ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 318 (2004) 373–408

Theoretical Computer Science

www.elsevier.com/locate/tcs

The decidability of a fragment of BB′IW-logic

Sabine Broda[a,*], Luís Damas[a], Marcelo Finger[b], Paulo Silva e Silva[b]

[a] *DCC-FC, Universidade do Porto, Rua do Campo Alegre 823, 4150 Porto, Portugal*

[b] *Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil*

Received 11 November 2002; received in revised form 19 January 2004; accepted 5 February 2004
Communicated by D. Plotkin

Abstract

Despite its simple formulation, the decidability of the logic BB′IW has remained an open problem. We present here a decision procedure for a fragment of it, called the arity-1 formulas.

The decidability proof is based on a representation of formulas called formula-trees, which is coupled with a proof method that computes long normal λ-terms that inhabit a formula.

A rewriting-system is associated with such λ-terms, and we show that a formula admits a BB′IW-λ-term if and only if the associated rewriting-system terminates. The fact that termination is decidable is proved using a result on the finiteness of non-ascending sequences of n-tuples in $\mathbb{N}^n$, which is equivalent to Kripke's Lemma.

1. Introduction

The Hilbert-style logic $T_{\to}$ of "ticket entailment", introduced and motivated by Anderson and Belnap [1], is the system of implicational propositional logic based on the axiom schemes

$$B : (\alpha \to \beta) \to (\gamma \to \alpha) \to \gamma \to \beta,$$

$$B' : (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma,$$

$$I : \alpha \to \alpha,$$

$$W : (\alpha \to \alpha \to \beta) \to \alpha \to \beta$$

* Corresponding author.
E-mail address: sbb@ncc.up.pt (S. Broda).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.02.002

and the rule

$$\frac{\alpha \to \beta \ \alpha}{\beta} \quad (\to e).$$

Surprisingly, the problem of decidability of this logic remains unsolved, although several studies have been carried out leading to a better insight on its nature.

Due to the fact that the axiom schemes of $T_{\to}$, or $\mathsf{BB'IW}$-logic, are the principal types of

$$\mathbf{B} \ : \ \lambda xyz.x(yz),$$

$$\mathbf{B'} \ : \ \lambda xyz.y(xz),$$

$$\mathbf{I} \ : \ \lambda x.x,$$

$$\mathbf{W} \ : \ \lambda xy.xyy,$$

the problem is in fact equivalent to finding an algorithm to decide whether a given type/formula has any inhabitants that are applicative combinations of these four terms. Also $\mathsf{BB'IW}$-proofs are represented by closed $\mathsf{BB'IW}$-definable λ-terms. In [12], a precise characterisation of the class of $\mathsf{BB'IW}$-abstractable terms was presented. That approach was developed in terms of combinators but it can easily be transposed to λ-terms, as in [5], leading to the decidable class of $\mathsf{BB'IW}$-λ-terms. Consequently, a type/formula τ is a theorem of $T_{\to}$ iff there is a $\mathsf{BB'IW}$-λ-term to which the type τ can be assigned. Based on this characterisation, Bunder presented an algorithm in [5] that, applied to a type τ, produces in finite time a (long) normal $\mathsf{BB'IW}$-inhabitant if there is one, but that may run forever if τ is not $\mathsf{BB'IW}$-inhabited. So, his algorithm does not, in general, provide a decision procedure.

In this paper, we solve the problem of decidability for a restricted class of formulas, called arity-1 formulas. In Section 2, we fix the notation and point to some results and methods that will be used in the remainder of the paper. We then present the class of arity-1 formulas, a fragment of the $\to$-language. In Section 3, we define the notion of terminating rewriting-systems over typed sequences that will be used to prove $\mathsf{BB'IW}$ decidability for arity-1 formulas. Our proof is based on a mapping $\mathscr{R}(\_)$ from arity-1 formulas to rewriting-systems. It will be shown in Section 4 that an arity-1 formula τ is $\mathsf{BB'IW}$-inhabited if and only if the rewriting-system $\mathscr{R}(\tau)$ is terminating. Decidability of termination of rewriting-systems over typed sequences is proved in Section 5 using a result on non-ascending $\mathbb{N}^n$-sequences equivalent to Kripke's Lemma, cf. [1].

2. Preliminaries

We assume familiarity with the basic notions in λ-calculus and use standard notation from [2] and [8]. Our notation differs from that in [2], since we denote type-variables (atoms) by "$\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots$" and arbitrary types by lower-case Greek letters. For type

assignment, we consider the system $\mathbf{TA}_\lambda$ of simply typed λ-calculus à la Curry (for an introduction see [8] or [2]). A term M has a *bound-variable clash* iff M contains an abstractor λx and a occurrence of x that is not in its scope. Note that for any λ-term M exists a λ-term $N =_\alpha M$ without bound-variable clashes. In this paper we will generally, but for the result of Algorithm 2.18, consider λ-terms without bound-variable clashes.

2.1. BB′IW-λ-terms

In the following, we describe the set of λ-terms that can be defined in terms of the combinators **B**, **B′**, **I** and **W**. We call these terms BB′IW-λ-terms since they represent BB′IW-proofs. Furthermore, a formula τ is a BB′IW-theorem if and only if there is some BB′IW-λ-term to which the type τ can be assigned.

Definition 2.1. Let $\vec{x} = x_1, \ldots, x_n$, where $n \geqslant 0$, be a finite sequence of distinct variables and let ε represent the empty sequence. The implicit order $\prec$ induced by a constant $\perp$ and $\vec{x}$ is $\perp \prec x_1 \prec \cdots \prec x_n$. The $\vec{x}$-*index* of a λ-term P, $Ind_{\vec{x}}(P)$, is the $\prec$-maximum of $x_1, \ldots, x_n$ in $FV(P)$, or $\perp$ if none of $x_1, \ldots, x_n$ occurs in $FV(P)$.

Definition 2.2. The set of *hereditary right-maximal terms* relative to $\vec{x}$, $HRM_{\vec{x}}$, is defined as:
- every variable is in $HRM_{\vec{x}}$;
- if $P \in HRM_{\vec{x}y}$ and $y \in FV(P)$, then $\lambda y.P \in HRM_{\vec{x}}$;
- if $P, Q \in HRM_{\vec{x}}$ and $Ind_{\vec{x}}(P) \preccurlyeq Ind_{\vec{x}}(Q)$, then $PQ \in HRM_{\vec{x}}$.

The notion of hereditary right-maximal terms was introduced in [12], and we note that our definition of $HRM_{\vec{x}}$ corresponds to $HRM_{(1,\ldots,n)} \cap Once^{+}_{(1,\ldots,n)}$ in that work. The following characterisation of the set of BB′IW-λ-terms results mainly from the work in [12].

Proposition 2.3. *A closed λ-term P is a* BB′IW-*term if and only if $P \in HRM_\varepsilon$.*

A BB′IW-term to which a type τ can be assigned is called a BB′IW-inhabitant of τ. The following result, together with the subject-reduction property, reduces the existence of BB′IW-inhabitants to the case of β-normal forms.

Property 2.4. *Consider a* BB′IW-*term M and a term N such that $M \to_\beta N$. Then, N is a* BB′IW-*term.*

Proof. It is sufficient to show that for any sequence $\vec{x}$ and redex $(\lambda y.P)Q \in HRM_{\vec{x}}$ we have $FV((\lambda y.P)Q) = FV(P[Q/y])$ and $P[Q/y] \in HRM_{\vec{x}}$. The former is trivial since $y \in FV(P)$, and for the latter we will show that for every subterm M of P such that $M \in HRM_{\vec{x}y\vec{z}}$, for some sequence of variables $\vec{z}$, one has $M[Q/y] \in HRM_{\vec{x}\,\vec{z}}$. Then, since $P \in HRM_{\vec{x}y}$, it follows that $P[Q/y] \in HRM_{\vec{x}}$.

First note that $Q \in HRM_{\vec{x}}$ and no element of $\vec{z}$ occurs in it. Now, we proceed by structural induction on M. The result is trivial if M is a variable. Now, suppose that $M = (P_1 P_2) \in HRM_{\vec{x}y\vec{z}}$, hence $P_1, P_2 \in HRM_{\vec{x}y\vec{z}}$ and $Ind_{\vec{x}y\vec{z}}(P_1) \leqslant Ind_{\vec{x}y\vec{z}}(P_2)$. By the induction hypothesis, $P_1[Q/y], P_2[Q/y] \in HRM_{\vec{x}\vec{z}}$. Thus, in order to prove that $(P_1 P_2)[Q/y] \in HRM_{\vec{x}\vec{z}}$ it suffices to show that $Ind_{\vec{x}\vec{z}}(P_1[Q/y]) \leqslant Ind_{\vec{x}\vec{z}}(P_2[Q/y])$. For that, consider the cases:

- $Ind_{\vec{x}\vec{z}}(P_1[Q/y]) = \bot$. The result is trivial.
- $Ind_{\vec{x}\vec{z}}(P_1[Q/y]) \in \vec{x}$. Then no $z \in \vec{z}$ occurs in P_1 and from $Ind_{\vec{x}y\vec{z}}(P_1) \leqslant Ind_{\vec{x}y\vec{z}}(P_2)$ we obtain the result.
- $Ind_{\vec{x}\vec{z}}(P_1[Q/y]) = z_i \in \vec{z}$. Since no $z \in \vec{z}$ occurs in Q, it follows from $Ind_{\vec{x}y\vec{z}}(P_1) \leqslant Ind_{\vec{x}y\vec{z}}(P_2)$ that $Ind_{\vec{x}\vec{z}}(P_2[Q/y]) = z_j$ and $z_i \leqslant z_j$, which gives us the result.

