash \varphi f \triangleright (\hat{h}_\varphi(u); g_{\varphi,1}(u); h_\varphi(u)) \quad (\text{APP1}) \\
\Gamma; x : o &\vdash x \triangleright (0; d_x) \quad (\text{APP0}) \\
\end{align*}
\]

where \( u \) and \( u' \) are the same as above.

Lemma 17 below says that the transformation preserves the meaning of ground terms. Here we regard constants in \( \Sigma \) as variables of up to order 1, and we define a substitution \( \theta^{\text{aux}}_\Sigma \) by:

\[
\theta^{\text{aux}}_\Sigma := \frac{\gamma a \in \Sigma, 1 \leq \text{ar}(a)}{1/c_{a,i}}, \frac{\gamma a \in \Sigma \setminus \{a_{\text{fix}}\}}{0/d_a}.
\]

(Recall that \( a_{\text{fix}} \in \Sigma \) above is the constant arbitrarily fixed at the end of Section 4.1.)

\textbf{Lemma 17 (preservation of meaning).} If \( \Sigma; t \triangleright (\cdot; e) \), then we have \#_{a_{\text{fix}}}(t) = \#_{e^\Sigma}\).

The above lemma follows from a usual substitution lemma (on internal variables) and a subject reduction property; see the full version for the proof.

The correctness of the transformation is stated as the following lemma.

\textbf{Lemma 18 (ordering reflection).} Let: \( \Sigma \) be an alphabet; \( a_{\text{fix}} \in \Sigma; \Gamma \) be an environment of the form

\[\Gamma = \varphi_1 : \kappa_1, \ldots, \varphi_m : \kappa_m, f_1 : \kappa_1 \rightarrow o, \ldots, f_\ell : \kappa_\ell \rightarrow o\]

where \( \text{order}(\kappa_i) = 2 \) and \( q_i \geq 0 \); \( t, t' \in \Lambda^0_{\Gamma; o} \); and

\[\Gamma; t \triangleright (\cdot; e) \quad \Gamma; t' \triangleright (\cdot; e').\]

Then we have:

\[t \triangleleft_{\text{order}} t' \quad \text{if} \quad e \triangleleft_{\text{order}} e'.\]

The proof of the above lemma is given in the full version, where we use Lemma 17 and substitution lemmas on external variables.
4.3 \( \preceq \) on order-2 terms is a wqo

The main goal of this subsection is to prove the following lemma.

Lemma 19 \((\preceq^{N}_{Γ,0} \text{ on order-2 terms is wqo})\). For \( Γ = f_{1} : α^{q_{1}} \to o, \ldots, f_{n} : α^{q_{n}} \to o \), the quasi-ordering \( \preceq^{N}_{Γ,0} \) on \( A^{Σ^{N}_{Γ,0}} \) is a wqo.

Lemma 15 follows as a corollary of Lemma 19 above and Lemma 18 in the previous subsection:

Proof of Lemma 15. Let \( t_{0}, t_{1}, \ldots \in A^{N}_{Γ,0} \) be an infinite sequence. We have the infinite sequence \( e_{0}, e_{1}, \ldots \in A^{Σ^{N}_{Γ,0}} \) such that \( Γ \vdash t_{i} \bowtie (;; e_{i}) \), and by Lemma 18, \( t_{i} \preceq^{#,Σ^{N}_{Γ,0}} t_{j} \) if \( e_{i} \preceq^{N}_{Γ,0} e_{j} \). By Lemma 19, there indeed exist \( i, j (i < j) \) such that \( e_{i} \preceq^{N}_{Γ,0} e_{j} \). Thus, we have \( t_{i} \preceq^{#,Σ^{N}_{Γ,0}} t_{j} \) as required. □

To prove Lemma 19, we restrict (without loss of generality) \( A^{Σ^{N}_{Γ,0}} \) to the set of \( β \)-normal forms (which we call \( \text{order-2 polynomials} \)), generated by the following grammar:

\[
P ::= 0 \mid 1 \mid P_{1} + P_{2} \mid P_{1} \times P_{2} \mid f_{1}P_{2} \cdots P_{q}
\]