Finally, if $M = \lambda u.P_1 \in HRM_{\vec{x}y\vec{z}}$, then $P_1 \in HRM_{\vec{x}y\vec{z}u}$ and $u \in FV(P_1)$. Thus, by the induction hypothesis, $P_1[Q/y] \in HRM_{\vec{x}\vec{z}u}$. Then, it follows from $u \in FV(P_1[Q/y])$ and from Definition 2.2 that $(\lambda u.P_1)[Q/y] = (\lambda u.(P_1[Q/y]) \in HRM_{\vec{x}\vec{z}}$. □

A β-normal inhabitant M of a type τ is called a long normal inhabitant of τ iff every variable-occurrence z in M is followed by the longest sequence of arguments allowed by its type, i.e. iff each component with form $(zP_1 \ldots P_n)$, $(n \geqslant 0)$, that is not in a function position has atomic type. The finite set of all terms obtained by η-reducing a λ-term M zero or more times is called the η-family of M and denoted by $\{M\}_\eta$. It has been shown (cf. [3,8]) that the η-families of the long normal inhabitants of τ partition the set of normal inhabitants of τ into non-overlapping finite subsets, each η-family containing just one long member. Furthermore, Ben-Yelles (cf. [3,8]) showed that every normal inhabitant of a type τ can be η-expanded to one unique (up to α-conversion) long normal inhabitant of τ. A simple expansion-algorithm can be found in [8]. The following result implies that when looking for β-normal $\mathsf{BB'IW}$-inhabitants of a type, one can just search for long normal $\mathsf{BB'IW}$-inhabitants from which all other β-normal inhabitants can be obtained by η-reduction.

Property 2.5. *Consider two λ-terms M and N such that $M \to_\eta N$. Then, M is a $\mathsf{BB'IW}$-term if and only if N is.*

Proof. Let $\vec{x}$ be a sequence of variables and $\lambda y.Py$ a term such that $y \notin FV(P)$. Then, $FV(\lambda y.Py) = FV(P)$ and consequently $Ind_{\vec{x}}(\lambda y.Py) = Ind_{\vec{x}}(P)$. On the other hand, $\lambda y.Py \in HRM_{\vec{x}}$ if and only if $Py \in HRM_{\vec{x}y}$ and since $y \notin FV(P)$ this holds if and only if $P \in HRM_{\vec{x}}$. □

In the following, we describe the straight relation that exists between subterms and variables in a long normal inhabitant of a type φ and the subtypes of φ. It is well known that for every (long) normal inhabitant M of a type φ there is exactly one deduction in the system $\mathbf{TA}_\lambda$ that assigns the type φ to M. Thus, in the remaining we sometimes refer to the types that are assigned to variables and subterms of a normal inhabitant during this unique deduction as *their* types. We begin recalling the rather standard definition of polarities of occurrences of subtypes/subformulas.

Definition 2.6. An occurrence of a subtype in a type is defined as *positive* or *negative* as follows:

- φ occurs positively in φ;
- if an occurrence of a subtype is positive (negative) in φ, then it is negative (resp. positive) in $\varphi \to \tau$;
- if an occurrence of a subtype is positive (negative) in τ, then it is positive (resp. negative) in $\varphi \to \tau$.

The following properties of long normal inhabitants follow almost directly from the definitions and give us some insight on the relation between the structure of a type and the structure of their long normal inhabitants (an exhaustive analysis of this relationship can be found in [8], cf. Section 8E—The structure of an *nf*-scheme).

Lemma 2.7. *Consider a long normal inhabitant M of a type φ and let P be a subterm of M.*

(i) *If $P = \lambda x_1 \dots x_n.N$, with $n \geqslant 1$ and such that N is no abstraction, then there is some positive occurrence of a subtype of the form $\tau = \alpha_1 \to \cdots \to \alpha_n \to A$ in φ and P is of type τ. Furthermore, $x_1, \dots, x_n$ are respectively of type $\alpha_1, \dots, \alpha_n$, which are negative occurrences of subtypes of φ, while N is of atomic type A.*

(ii) *If $P = yM_1 \dots M_n$, with $n \geqslant 0$ and such that $M_1, \dots, M_n$ are all the arguments applied to y, then there is some negative occurrence of a subtype of the form $\tau = \alpha_1 \to \cdots \to \alpha_n \to A$ in φ, y has type τ and $P, M_1, \dots, M_n$ have respectively types $A, \alpha_1, \dots, \alpha_n$.*

We conclude mainly that variables in abstraction sequences correspond to negative occurrences of subtypes and that subterms in argument position correspond to positive occurrences of subtypes. Also, every variable x with a type $\alpha_1 \to \cdots \to \alpha_n \to A$ occurs always with exactly n arguments in a long normal inhabitant, i.e. in an expression of the form $xP_1 \dots P_n$ of type A and such that $P_1, \dots, P_n$ are, respectively, of types $\alpha_1, \dots, \alpha_n$. Moreover, if $\alpha_i = \beta_1 \to \cdots \to \beta_k \to B$, $i \in \{1, \dots, n\}$, then P_i is of the form $\lambda y_1 \dots y_k.N$ such that $y_1, \dots, y_k$ have types $\beta_1, \dots, \beta_k$ which are negative occurrences of subtypes, and N has type B.

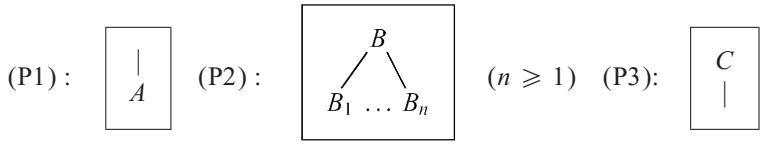
2.2. The formula-tree proof method

A central element in our decidability proof is the tree-like representation of types called *formula-tree* representation, first introduced in [4]. [1] In this representation, every type φ is split into primitive parts, and the primitive parts themselves form a tree-like structure, which is called the *formula-tree* of φ, and that defines some kind of hierarchy over the primitive parts of the formula. Here, every primitive part, except the

[1] A short presentation of the method together with an implementation as a Java applet can be found at http://www.ncc.up.pt/~sbb/FTLab/ftlab.

one in the root of the formula-tree, represents together with the subtree rooted in it a subtype of φ that, during the construction of a long normal inhabitant of φ, may be assigned to variables.[2] The formula-tree of φ, $\mathtt{tree}(\varphi)$, can then be used to build *proof-trees* for φ, each proof-tree being a representation of a finite non-empty set of long normal inhabitants of φ. Since φ may have none, a finite or an infinite number of long normal inhabitants it may consequently be possible to build none, a finite or even an infinite number of different proof-trees for φ.

Primitive parts (frequently represented inside of boxes) are items of either one of the following forms (P1), (P2) or (P3), where $A, B, B_1, \ldots, B_n, C$ denote type-variables:



Here, $A, B_1, \ldots, B_n$ are called the *tail-variables* of the respective primitive part, while B and C are *head-variables*. The *arity* of a primitive part is the number of its tail-variables. Moreover, we will always associate with each primitive part a unique label p_i which allow us to have distinct primitive parts with the same appearance.

Definition 2.8. A formula-tree is a tree-like structure with primitive parts as nodes, but such that subtrees descend from the branches (or tail-variables) of primitive parts rather than from the whole primitive parts as nodes. Then, such a structure is called a *formula-tree* iff

- the root of the formula-tree is the only primitive part of form (P1);
- every node of form (P2) or (P3) in the formula-tree descends from a tail-variable in another primitive part;
- every (labeled) primitive part occurs only once in a formula-tree.

The following algorithm computes the formula-tree $\mathtt{tree}(\varphi)$ of a type φ. We use dashed lines for the edges of the formula-tree in order to distinguish them from the edges in the primitive parts (nodes) of the formula-tree. Furthermore, we will use the expression *branch* only when referring to edges in primitive parts, but not when referring to (dashed) edges of formula-trees.

Formula-tree Algorithm 2.9.

Input: A type $\varphi = \alpha_1 \to \cdots \to \alpha_n \to A$, where A is an atom and $n \geqslant 0$.

Output: φ's formula-tree $\mathtt{tree}(\varphi)$.

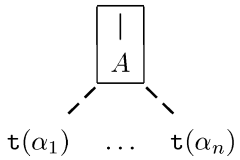
The formula-tree $\mathtt{tree}(\varphi)$ is given by the following.

- If $n = 0$, i.e. $\varphi \equiv A$, then $\mathtt{tree}(\varphi) =$ 

.

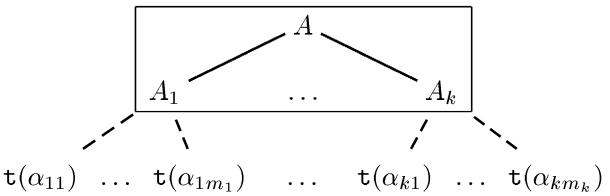
[2] In [8], these subtypes are also called negative subpremises.

- If $n \geqslant 1$, then $\mathtt{tree}(\varphi) =$

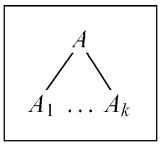


where $\mathtt{t}(A) =$

and for $k \geqslant 1$ and $m_1, \ldots, m_k \geqslant 0$ we recursively define
$\mathtt{t}((\alpha_{11} \to \cdots \to \alpha_{1m_1} \to A_1) \to \cdots \to (\alpha_{k1} \to \cdots \to \alpha_{km_k} \to A_k) \to A) =$



Furthermore, for $1 \leqslant j \leqslant k$, we call the primitive parts in the top of $\mathtt{t}(\alpha_{j1}), \ldots, \mathtt{t}(\alpha_{jm_j})$ the *descendents* of primitive part



at branch j (or tail-variable A_j). Note, that in case $m_j = 0$, branch j, i.e. tail-variable A_j, has no children/descendents.

We assume that all primitive parts introduced above are given distinct labels, thus ensuring that each primitive part occurs only once as required in the definition of a formula-tree.

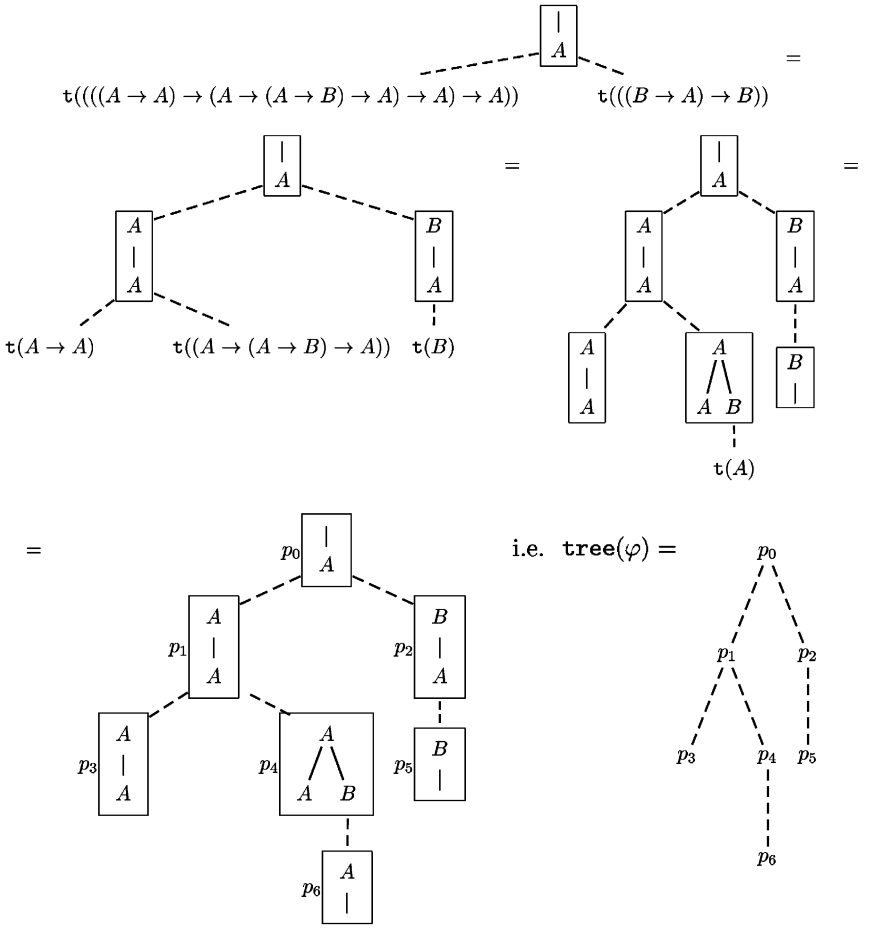
In the following we exemplify the formula-tree algorithm as well as the notion of *descendents of a primitive part at a branch* (*tail-variable*).

Example 2.10. The formula

$$\varphi = (((A \to A) \to (A \to (A \to B) \to A) \to A) \to A) \to ((B \to A) \to B) \to A$$

has formula-tree

$$\mathtt{tree}(\varphi) =$$
$$\mathtt{tree}((((A \to A) \to (A \to (A \to B) \to A) \to A) \to A) \to ((B \to A) \to B) \to A) =$$



with primitive parts

$$p_0 = \boxed{\begin{array}{c}|\\A\end{array}},\quad p_1 = \boxed{\begin{array}{c}A\\|\\A\end{array}},\quad p_2 = \boxed{\begin{array}{c}B\\|\\A\end{array}},\quad p_3 = \boxed{\begin{array}{c}A\\|\\A\end{array}},\quad p_4 = \boxed{\begin{array}{c}A\\/\ \backslash\\A\quad B\end{array}},$$

$$p_5 = \boxed{\begin{array}{c}B\\|\end{array}} \quad \text{and} \quad p_6 = \boxed{\begin{array}{c}A\\|\end{array}}.$$

In the formula-tree $\mathtt{tree}(\varphi)$, p_0 has descendents p_1 and p_2 at p_0's branch 1 (p_0's only branch), p_1 has descendents p_3 and p_4 at p_1's branch 1 (p_1's only branch), p_2 has descendent p_5 at its only branch 1, p_4 has descendent p_6 at branch 2 and no descendent at branch 1, and finally p_3, p_5 and p_6 have no descendents at all.

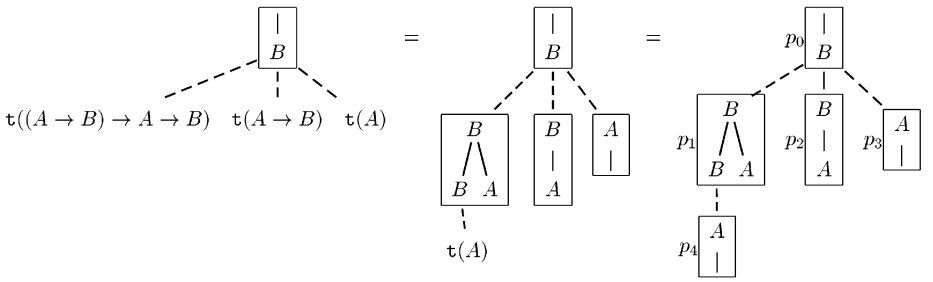
It is easy to see that for every negative occurrence of a subtype α of a type φ there will be a subtree $\mathtt{t}(\alpha)$ of $\mathtt{tree}(\varphi)$, suggesting the following relation between subtrees and the types of variables in long normal inhabitants (cf. Lemma 2.7): given a long normal inhabitant M of a type φ and a variable x in M of type α there is exactly one subtree $\mathtt{t}(\alpha)$ of $\mathtt{tree}(\varphi)$ corresponding to x. Any such subtree can be identified by the primitive part in its top position. So, from now on we call this *the primitive part corresponding to the variable x in M*. Note that, while every variable in M has exactly one corresponding primitive part in $\mathtt{tree}(\varphi)$, given a primitive part p in $\mathtt{tree}(\varphi)$ there may be more than one variable in M for which p is the corresponding primitive part, or there may even be no such variable. The result below follows from the definition of *corresponding primitive part* and from the observations made after Lemma 2.7.

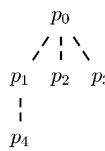
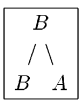
Lemma 2.11. *Let M be a long normal inhabitant of a type φ with formula-tree* $\mathtt{tree}(\varphi)$, *consider any variable x in M and let p be the primitive part in* $\mathtt{tree}(\varphi)$ *corresponding to x. If p is of arity $k \geqslant 0$, then x occurs always with exactly k arguments in M. Furthermore, for $k \geqslant 1$, if $i \in \{1,\ldots,k\}$ and p has descendents $p_1,\ldots,p_m$ at branch/tail-variable i in* $\mathtt{tree}(\varphi)$, *then the ith argument of x is of the form $\lambda y_1 \ldots y_m.N$ and the primitive parts in* $\mathtt{tree}(\varphi)$ *corresponding to $y_1,\ldots,y_m$ are, respectively, $p_1,\ldots,p_m$.*

Furthermore, one can easily define an inverse algorithm, which given a formula-tree FT computes the unique type φ such that $\mathtt{tree}(\varphi) = \mathtt{FT}$. In the following example, we give some intuition on the meaning of primitive parts and the hierarchy defined on them.

Example 2.12. The formula $\varphi = ((A \to B) \to A \to B) \to (A \to B) \to A \to B$ has formula-tree

$$\mathtt{tree}(\varphi) = \mathtt{tree}(((A \to B) \to A \to B) \to (A \to B) \to A \to B) =$$

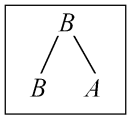


i.e. $\mathtt{tree}(\varphi) =$  with primitive parts $p_0 =$  , $p_1 =$  , $p_2 =$  ,

$p_3 =$  and $p_4 =$  .

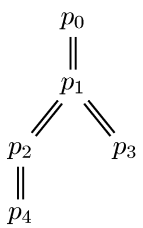
Note that part p_4 connects to the branch 1 of part p_1 with tail variable B.

Then we get the following information from this representation: during the construction of a long normal inhabitant of φ there may appear variables of type $(A \to B) \to A \to B$, of type $A \to B$ or of type A, which are the negative subtypes of φ, cf. Lemma 2.7. These are represented, respectively, by the subtrees $\mathtt{t}((A \to B) \to A \to B)$, $\mathtt{t}(A \to B)$ and $\mathtt{t}(A)$ (the latter occurs twice). As such, using a variable x of type $(A \to B) \to A \to B$, corresponding to the subtree $\mathtt{t}((A \to B) \to A \to B)$ with primitive part

$p_1 =$ 

at its top, allows us to construct a term of type B—the head-variable of p_1. Furthermore, in order to complete the construction of this term, two other terms of type B and of type A, respectively,—the tail-variables of p_1—have to be constructed. Also x will appear with two—the arity of p_1—arguments, and for the construction of the first there will appear a new variable of type A represented by $\mathtt{t}(A)$, i.e. corresponding to p_4—the descendent of the first branch of p_1. Actually, all this parallels the fact, cf. Lemma 2.7, that every long normal term beginning with a variable x of type $(A \to B) \to A \to B$ is of the form $x(\lambda y.M)N$ of type B, where y, M and N are, respectively, of type A, B and A, and finally in M there may appear a new variable y of type A.

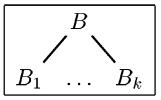
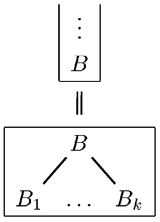
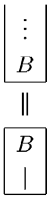
The previous discussion illustrates that we can associate to each variable in a long normal inhabitant M of a type φ a subtree $\mathtt{t}(\alpha_i)$ of $\mathtt{tree}(\varphi)$ with primitive part p_i at its top (the corresponding primitive part). Now, consider M' obtained from M by replacing each variable in M with the name of the corresponding primitive part and by erasing abstractions. For example, for the long normal inhabitant $M = \lambda xyz.x(\lambda w.yw)z$ of φ as above we obtain $M' = p_1(p_2\, p_4)\, p_3$, which represented as a tree gives, after inserting a top level node p_0, the following tree of primitive parts



It turns out that one can characterise precisely the trees of primitive parts that correspond to long normal inhabitants of a type and which will be called *valid proof-trees*. We now describe the set of rules that allows us to build proof-trees and then give two conditions to be satisfied by a proof-tree in order for it to be valid. In the remaining we will often use the notation $\bar{o}$ to refer to a specific occurrence of an object o.