Here, in \( f_{1}P_{2} \cdots P_{q}, f \) should have type \( α^{q} \to o \). We write \( P_{2}^{N} \) for the set of all order-2 polynomials, and write \( P_{2}^{N}_{Γ,0} \) for \( Λ^{Σ_{Γ,0}} \cap P_{2}^{N} \). Note that the arity of \( f \) may be \( 0 \), so that, for example, \( f_{1}(f_{2} \times (f_{2} + 1)) \in P_{2}^{N}_{f_{1}:o \to f_{2}:o,o,o} \). Thus, for Lemma 19, the following suffices:

Lemma 20 \((\preceq^{N}_{Γ,0} \text{ on order-2 polynomials is wqo})\). For \( Γ = f_{1} : α^{q_{1}} \to o, \ldots, f_{n} : α^{q_{n}} \to o \), the quasi-ordering \( \preceq^{N}_{Γ,0} \) on \( P_{2}^{N}_{Γ,0} \) is a wqo.

The idea for proving this lemma is as follows:

- An order-2 polynomial is regarded as a tree. Thus, by Kruskal’s tree theorem (Proposition 8), the set \( P_{2}^{N}_{Γ,0} \) is well-quasi-ordered with respect to the homeomorphic embedding \( \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} \). Unfortunately, however, the relation \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) does not necessarily imply \( \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} \); for example, if \( P_{1} = 1 \) and \( P_{2} = f_{1}(1) \), then \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) holds but \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) does not, because \( f_{1} \) may be instantiated to \( λx.0 \). Similarly for \( P_{1} = f_{2} \) and \( P_{2} = f_{2} \times 0 \).

- To address the problem above, we classify the values of \( f \in P_{2}^{N}_{Γ,0} \) (i.e. elements of \( Λ^{Σ_{0}:Γ} \)) into a finite number of equivalence classes \( A^{(1)}, \ldots, A^{(t)} \), and use the classification to further normalize order-2 polynomials, so that \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) implies \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) on the normalized polynomials. For example, in the case of \( P_{1} = 1 \) and \( P_{2} = f_{1}(1) \) above, the values of \( f_{1} \) are classified to (i) those that use the argument, (ii) those that return a positive constant without using the argument, and (iii) those that always return 0. We can then normalize \( P_{2} = f_{1}(1) \) to \( f_{1}(1) \) (in case (i)), \( f_{1}(0) \) (in case (ii)), and 0 (in case (iii)), respectively. (In case (ii), any argument is replaced with 0, because the argument is irrelevant.) Thus, we can indeed deduce \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) from \( P_{1} \preceq^{#,Σ_{0}:\Gamma}^{N}_{Γ,0} P_{2} \) when the value of \( f_{1} \) is restricted to just those in (i); and the same holds also for (ii) and (iii).

It follows that the restriction of the relation \( \preceq^{N}_{Γ,0} \) to each classification of the values of \( f_{1}, \ldots, f_{t} \in \text{dom}(Γ) \) is a wqo. Since the number of classifications is finite, by Dickson’s theorem (recall the sentence below Proposition 8), \( \preceq^{N}_{Γ,0} \) (which is the intersection of the restrictions of \( \preceq^{N}_{Γ,0} \) to the finite number of classifications) is also a wqo.

We first formalize and justify the reasoning in the last part (using Dickson’s theorem).
λ-Definable Order-3 Tree Functions are Well-Quasi-Ordered

Definition 21 (finite case analysis). For Γ = f₁ : κ₁,...,fₙ : κₙ, we call a finite case analysis of Γ a family (Aᵢ)ᵢ≤n,j∈Jᵢ of sets such that Aᵢ = ∪j∈JᵢAᵢ. For each i ≤ n, we define a quasi-ordering ≼₁(Γ,Aᵢ) on Aᵢ as follows:

\[ t ≼₁(Γ,Aᵢ), t' \iff \forall i \in A₁,...,Aₙ, \|t[i/f_i]\| ≤ \|t'[i/f_i]\| \]

We often omit the subscript Γ of ≼₁, and write ≼₁(Γ,Aᵢ).