Definition 2.13 (Proof-tree structure). A *proof-tree* for a type φ is a tree-like structures whose nodes are primitive parts in $\mathtt{pp}(\varphi)=\{p_0,\ldots,p_n\}$, the set of (labels of the) primitive parts in the formula-tree of φ and where p_0 denotes the primitive part in the root of $\mathtt{tree}(\varphi)$ (thus the only primitive part of form (P1) in $\mathtt{pp}(\varphi)$). In the following, we shall use the notation  to designate a proof-tree that has in a leaf a primitive part with a tail-variable A.

Then, the set of proof-trees for φ is given by the following:

- If $p_0 =$  $\in \mathtt{pp}(\varphi)$, then  is a proof-tree for φ;
- if  is a proof-tree for φ and  $\in \mathtt{pp}(\varphi)$ where $k\geqslant 1$, then  is a proof-tree for φ;
- if  is a proof-tree for φ and  $\in \mathtt{pp}(\varphi)$, then  is a proof-tree for φ.

Definition 2.14 (Valid proof-trees). Given a proof-tree PT for a type φ, we call it a *valid proof-tree* for φ iff

- the number of subtrees rooted in any node/primitive part in PT equals the arity of that primitive part;
- if

$$p' = \boxed{\begin{array}{c} B \\ B_1 \quad \ldots \quad B_k \end{array}}$$

with $k\geqslant 1$ is a primitive part in the formula-tree $\mathtt{tree}(\varphi)$, and p is a descendent of p' in $\mathtt{tree}(\varphi)$ at branch i, with $1\leqslant i\leqslant k$, then above every occurrence $\bar{p}$ of p in PT there is at least one occurrence $\bar{p}'$ of p' in PT, such that the occurrence $\bar{p}$ occurs in the subtree of PT rooted in the i'th branch of $\bar{p}'$.

This last item corresponds to the hierarchy of $\mathtt{tree}(\varphi)$ being imposed on the construction of a valid proof-tree and will be illustrated by the following example. From

now on, we will usually, and for the sake of saving space, represent *valid proof-trees* without drawing edges between primitive parts explicitly. As such, the tree of primitive parts at the end of Example 2.12 will be represented by

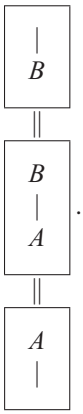
$$\begin{array}{c} p_0 \\ p_1 \\ p_2 \;\; p_3 \\ p_4 \end{array}.$$

Due to the first condition in the definition of a valid proof-tree (Definition 2.14) there will be no ambiguity regarding descendents/parents in the tree. Also, for the sake of saving space, we will *unify* the head variable and tail variables of two parts that are linked in a proof-tree.

Example 2.15. A valid proof-tree for the formula φ in Example 2.12 is

$$\begin{array}{c} p_0 \\ \| \\ p_2 \\ \| \\ p_3 \end{array}$$

i.e.



Using the compact representation this proof-tree will be represented by $\begin{array}{c} p_0 \\ p_2 \\ p_3 \end{array}$ and has appearance

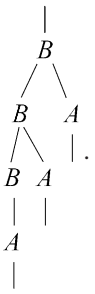
after unifying/overlapping the linked variables. On the other hand, the tree $\begin{array}{c} p_0 \\ p_2 \\ p_4 \end{array}$ is not a valid proof-tree for φ (though it would have the same appearance), since it does not respect the hierarchy given by the formula-tree of φ, which requires that in a valid proof-tree all occurrences of p_4 occur in some subtree that is rooted in the left branch of an occurrence of p_1. Moreover, there are exactly three proof-trees with appearance

, that are $\begin{array}{c} p_0 \\ p_1 \\ p_2 \;\; p_3 \\ p_4 \end{array}$, $\begin{array}{c} p_0 \\ p_1 \\ p_2 \;\; p_3 \\ p_3 \end{array}$ and $\begin{array}{c} p_0 \\ p_1 \\ p_2 \;\; p_4 \\ p_4 \end{array}$.

Only the first two of these proof-trees are valid proof-trees, since the third does again not satisfy the second condition of Definition 2.14.
Another valid proof-tree is

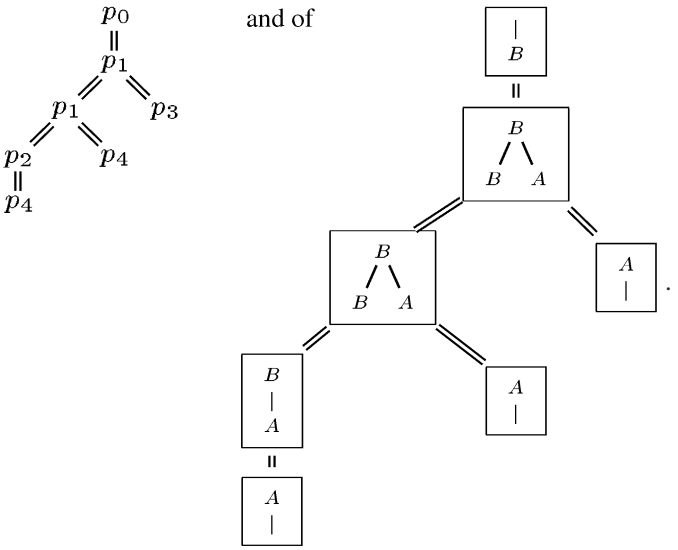
$$\begin{array}{c} p_0 \\ p_1 \\ p_1 \;\; p_3 \\ p_2 \;\; p_4 \\ p_4 \end{array}$$

with appearance



.

Note that these again are, respectively, the compact representations of



The following example will give some intuition on the fact that, given a formula φ, the search for a valid proof-tree, and consequently the search for normal inhabitants, is finite. A sketch of a search-algorithm is given after the example.

Example 2.16. Consider again the formula φ and its formula-tree $\mathtt{tree}(\varphi)$ from Example 2.10. We will now attempt to build a valid proof-tree for this formula, applying the rules for constructing proof-trees, in such a way that none of the conditions in Definition 2.14 is violated:

1. The only primitive part that can be used in the beginning is p_0, leading to the tree p_0 with appearance $\begin{array}{c}|\\ A\end{array}$. This is no valid proof-tree, since p_0 has arity 1 and consequently there has to be a subtree rooted in A.
2. Now, in order to build that subtree, applying one of the other two construction-rules for proof-trees, we have to use a primitive part with head-variable A. The primitive parts in the formula-tree with head-variable A are p_1, p_3, p_4 and p_6. But, actually the only primitive part that can be used without violating the second condition in Definition 2.14 is p_1. Being a descendent of p_0 at branch 1, this primitive part, as well as p_2, has been made available by the use of p_0. The resulting proof-tree is $\begin{array}{c}p_0\\ p_1\end{array}$ with appearance

$$\begin{array}{c}|\\ A\\ |\\ A\end{array},$$

that is still not complete.

3. Again a subtree rooted in type-variable A has to be constructed, but now there are two more primitive parts that may be used. In fact, p_3 and p_4 (the descendents of p_1 at branch 1) have been made available by the use of p_1, and so one may now choose between p_1, p_3 or p_4 (p_2 is available, but its head-variable does not fit). Using p_1 again would lead to the still incomplete proof-tree

p_0
p_1
p_1

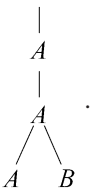
with appearance

$$\begin{array}{l} | \\ A \\ | \\ A \\ | \\ A \end{array}.$$

Again a subtree rooted in A has to be constructed and the use of p_1 does not allow us to use any primitive part different from those available in the step before. The same will happen if we choose to use p_3 instead of p_1. In fact, p_3 has no descendents and consequently there will be no further primitive parts available, but the same as in the step before. Furthermore, p_3 has also tail-variable A and consequently we will still be attempting to construct a subtree rooted in A. Finally, we can use part p_4 which leads us to the incomplete proof-tree

p_0
p_1
p_4

with appearance



Now, we have to construct two subtrees. One rooted in type-variable A and another rooted in type-variable B. Note, that p_4 has no descendent at branch 1 and the descendent p_6 at branch 2. Although, p_6 is now available for the construction of the subtree rooted in B, there is no new primitive part available for the construction of the subtree rooted in A. Consequently, again one of our tasks is the same as before and we conclude that there is no valid proof-tree for this formula.

Based on the argument used in the previous example, one can define a search algorithm for a valid proof-tree as follows. Begin the construction of a valid proof-tree

with p_0, the unique primitive part of form (P1) in the formula-tree. Proceed, building a complete subtree rooted at the tail-variable of p_0. During the construction of subtrees only use primitive parts that are allowed by the second condition in Definition 2.14. Call these the available parts at a certain stage. During the construction, explore all possibilities, using all available primitive part with the matching head-variable. The search stops in one of two ways. With success, if a valid proof tree is obtained. With failure, whenever trying to build a complete subtree rooted in the same type-variable and with the same set of available primitive parts as in one of the steps before; this means that there is no need to construct proof-trees with depth greater than the number of different type-variables (in the type) times the number of primitive parts (in the formula-tree). Eventually, one of the two halting conditions will be obtained, since the number of atoms and primitive parts found in φ is finite.

In order to establish the relation between the long normal inhabitants of a type and its valid proof-trees, we now describe an algorithm $\mathtt{PT}(\_)$ that given a long inhabitant M of a type φ computes a valid proof-tree for $\mathtt{PT}(M)$ for φ. The definition of this algorithm parallels the construction of a proof-tree from a long normal inhabitant in Example 2.12. In the following algorithm, we use the notion of erasing abstractions in a term, which consists of removing all prefixes λx in the term, obtaining as a result a purely applicative term.

Proof-tree Algorithm 2.17.

Input: a type φ and M, a long normal inhabitant of φ.

Output: a valid proof-tree $\mathtt{PT}(M)$ for φ.

Let M' be the term obtained from M by erasing abstractions in M and replacing each variable with the name of the corresponding primitive part in $\mathtt{tree}(\varphi)$. Then, $\mathtt{PT}(M)$ is the graphical representation of M' after inserting a top level node p_0 (the root-node of $\mathtt{tree}(\varphi)$).

We now examine an "inverse" algorithm which computes, given a type φ and any valid proof-tree $\mathtt{PT}$ for φ, a finite non-empty set $\mathtt{Terms}(\mathtt{PT})$ of long closed normal inhabitants of φ. The algorithm consists of three parts.

In the first two parts, the common structure of the lambda-terms that correspond to a certain valid proof-tree is recovered by a λ-term, that we call a *term-scheme*. These term-schemes resemble lambda-terms (possibly with bounded-variable clashes) in the simple type-system with the total discharge convention, or Prawitz-style natural deduction system [10], but actually they differ from them in the following way. While all free (distinct) variables in a subterm of a term in Prawitz' system have distinct types, distinct (free or bounded) variables in a subterm of a term-scheme may have the same type as long as they correspond to different primitive parts (or equivalently to different negative occurrences of the type as a subtype).

In the last part, we compute the set of all terms from which a given term-scheme may be obtained, by identifying in these terms all variables, that correspond to the same primitive part in the formula-tree. For example, for $\lambda x\,y.x(\lambda z.x(\lambda z.yzz))$ the term-scheme of type $((B\to A)\to A)\to(B\to B\to A)\to A$, the two bounded z's are given

distinct names and all possibilities explored for the subterms in their scope, leading to the following set of terms:

$$\{\lambda xy.x(\lambda z.x(\lambda z'.yzz)), \lambda xy.x(\lambda z.x(\lambda z'.yzz')), \lambda xy.x(\lambda z.x(\lambda z'.yz'z)),$$
$$\lambda xy.x(\lambda z.x(\lambda z'.yz'z'))\}.$$

Terms Algorithm 2.18.

Input: a type φ and a valid proof-tree PT for φ.

Output: a finite non-empty set Terms(PT) of long closed normal inhabitants of φ.

Let $p_0,\ldots,p_n$ be the primitive parts in tree(φ), where p_0 is the root-part and $n\geqslant 0$. The set Terms(PT) is constructed in the following way.

(a) Represent PT as an application using for $0\leqslant i\leqslant n$ the variable name x_i instead of p_i. Then, for each variable-occurrence x_i in this application, such that the primitive part p_i has arity $k_i\geqslant 0$, insert a (possibly empty) abstraction sequence before each of its k_i arguments. Here, the variable names in an abstraction sequence $\lambda x_{j_1}\ldots x_{j_{l_j}}$ to be inserted before the jth argument, $1\leqslant j\leqslant k_i$, correspond to the descendents $p_{j_1}\ldots p_{j_{l_j}}$ at branch j of p_i in tree(φ).

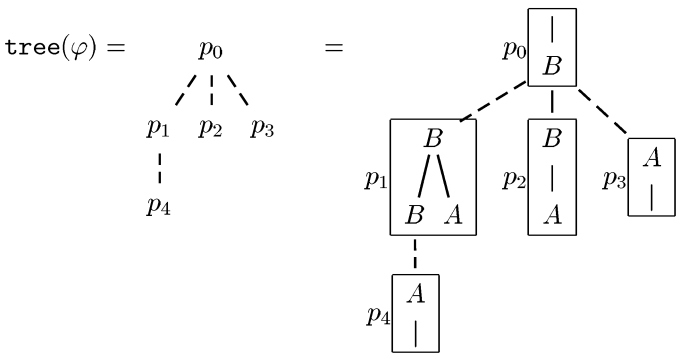
(b) Now, erase the variable x_0 at the top.

(c) Finally, for term-scheme T obtained in the previous step compute Terms(PT) $= T^{0\ldots0}_{x_1\ldots x_n}$ defined below, where $x^0=x$ for any variable x.

- $x_i{}^{k_1\ldots0\ldots k_n}_{x_1\ldots x_i\ldots x_n}=\{\bar{x}_i\}$;[3]
- $x_{i\,x_1\ldots x_i\ldots x_n}^{k_1\ldots k_i\ldots k_n}=\{x_i,x_i',x_i'',\ldots,x_i^{k_i-1}\}$, $k_i\geqslant 1$;
- $(ST)^{k_1\ldots k_n}_{x_1\ldots x_n}=\{S_iT_j \mid S_i\in S^{k_1\ldots k_n}_{x_1\ldots x_n}, T_j\in T^{k_1\ldots k_n}_{x_1\ldots x_n}\}$;
- $(\lambda x_i.T)^{k_1\ldots k_n}_{x_1\ldots x_n}=\{\lambda x_i^{k_i}.T_i \mid T_i\in T^{k_1\ldots(k_i+1)\ldots k_n}_{x_1\ldots x_i\ldots x_n}\}$.

The following example illustrates the application of the algorithm.

Example 2.19. Consider again $\varphi=((A\to B)\to A\to B)\to(A\to B)\to A\to B$ from Example 2.12 with formula-tree



[3] This case corresponds to the renaming of free occurrences of variables in T.

Let PT be the last valid proof-tree for φ in Example 2.15, i.e.

$$\mathtt{PT} = \begin{array}{c} p_0 \\ p_1 \\ p_1 \;\; p_3 \\ p_2 \;\; p_4 \\ p_4 \end{array} .$$

In the first step of the Terms algorithm, PT is represented by the application $x_0(x_1(x_1(x_2\, x_4)x_4)x_3)$. Then, since p_0 has arity 1 and descendents p_1, p_2 and p_3 at branch 1, the abstraction sequence $\lambda x_1x_2x_3$ is inserted before the first argument of x_0, leading to $x_0(\lambda x_1x_2x_3.x_1(x_1(x_2x_4)x_4)x_3)$. As p_1 has arity 2, descendent p_4 at branch 1 and no descendent at branch 2, the abstraction sequence λx_4 has to be inserted before the first argument of (every occurrence of) x_1 and no (or the empty) abstraction sequence will be inserted before the second argument of (all occurrences of) x_1, leading to $x_0(\lambda x_1x_2x_3.x_1(\lambda x_4.x_1(\lambda x_4.x_2x_4)x_4)x_3)$. Neither p_2, p_3 or p_4 have descendents; thus no further abstraction sequence has to be inserted. After erasing x_0 at the top we obtain the term-scheme $T = \lambda x_1x_2x_3.x_1(\lambda x_4.x_1(\lambda x_4.x_2x_4)x_4)x_3$. Finally, $\mathtt{Terms(PT)} = T^{0000}_{x_1x_2x_3x_4} = \{\lambda x_1x_2x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2x_4)x_4)x_3, \lambda x_1x_2x_3.x_1(\lambda x_4.x_1(\lambda x'_4.x_2x'_4)x_4)x_3\}$.

The following result proves the correctness of Algorithms 2.17 and 2.18 and will be used in the proof of Theorem 4.10. Note that the set of long normal inhabitants produced by Algorithm 2.18 may contain terms with bounded-variable clashes, but is of course equivalent (modulo α-conversion) to a set of long normal inhabitants without bounded-variable clashes.

Proposition 2.20. (i) *If M is a long normal inhabitant of a type φ, then* $\mathtt{PT}(M)$ *is a valid proof-tree for φ.*

(ii) *If* PT *is a valid proof-tree for a type φ, then every member of* Terms(PT) *is a closed long normal inhabitant of φ.*

Furthermore, the two algorithms are complementary in the sense that for every closed long normal inhabitant M of φ there is $M \in \mathtt{Terms}(\mathtt{PT}(M))$.

Proof. We begin the proof of (i) showing that $\mathtt{PT}(M)$ can in fact be obtained applying the rules for constructing proof-trees in Definition 2.13, and showing that both conditions in Definition 2.14 are satisfied by $\mathtt{PT}(M)$. For that, consider any occurrence $\bar{x}$ (not in an abstraction-sequence) of a variable x in M. Suppose that x has type $\alpha = (\alpha_{11} \to \cdots \to \alpha_{1m_1} \to A_1) \to \cdots \to (\alpha_{k1} \to \cdots \to \alpha_{km_k} \to A_k) \to A$, let p be the primitive part in $\mathtt{tree}(\varphi)$ that corresponds to x, i.e. p is the primitive part in the top of $\mathtt{t}(\alpha)$, and finally let $\bar{p}$ be the occurrence of p in $\mathtt{PT}(M)$ created by the replacement of occurrence $\bar{x}$ of x by p during the application of algorithm PT( ).

It follows from the Formula-tree Algorithm and from the structure of α, that p has arity $k \geqslant 0$ with head-variable A and tail-variables $A_1, \ldots, A_k$. Furthermore, for $1 \leqslant j \leqslant k$, p has exactly m_j descendents at branch j. Thus, by Lemma 2.11, the occurrence $\bar{x}$ of x in M occurs exactly with k arguments, which for $1 \leqslant j \leqslant k$ are respectively of the form

$\lambda y_{j1} \ldots y_{jm_j}.N_j$ and such that N_j is of atomic type A_j. Due to their atomic type, each N_j has to be of the form $z_j P_{j1} \ldots P_{js_j}$, $s_j \geqslant 0$. Thus, $\bar{p}$ will have exactly k arguments, $\bar{p}_1, \ldots, \bar{p}_k$, in $\mathtt{PT}(M)$ which are respectively the primitive parts corresponding to z_1 in N_1, ... and to z_k in N_k. This, satisfies the first condition in Definition 2.14. Furthermore, since $N_1, \ldots, N_k$ have types $A_1, \ldots, A_k$, respectively, we infer that the primitive parts corresponding to $z_1, \ldots, z_k$, i.e. $\bar{p}_1, \ldots, \bar{p}_k$, have head-variables $A_1, \ldots, A_k$, respectively, so, by the formation rules in Definition 2.13, they fit as arguments for $\bar{p}$.

Finally, we have to show that also the second condition in Definition 2.14 is satisfied by $\mathtt{PT}(M)$. Consider $\bar{p}$; we have to show that if p is a descendent at branch i of some primitive part p' in $\mathtt{tree}(\varphi)$, then there is some occurrence of $\bar{p}'$ in $\mathtt{PT}(M)$, such that $\bar{p}$ occurs in the subtree of $\mathtt{PT}(M)$ rooted in the ith branch of $\bar{p}'$.