The following lemma follows immediately from the fact that the intersection of a finite number of wqo’s is a wqo (which is in turn an immediate corollary of Dickson’s theorem).

Lemma 22. For Γ = f₁ : κ₁,...,fₙ : κₙ and a finite case analysis (Aᵢ)ᵢ≤n,j∈Jᵢ of Γ, if ≼₁(Γ,Aᵢ) is a wqo for any “case” (j)ᵢ≤n ∈ ∐i≤n Jᵢ, then so is ≼₁(Γ,Aᵢ).

Thus, to prove Lemma 20, it remains to find an appropriate decomposition Aₛₙ = ∪j≤Jₙ A_j (where κₙ is an order-1 type \(\sigma \to \sigma\)), and prove that ≼₁(Γ,A_j) is a wqo.

Henceforth we identify an element of Aₙ with the corresponding element of the polynomial semi-ring \(\mathbb{N}[x₁,...,xₙ]\). For example, \(\lambda x_1.\lambda x_2.((\lambda y.y)x_1) + x_2 \times x_3\) is identified with the polynomial \(x_1 + x_2^2\) (which is obtained by normalizing and omitting \(\lambda\)-abstractions, assuming a fixed ordering of the bound variables). For \(t \in Aₙ\) write \(\text{poly}(t)\) for the corresponding polynomial.

We define the equivalence relation \(\sim\) as the least semi-ring congruence relation on \(\mathbb{N}[x₁,...,xₙ]\) that satisfies (i) \(a \sim 1\) if \(a > 0\) and (ii) \(x₁^i \sim x₁\) if \(j > 0\). For example, \(2x₁^2x₂ + 3x₁x₃^2 + x₁ + 4 \sim x₁x₂ + x₁ + 1\), and the quotient set \(\mathbb{N}[x₁]/\sim\) consists of:

\[ [0]_=, [1]_=, [x₁]_=, [x₁ + 1]_=, \]

and \(\mathbb{N}[x₁,x₂]/\sim\) consists of

\[ [0]_=, [1]_=, [x₁]_=, [x₂]_=, [x₁x₂]_=, [1 + x₁]_=, [1 + x₂]_=, [1 + x₁x₂]_=, [x₁ + x₂]_=, ..., [1 + x₁ + x₂ + x₁x₂]_=. \]

Thus, for each case \(\Phi₁,...,\Phiₙ\) \(\in \mathcal{P}(\mathcal{P}(\{q\}))\) where \(\{q\}\) denotes \(\{1,\ldots,q\}\) and \(\mathcal{P}(X)\) denotes the powerset of \(X\), we can induce an equivalence relation on \(Aₙ\) from \(\sim\) on \(\mathbb{N}[x₁,...,xₙ]\), and let \(A_q\) be the equivalence class corresponding to \(\Phi\), i.e.,

\[ A_q := \{ t \in Aₙ \mid \text{poly}(t) \sim \sum_{(p₁,\ldots,pₙ)\in \Phi} x_{p₁} \cdots x_{pₙ} \}. \]

Then we have \(Aₙ = \bigcup_{\Phi \in \mathcal{P}(\mathcal{P}(\{q\}))} A_q\). Now, given Γ = f₁ : \(\sigma\to \sigma\),...,fₙ : \(\sigma\to \sigma\), we have obtained a finite case analysis of Γ as (Aₙ)₁≤n,\(\Phi \in \mathcal{P}(\mathcal{P}(\{q\}))\); for (\(\Phi_i\))₁≤n \(\in \prod_{i≤n} \mathcal{P}(\mathcal{P}(\{q\}))\), we write \(\sim_{(\Phi_i)}\) for \(\sim_{(A_q)}\). Thus it remains to show that \(\sim_{(\Phi_i)}\) is a wqo for each \(\Phi_i\) \(\in \prod_{i≤n} \mathcal{P}(\mathcal{P}(\{q\}))\).