Suppose $p = p_0$, the primitive part in the root of $\mathtt{tree}(\varphi)$, then there is no such p' and the second condition in Definition 2.14 is trivially true.

Suppose $p \neq p_0$ and let x be the variable corresponding to p. Since M is closed, $\bar{x}$ occurs in the scope of some abstraction sequence $\vec{x}$ containing x. If $\vec{x}$ is the initial abstraction sequence in M, then p descends from $p' = p_0$ at branch 1 in $\mathtt{tree}(\varphi)$ and the second condition in Definition 2.14 is satisfied, since p_0 is the primitive part in the root of $\mathtt{PT}(M)$. Otherwise, $\vec{x}$ is the initial abstraction sequence of the ith argument of a subterm of M of the form $yN_1 \ldots N_k$ and $i \in \{1, \ldots, k\}$, $k \geqslant 1$. Then, p descends in $\mathtt{tree}(\varphi)$ from the ith branch/tail-variable of the primitive part p' that corresponds to y. Since, $\bar{x}$ occurs in the ith argument of an occurrence of y, we conclude that $\bar{p}$ occurs in the subtree rooted at the ith branch of an occurrence of p' in $\mathtt{PT}(M)$.

For (ii) we begin by noting that the only occurrence of p_0 in $\mathtt{PT}$ is in its root and that p_0 has arity 1. Therefore the expression constructed in part (a) of Algorithm 2.18 is of the form $x_0(T)$ and that $T = \lambda x_{i_1} \ldots x_{i_k}.N$, where $p_{i_1}, \ldots, p_{i_k}$ are the descendents of p_0 (at branch 1) in $\mathtt{tree}(\varphi)$ and such that N corresponds to the subtree beneath p_0 in $\mathtt{PT}$.

The term $x_0(T)$ was constructed starting from an application of variables, by introduction of abstraction-sequences before subterms in argument position, but never in function position. So it is surely a λ-term in β-normal form and consequently the same is true for the term-scheme T, which is the result after step (b).

Now, consider any occurrence $\bar{x}_i$ (not in an abstraction-sequence) of a variable x_i in T, which is due to an occurrence $\bar{p}_i$ in $\mathtt{PT}$. Let p_m be the primitive part of arity $l \geqslant 1$ such that, in $\mathtt{tree}(\varphi)$, p_i is a descendent of p_m at some branch j. The second condition in the definition of valid proof-trees guarantees that there is at least one occurrence $\bar{p}_m$ in $\mathtt{PT}$ such that the $\bar{p}_i$ is in the subtree rooted in the jth branch of p_m. This is reflected in T as an occurrence of the variable x_m with l arguments and such that the occurrence $\bar{x}_i$ is in the jth argument of x_m, which begins with an abstraction sequence in which x_i occurs, binding thus the occurrence $\bar{x}_i$ (if not already binded[4]). We conclude that T is closed.

Finally, denote by φ_i, for $1 \leqslant i \leqslant n$, the subtype of φ such that the subtree $\mathtt{t}(\varphi_i)$ in $\mathtt{tree}(\varphi)$ has primitive part p_i at its top (cf. Algorithm 2.9). Now, consider any

[4] The same variable name may appear in more than one abstraction, corresponding to different occurrences of the same primitive part.

occurrence $\bar{p}_s$ of a primitive part p_s of arity $l \geqslant 1$ in PT, let PT′ be the subtree of PT rooted in branch $j \in \{1,\ldots,l\}$ of $\bar{p}_s$, let A be the jth tail-variable of p_s (and consequently the head-variable of the primitive part in the top of PT′) and finally let $p_{j_1},\ldots,p_{j_t}$, $t \geqslant 0$, be the descendents of p_s at branch j in $\mathtt{tree}(\varphi)$. It follows from the algorithm Terms that the jth argument of the occurrence $\bar{x}_s$ in $x_0(T)$, created due to $\bar{p}_s$, is of the form $T' = \lambda x_{j_1} \ldots x_{j_t}.P$. We will show by induction on the number of occurrences of type-variables in PT′ that, for the type-context $\Gamma = \{x_1 : \varphi_1,\ldots,x_n : \varphi_n\}$ there is $\Gamma \vdash \lambda x_{j_1} \ldots x_{j_t}.P : \varphi_{j_1} \to \cdots \to \varphi_{j_t} \to A$ in $\mathbf{TA}_\lambda$, i.e. given Γ the type $\varphi_{j_1} \to \cdots \to \varphi_{j_t} \to A$ can be assigned to $\lambda x_{j_1} \ldots x_{j_t}.P$, cf. [8, Chapter 2]. Then, taking for $\bar{p}_s$ the unique occurrence of p_0 in **PT**, one has $\Gamma \vdash \lambda x_{i_1} \ldots x_{i_k}.N : \varphi_{i_1} \to \cdots \to \varphi_{i_k} \to A_0$, where A_0 is the tail-variable of p_0, i.e. $\Gamma \vdash T : \varphi$. Furthermore, since T is closed, we conclude that also $\vdash T : \varphi$ is true in $\mathbf{TA}_\lambda$. On the other hand, T is clearly long, since every variable x_i of type $\varphi^1 \to \cdots \to \varphi^l \to A$ appears always with exactly l arguments, due to the fact that the corresponding primitive part p_i has l tail-variables.

We begin the induction proof with the case $\mathbf{PT}' = p_i$, $i \in \{1,\ldots,n\}$ and

$$p_i = \overset{A}{|},$$

hence $\varphi_i = A$. Since p_i has no tail-variable, there are no primitive parts descending from p_i in $\mathtt{tree}(\varphi)$, thus $\lambda x_{j_1} \ldots x_{j_t}.P = \lambda x_{j_1} \ldots x_{j_t}.x_i$ and $\Gamma \vdash \lambda x_{j_1} \ldots x_{j_t}.x_i : \varphi_{j_1} \to \cdots \to \varphi_{j_t} \to A$ is true in $\mathbf{TA}_\lambda$.

Now, let the primitive part in the top of $\mathbf{PT}'$ be p_i of arity $l \geqslant 1$. Furthermore, for $1 \leqslant s \leqslant l$ let $p_{s_1},\ldots,p_{s_{l_s}}$ be the descendents (in $\mathtt{tree}(\varphi)$) from p_i at branch s/tail-variable A_s, let $\lambda\vec{x}_s$ be the abstraction-sequence $\lambda x_{s_1} \ldots x_{s_{l_s}}$ and let $\varphi^s = \varphi_{s_1} \to \cdots \to \varphi_{s_{l_s}} \to A_s$. Then, $\varphi_i = \varphi^1 \to \cdots \to \varphi^l \to A$ and $P = x_i(\lambda\vec{x}_1.P_1)\ldots(\lambda\vec{x}_l.P_l)$, where each $\lambda\vec{x}_s.P_s$ is constructed from subtree $\mathbf{PT}_s$ rooted in the sth tail-variable of p_i. In order to show that $\Gamma \vdash \lambda x_{j_1} \ldots x_{j_t}.P : \varphi_{j_1} \to \cdots \to \varphi_{j_t} \to A$ is true in $\mathbf{TA}_\lambda$, it suffices to show that for $1 \leqslant s \leqslant l$, given context Γ the type φ^s can be assigned to $\lambda\vec{x}_s.P_s$ in $\mathbf{TA}_\lambda$. This follows from the induction hypothesis applied to $\mathbf{PT}_s$, thus finishing the induction.

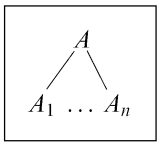
To finish the proof of part (ii) first note that for $n \geqslant 1$ and T resulting from (b) of Terms Algorithm, every term $M \in T^{0\ldots0}_{x_1\ldots x_n}$ is a closed term (for this it suffices to show by structural induction on N, that for any term N with variables in $\{x_1,\ldots,x_n\}$, if $M \in N^{k_1\ldots k_n}_{x_1\ldots x_n}$, then for the set $FV(M)$ of free variables of M, one has $FV(M) \subseteq \{x_i^j \mid 1 \leqslant i \leqslant n, 0 \leqslant j \leqslant k_i - 1\} \cup \{\bar{x}_i \mid 1 \leqslant i \leqslant n, x_i \in FV(N), k_i = 0\}$, and consequently for $M \in T^{0\ldots0}_{x_1\ldots x_n}$ there is $FV(M) = \emptyset$). On the other hand, we know that there is a deduction of $T : \varphi$ in $\mathbf{TA}_\lambda$. This deduction can then easily be changed into a deduction of $M : \varphi$, assigning identical types to variables with the same index.

The remaining claim that for every closed long normal inhabitant M of φ there is $M \in \mathtt{Terms}(\mathtt{PT}(M))$, included here for the sake of completeness, is a consequence of the symmetry of the two algorithms, and will actually not be used in this paper. □

2.3. Arity-1 formulas

In this subsection, we identify the class of formulas for which decidability of BB′IW-inhabitation will be shown.

Definition 2.21. A primitive part p in a formula-tree is called *terminating* iff it is of the form $\begin{array}{|}\hline A\\ \hline\end{array}$. It is called *composed* of arity $n \geqslant 1$ iff it is of the form



.

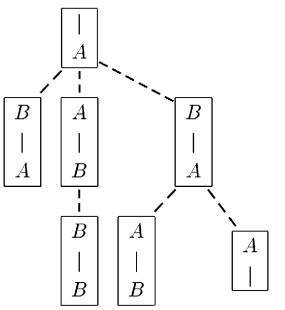
We call a formula an *arity*-1 *formula* iff every composed primitive part in its formula-tree is of arity 1.

Another simple characterisation of arity-1 formulas, that does not rely on the formula-tree approach can be given using the notion of polarities of occurrences of subformulas, cf. Definition 2.6. Then, it follows that arity-1 formulas are exactly the formulas such that all negative occurrences of subformulas are of the form $\alpha \to A$ or A, where α is a type and A a type-variable.