The following lemma justifies the partition of polynomials based on \(\sim\).

Lemma 23 (zero/positive). For any Γ = f₁ : \(\sigma\to \sigma\),...,fₙ : \(\sigma\to \sigma\), \(\Phi_i\) \(\in \prod_{i≤n} \mathcal{P}(\mathcal{P}(\{q\}))\), and \(\Gamma \vdash P : \sigma\), we have either P \(\sim_{(\Phi_i)}\) 0 or 1 \(\sim_{(\Phi_i)}\) P.
In other words, the lemma above says that, given an order-2 polynomial \( P \), whether \( P[i_1, f_1, \ldots, i_n/f_n] \) evaluates to 0 or not is solely determined by the equivalence classes \( i_1, \ldots, i_n \) belong to.

**Example 24.** Let \( \Gamma := f : \sigma^2 \to o \), and \( \Phi := \{\emptyset, \{1,2\}\} \in \mathcal{P}(\mathcal{P}([2])) \), which denotes the equivalence class \([1 + x_1, x_2]_\infty \). We have \( \leq^N_{\Phi} f P_1 P_2 \) for any \( P_1 \) and \( P_2 \), since any element of the equivalence class is of the form \( a + \cdots \) for some natural number \( a \geq 1 \).

Based on the property above, we define the rewriting relation \( \rightarrow_{(\Phi_1)_i} \), to simplify order-2 polynomials by replacing (i) subterms that always evaluate to 0, and (ii) arguments of a function that are irrelevant, with \( 0 \).

**Definition 25** (rewriting relation and \((\Phi_1)_i\)-normal form). For \( \Gamma = f_1 : \sigma^{q_1} \to o, \ldots, f_n : \sigma^{q_n} \to o \) and \( \Phi := \{\emptyset, \{1,2\}\} \in \prod_{i \leq n} \mathcal{P}(\{q_i\}) \), we define the relation \( \rightarrow_{(\Phi_1)_i} \) by the following two rules.

\[
\begin{align*}
P & \rightarrow_{(\Phi_1)_i} 0 \text{ if } P \leq^N_{(\Phi_1)_i} 0 \text{ and } P \neq 0. \\
f_\ell P_1 \cdots P_{q_t} & \rightarrow_{(\Phi_1)_i} f_\ell P_1 \cdots P_{k-1} 0 P_{k+1} \cdots P_{q_t} \text{ if (i) } P_k \neq 0 \text{ and (ii) for all } \phi \in \Phi_\ell \text{ such that } k \in \phi, \text{ there exists } p \in \phi \text{ such that } P_p \leq^N_{(\Phi_1)_i} 0.
\end{align*}
\]

We write \( P_0 \rightarrow_{(\Phi_1)_i} P_1 \) if \( P_1 = E[P'_0] \) and \( P'_0 \rightarrow_{(\Phi_1)_i} P'_1 \) for some \( E, P'_0 \) and \( P'_1 \), where the evaluation context \( E \) is defined by:

\[
E ::= [] | E + P | P + E | E \times P | P \times E | f P_1 \ldots P_{i-1} E P_{i+1} \ldots P_q.
\]

We call a normal form \( \rightarrow_{(\Phi_1)_i} \) a \((\Phi_1)_i\)-normal form.

Intuitively, the condition (ii) in the second rule says that whenever the \( k \)-th argument \( P_k \) is used by \( f_\ell \), it occurs only in the form of \( P_k \times P_p \times \cdots \) (up to equivalence) and \( P_p \) always evaluates to 0; thus, the value of \( P_k \) is actually irrelevant.

**Example 26.** We continue Example 24. Recall \( \Gamma = f : \sigma^2 \to o \) and \( \Phi = \{\emptyset, \{1,2\}\} \). Consider the order-2 polynomial \( f_1 (1 \times 0) \). It can be rewritten to \( f_1 0 \) by using the first rule (and the evaluation context \( E = f_1 [] \)). We can further apply the second rule to obtain \( f_1 0 \rightarrow_{\Phi} f_0 0 \), because \( k = 1 \) satisfies the conditions ((i) and (ii)). In fact, if \( 1 \in \phi \in \Phi \), then \( \phi = \{1,2\} \); hence, the required condition holds for \( p = 2 \). Note that \( f_0 \) is a \( \Phi \)-normal form; the first rule is not applicable, as \( f_0 0 \leq^N_{(\Phi_1)_i} 0 \) by the discussion in Example 24.