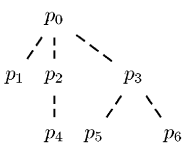
Example 2.22. The arity-1 formula

$$\tau = (A \to B) \to (((B \to B) \to B) \to A) \to (((B \to A) \to A \to A) \to B) \to A$$

has the following formula-tree



i.e. the tree



with primitive parts $p_0 = \begin{array}{|}\hline A\\ \hline\end{array}$, $p_1 = \begin{array}{|}\hline B\\ \hline A\\ \hline\end{array}$, $p_2 = \begin{array}{|}\hline A\\ \hline B\\ \hline\end{array}$, $p_3 = \begin{array}{|}\hline B\\ \hline A\\ \hline\end{array}$, $p_4 = \begin{array}{|}\hline B\\ \hline B\\ \hline\end{array}$,

$p_5 = \begin{array}{|}\hline A\\ \hline B\\ \hline\end{array}$, and $p_6 = \begin{array}{|}\hline A\\ \hline\end{array}$. A valid proof-tree for this formula is

p_0 with appearance $\begin{array}{c}|\\A\\|\\B\\|\\A\\|\end{array}$.
p_2
p_3
p_6

It is important to note that a proof-tree for an arity-1 formula is not really a tree, but it degenerates into a sequence/chain of parts $p_0,\ldots,p_f$. This is due to the fact that each internal primitive part in an arity-1 formula must always be of the form

$$\boxed{\begin{array}{c}X\\|\\Y\end{array}},$$

as can be seen in the two previous examples. Furthermore, note that whenever a primitive part p descends from another primitive part p', then p' must have arity 1 and p descends from p' at branch 1. So, in the remaining, where we deal with arity-1 formulas, we will usually just say that p descends from p' and no longer specify the branch which is necessarily branch 1.

The following properties or arity-1 formulas will be used later on. Here, we call a term *compound* if there occurs at least one application in it.

Lemma 2.23. *Let M be a long normal* BB′IW*-inhabitant of an arity-*1 *formula φ and such that M contains a compound subterm N. Then, N is of the form $\lambda x_1\ldots x_n.yP$, with $n\geqslant 0$. Moreover, $x_1,\ldots,x_{n-1}$ are all of non-atomic type. Furthermore, there are abstractions in P if and only if x_n is of non-atomic type.*

Proof. First note that neither M nor any subterm of M can be of the form $\lambda x.x$. The former is due to the fact that M has to contain at least one application in order to have a compound subterm N. The latter follows from the fact that if M has a subterm of the form $\lambda x.x$ then there must be another subterm of M of the form $zU_1\ldots(\lambda x.x)\ldots U_l$, $l\geqslant 0$, covered by abstraction sequences $\vec{s_1},\ldots,\vec{s_t}$, with $t\geqslant 1$, in which z occurs. This subterm then violates the condition imposed on BB′IW-inhabitants that there should be $Ind_{\vec{s_1},\ldots,\vec{s_t}}(z)\preccurlyeq Ind_{\vec{s_1},\ldots,\vec{s_t}}(\lambda x.x)$ which is not the case since $z\not\preccurlyeq\perp$. Furthermore, note that every variable x in M is of some type τ, where τ is a negative occurrence of a subformula of φ, also called a negative subpremise in [8]. Since φ is an arity-1 formula, it follows that τ is either of the form $\alpha\to A$ or A, where α is a type and A a type-variable. Thus, it follows from M being a long inhabitant that every variable occurs always with one argument if its type is of the form $\alpha\to A$ and always with zero arguments if its type is of the form A. We conclude that every compound subterm of M is of the form $\lambda x_1\ldots x_n.yP$, with $n\geqslant 0$. Since P cannot be of the form $\lambda x.x$ we

gain enough insight to conclude that M is necessarily of the form

$$\lambda\vec{s}_1.z_1^1(z_1^2(\ldots z_1^{k_1}(\lambda\vec{s}_2.z_2^1(\ldots z_2^{k_2}(\lambda\vec{s}_3.\ldots(\lambda\vec{s}_m.z_m^1(\ldots(z_m^{k_m}z)\ldots))\ldots)\ldots))\ldots))$$

with non-empty abstraction sequences $\vec{s}_1,\ldots,\vec{s}_m$, $m\geqslant 1$, $k_1,\ldots,k_m\geqslant 1$, such that every variable z_i^j is of non-atomic type and occurs in one of $\vec{s}_1,\ldots,\vec{s}_i$, with $1\leqslant i\leqslant m$ and $1\leqslant j\leqslant k_j$, while the only variable of atomic type is z. Finally, it follows from Definition 2.2 that z has to be the last variable in $\vec{s}_m$, which concludes the proof. □

Lemma 2.24. *Every* BB′IW*-inhabitated arity-*1 *formula has exactly one terminating primitive part in its formula-tree.*

Proof. This result follows from the analysis made in the proof of the previous lemma together with the fact that negative occurrences of atomic subformulas in a formula correspond to terminating primitive parts in its formula-tree. □

Due to Lemma 2.24, from now on we only consider arity-1 formulas with exactly one terminating primitive part in their formula-trees.

3. Rewriting-systems over typed sequences

In this section, we define the notion of rewriting-systems over typed sequences that will be used in order to prove decidability for arity-1 formulas. In fact, decidability will be shown using a mapping from arity-1 formulas to rewriting-systems. The main goal of these rewriting-systems is to describe the important steps during the search of a long normal BB′IW-inhabitant of an arity-1 formula. As such, remember that each variable x of type α in a long normal inhabitant of a formula φ corresponds to a negative occurrence of α in φ, and is represented by the subtree $\mathtt{t}(\alpha)$ of the formula-tree $\mathtt{tree}(\varphi)$. The primitive part, say p_x, in the top of this subtree $\mathtt{t}(\alpha)$ describes the consequences of using x during the construction of a subterm M: if

$$p_x = \boxed{\begin{array}{c} A \\ | \\ B \end{array}}$$

and has descendents $p^1,\ldots,p^n$ (at branch 1) in $\mathtt{tree}(\varphi)$, then M is of the form $x(\lambda x^1\ldots x^n.N)$ of atomic type A, where N of atomic type B has to be constructed. Furthermore, all the new variables $x^1,\ldots,x^n$ have to be used in N. Also all the variables in the abstraction sequence over M that have not been used before have to appear in N, as well as the last variable in this sequence (even if it was already used, as will be seen in Lemma 4.5). All this will be represented as a rewriting system. Here sequences of variables juxtaposed will be used to represent the whole abstraction sequence of variables over the subterm that is actually being constructed, i.e. all the variables, in order from the left, that have the subterm in their scope. A type-variable assigned to this sequence represents the type of this subterm, and rewriting-rules will describe the change of the current task due to the use of variables.

Definition 3.1. A *typed sequence* is a finite sequence of variables juxtaposed $x_1 \dots x_n$, with $n \geqslant 0$, to which a type-variable A has been assigned, and will be represented by $x_1 \dots x_n :: A$.

The meaning of a typed sequence $x_1 \dots x_n :: A$ (where $x_1 \dots x_n$ represents a sequence, in which variables may occur more than once, and not as in λ-calculus an application of variables) is the following. Given a type φ there will be a variable name associated with each primitive part in $\mathtt{tree}(\varphi)$, or equivalently to each negative subformula of φ. Then, during the construction of a long normal $\mathsf{BB'IW}$-inhabitant of φ the typed sequence $x_1 \dots x_n :: A$ represents an intermediate step such that so far the sequence of variables that have been introduced in abstractions have types corresponding to x_1, to $x_2, \dots$, to x_n. Also, in order to obtain a long normal $\mathsf{BB'IW}$-inhabitant a subterm of type A still has to be constructed. Note that although variables may occur more than once in a typed sequence, this does not imply a clash of bounded-variables. In fact, it rather describes the construction of a term in which (at least) two distinct variables have the same corresponding primitive part.

Definition 3.2. A *rewriting-system* over typed sequences is a tuple $\mathscr{R} = (\theta, R, x_f :: C_f)$ such that

- x_f is a variable and C_f a type-variable;
- θ is a typed sequence;
- R is a finite set of rules of the form $x :: A \mapsto x_1 \dots x_n :: B$, where $x, x_1, \dots, x_n$ are term-variables, A and B are type-variables and $n \geqslant 0$.

Definition 3.3. The *expansion rule* $\Rightarrow_{\mathscr{R}}$ of typed sequences with a rewriting-system $\mathscr{R} = (\theta, R, x_f :: C_f)$ is defined as follows:

- If $z_1 \dots z_m :: A$ is a typed sequence and $z_i :: A \mapsto x_1 \dots x_n :: B \in R$, for some $1 \leqslant i \leqslant m$, then

$$z_1 \dots z_m :: A \Rightarrow_{\mathscr{R}} z_1 \dots z_m x_1 \dots x_n :: B.$$

  The corresponding occurrence of z_i in $z_1 \dots z_m$ is said to be *used* in this expansion-step. Furthermore, if $i \neq m$, we call it to be *used in a non-final position*.

A typed sequence σ expands to $\tilde{\sigma}$ iff $\sigma \Rightarrow_{\mathscr{R}}^{\star} \tilde{\sigma}$, where $\Rightarrow_{\mathscr{R}}^{\star}$ denotes the reflexive, transitive closure of $\Rightarrow_{\mathscr{R}}$. A typed sequence σ *expands successfully* with $\mathscr{R} = (\theta, R, x_f :: C_f)$ iff there is some expansion from σ to a typed sequence $\tilde{\sigma} = z_1 \dots z_M :: C_f$ such that

- $z_M = x_f$ and $z_i \neq x_f$ for any other $z_i \in \{z_1, \dots, z_{M-1}\}$;
- every one of the occurrences of variables in $z_1, \dots, z_{M-1}$ has been used in a non-final position at least once during the expansion of σ to $\tilde{\sigma}$.

If the initial sequence θ expands successfully with $\mathscr{R} = (\theta, R, x_f :: C_f)$, then $\mathscr{R}$ is also called a *terminating rewriting-system*.

In the following, we define a function $\mathscr{R}(\_)$ that maps formulas into rewriting-systems. In Section 4, it will be shown that $\mathsf{BB'IW}$-theorems are associated with terminating rewriting-systems, while formulas that are not $\mathsf{BB'IW}$-theorems are

associated with non-terminating rewriting-systems. A decision algorithm for termination of rewriting-systems will be given in Section 5.