The following lemma guarantees that any order-2 polynomial can be transformed to at least one equivalent \((\Phi_1)_i\)-normal form.

**Lemma 27** (existence of normal form).

1. \( \rightarrow_{(\Phi_1)_i} \) is strongly normalizing.
2. If \( P \rightarrow_{(\Phi_1)_i} P' \) then \( P \approx^N_{(\Phi_1)_i} P' \).

We can reduce the wnqss of \( \leq^N_{(\Phi_1)_i} \) to that of \( \leq^N_{(\Phi_1)_i} \) by the following lemma:

**Lemma 28.** For \( \Gamma = f_1 : \sigma^{q_1} \to o, \ldots, f_n : \sigma^{q_n} \to o, \Phi := \{\emptyset, \{1,2\}\} \in \prod_{i \leq n} \mathcal{P}([q_i]) \), and \((\Phi_1)_i\)-normal forms \( \Gamma \vdash P, P : o \), if \( P \rightarrow_{(\Phi_1)_i} P' \) then \( P' \approx^N_{(\Phi_1)_i} P \).

The proof is given by a simple calculation using Lemma 23 and that the given \((\Phi_1)_i\)-normal forms \( P', P \) do not satisfy the condition for the rewriting \( \rightarrow_{(\Phi_1)_i} \).

Now we are ready to prove Lemma 20.
**Proof of Lemma 20.** By Lemma 22, it suffices to show that $\preceq^N_{\Phi_i}$ on $\prod_{i \leq n} P(P([q_i]))$ is a wqo for each $(\Phi_i)_i \in \prod_{i \leq n} P(P([q_i]))$. By the Kruskal’s tree theorem, $\preceq^T_{\beta,\Sigma_0\cup\Gamma}$ on $\prod_{i \in \omega} \mathcal{P}^N_{\Gamma_i}$ is a wqo, and hence the sub-ordering $\preceq^N_{\beta,\Sigma_0\cup\Gamma}$ on the subset

$$\{P \in \prod_{i \in \omega} \mathcal{P}^N_{\Gamma_i} \mid P \text{ is a } (\Phi_i)_i\text{-normal form}\} \subseteq \prod_{i \in \omega} \mathcal{P}^N_{\Gamma_i}$$

is a wqo. Therefore by Lemma 28, $\preceq^N_{\Phi_i}$ on $\{P \in \prod_{i \leq n} \mathcal{P}^N_{\Gamma_i} \mid P \text{ is a } (\Phi_i)_i\text{-normal form}\}$ is a wqo. By Lemma 27, $\{P \in \prod_{i \leq n} \mathcal{P}^N_{\Gamma_i} \mid P \text{ is a } (\Phi_i)_i\text{-normal form}\}$ and $\prod_{i \leq n} \mathcal{P}^N_{\Gamma_i}$ – both modulo $\beta\eta$-equivalence – are isomorphic (with respect to $\preceq^N_{\Phi_i}$ and $\preceq^N_{(\Phi_i)_i}$); hence $\preceq^N_{\Phi_i}$ on $\prod_{i \leq n} \mathcal{P}^N_{\Gamma_i}$ is a wqo. ◼

## 5 Conclusion

We have introduced the nAK-conjecture, a weaker version of the AK-conjecture in [2], and proved it up to order 3. We have also proved a pumping lemma for higher-order grammars (which is slightly weaker than the pumping lemma conjectured in [2]) under the assumption that the nAK-conjecture holds. Obvious future work is to show the nAK-conjecture or the original AK-conjecture for arbitrary orders. Finding other applications of the two conjectures (cf. an application of Kruskal’s tree theorem to program termination [4]) is also left for future work.

### References