Definition 3.4. Given an arity-1 formula τ, consider its formula-tree $FT(\tau)$ with primitive parts $p_0,\ldots,p_n,p_f$, where p_0 is the root of $FT(\tau)$ and

$p_f =$ 

its unique terminating primitive part. We associate with τ a rewriting-system $\mathscr{R}(\tau) = (\theta, R, x_f :: C_f)$ by the following:

- θ is the typed sequence $x_1 \ldots x_m :: A$, where $p_1,\ldots,p_m$ are the primitive parts descending from the root

  $p_0 =$ 

  in $\mathtt{tree}(\tau)$;
- for every primitive part p_i of the form

  and with descendents $p_{i_1},\ldots,p_{i_{m_i}}$ in $\mathtt{tree}(\tau)$, where $m_i \geqslant 0$, there is a rule $x_i :: B \mapsto x_{i_1} \ldots x_{i_{m_i}} :: C$ in R (occasionally abbreviated by $x_i :: B \mapsto \vec{x}_i :: C$).

Example 3.5. For τ from Example 2.22, we have $\mathscr{R}(\tau) = (\theta, R, x_6 :: A)$, where

- $\theta = x_1x_2x_3 :: A$
- and

$$
\begin{array}{lll}
R: & x_1 :: B \mapsto & \varepsilon :: A \\
& x_2 :: A \mapsto & x_4 :: B \\
& x_3 :: B \mapsto & x_5x_6 :: A \\
& x_4 :: B \mapsto & \varepsilon :: B \\
& x_5 :: A \mapsto & \varepsilon :: B.
\end{array}
$$

The following expansion sequence shows that $\mathscr{R}(\tau)$ is terminating, since θ expands successfully with $\mathscr{R}(\tau)$ to $\sigma = x_1x_2x_3x_4x_4x_5x_6 :: A$.

$$
\begin{array}{rl}
\underline{x_1x_2}x_3 :: A & \Rightarrow \underline{x_1}x_2x_3x_4 :: B \\
& \Rightarrow x_1\underline{x_2}x_3x_4 :: A \\
& \Rightarrow x_1x_2x_3\underline{x_4}x_4 :: B \\
& \Rightarrow x_1x_2\underline{x_3}x_4x_4 :: B \\
& \Rightarrow x_1x_2x_3x_4x_4\underline{x_5}x_6 :: A \\
& \Rightarrow x_1x_2x_3x_4\underline{x_4}x_5x_6 :: B \\
& \Rightarrow \underline{x_1}x_2x_3x_4x_4x_5x_6 :: B \\
& \Rightarrow x_1x_2x_3x_4x_4x_5x_6 :: A.
\end{array}
$$

Note, that for sake of readability in every step of this expansion we underlined the occurrence of the variable used to obtain the next sequence. Furthermore, we omitted reference to $\mathscr{R}(\tau)$ during this expansion, writing $\Rightarrow$ rather than $\Rightarrow_{\mathscr{R}(\tau)}$. This will be done whenever there is no ambiguity concerning the rewriting-system we refer to.

4. BB′IW-inhabitation and terminating rewriting-systems

4.1. BB′IW-*inhabitation implies terminating rewriting-system*

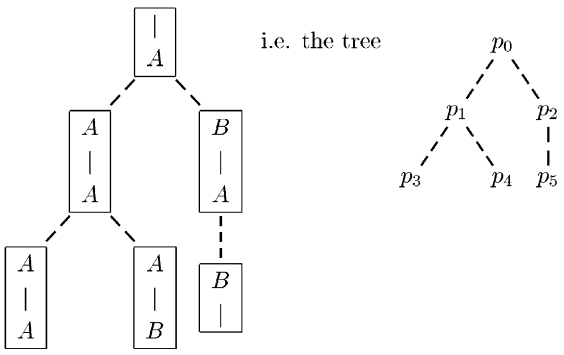
In this subsection, we show that every BB′IW-inhabitated arity-1 formula τ gives rise to a terminating rewriting-system $\mathscr{R}(\tau)=(\theta,R,x_f :: C_f)$. For this, we prove that given a BB′IW-inhabitant M of τ, the initial sequence θ expands successfully to the typed sequence $Seq(M)$ defined below, in Definition 4.2.

First, consider the unique long inhabitant M^+ of τ to which M η-expands. Every variable in M^+ corresponds exactly to one primitive part, different from p_0, in the formula-tree $\mathtt{tree}(\tau)$ with primitive parts $p_0, p_1, \ldots, p_n, p_f$, where $n \geqslant 0$. Now, let N be any term obtained by renaming variables in M^+ in the following way: all occurrences of variables corresponding to a primitive part p_i, with $i \in \{1,\ldots,n,f\}$, are named x_i^j for some $j \geqslant 1$. Here, occurrences of the same variable are given the same exponent j (i.e. the same name) and different variables are given different exponents (i.e. receive different names). Obviously, there is $N \equiv_\alpha M^+$ and N is a long normal BB′IW-inhabitant of τ.

Example 4.1. Consider the arity-1 formula

$$\tau = (((A \to A) \to (B \to A) \to A) \to A) \to ((B \to A) \to B) \to A$$

with formula-tree



with primitive parts

$$p_0 = \boxed{\begin{array}{c}|\\ A\end{array}},\quad p_1 = \boxed{\begin{array}{c}A\\|\\ A\end{array}},\quad p_2 = \boxed{\begin{array}{c}B\\|\\ A\end{array}},\quad p_3 = \boxed{\begin{array}{c}A\\|\\ A\end{array}},\quad p_4 = \boxed{\begin{array}{c}A\\|\\ B\end{array}} \text{ and } p_5 = \boxed{\begin{array}{c}B\\|\end{array}}.$$

A (long) normal BB′IW-inhabitant of τ is

$$M = \lambda xy.x(\lambda zu.x(\lambda vs.u(y(\lambda t.z(v(st)))))),$$

where x corresponds to p_1, y to p_2, z and v to p_3, u and s to p_4, and finally t to p_5.

After renaming we obtain the term

$$N = \lambda x_1^1 x_2^1.x_1^1(\lambda x_3^1 x_4^1.x_1^1(\lambda x_3^2 x_4^2.x_4^1(x_2^1(\lambda x_5^1.x_3^1(x_3^2(x_4^2 x_5^1)))))).$$

Definition 4.2. Let M be a long inhabitant of an arity-1 formula τ and consider a term $N \equiv_\alpha M$ as above, i.e. a long normal inhabitant of τ where variables are indexed according to the primitive part in $\mathtt{tree}(\tau)$ they correspond to. Furthermore, let C_f be the type-variable in the (unique) terminating primitive part in $\mathtt{tree}(\tau)$. We define $Seq(M) = x_1 \dots x_n :: C_f$, where $x_1 \dots x_n$ is the sequence of variables in successive abstractions in N after erasing superscripts/exponents.

Example 4.3. For the term from Example 4.1 we have $Seq(M) = x_1x_2x_3x_4x_3x_4x_5 :: B$.

Below we will show that for every type τ with normal inhabitant M the rewriting-system $\mathscr{R}(\tau)$ is terminating, since its initial sequence θ expands successfully to $Seq(M)$. During the proof, we will also need the notion of effective occurrences of variables in terms.

Definition 4.4. We call the underlined occurrence of x in a term of the form $\lambda x.y_1(\dots y_k(\underline{x}P)\dots)$, with $k \geqslant 0$, *ineffective*. Every ineffective occurrence of a variable x in a subterm of a term M is an ineffective occurrence of x in M. An occurrence of a variable in a term (but not in an abstraction sequence) that is not ineffective is called *effective*.

In other words, an occurrence of a variable x in a long normal BB′IW-term is always effective, unless it is the first x in a subterm of the form xP covered by an abstraction sequence whose *last* variable is x. The following result shows that all variables in a long normal BB′IW-term have at least one effective occurrence.

Lemma 4.5. *If $Q = \lambda x.y_1(\dots y_k(xP)\dots)$ is a subterm of a long normal* BB′IW*-term M, then there is at least one effective occurrence of x in P.*

Proof. Consider the sequence of the variables $\vec{x} = x_1, \dots, x_n, x$, $n \geqslant 0$, that occur in abstractions in M which have the subterm $y_1(\dots y_k(xP)\dots)$ in their scope, together with the induced implicit order $\bot \prec x_1 \prec \cdots \prec x_n \prec x$. From Definitions 2.1 and 2.2, we conclude that $Ind_{\vec{x}}(P) = x$, thus $x \in FV(P)$. We now prove the result by induction on the

length of P. If $P=x$, this occurrence of x is an effective occurrence. If $P=yP'$, then $x\in FV(P')$ (for $y\neq x$ as well as for $y=x$) and by the induction hypothesis x has an effective occurrence in P'. Otherwise, $P=\lambda y.P'$, where $x\in FV(P')$. All occurrences of x in P' are clearly effective. □

Theorem 4.6. *For every* BB′IW*-inhabitant M of an arity-*1 *formula τ, the initial sequence θ expands successfully with $\mathscr{R}(\tau)=(\theta,R,x_f::C_f)$ to $Seq(M)$.*

Proof. Consider $N\equiv_\alpha M$ as before, i.e. obtained from the unique long normal inhabitant of τ to which M η-expands by renaming variables according to the primitive parts they correspond to.

First note that for $Seq(M)=s_1\dots s_l::A$ there is exactly one occurrence of x_f in $s_1\dots s_l$, which is s_l and that $A=C_f$. The latter results from the definition of $Seq(M)$ and the former from x_f being the only variable of atomic type and from Lemma 2.23.

Now let P be any long subterm of N that is not an abstraction and which is in the scope of variables $\vec{x}_P=x_1,\dots,x_n$ and let B be the atomic type of P, for P is long. Let $\vec{y}_P=y_1,\dots,y_k$ be the subsequence of $x_1,\dots,x_n$ obtained by deleting all variables x_j such that P occurs as a subterm of the argument of an effective occurrence of x_j. Note that all of $y_1,\dots,y_k$ occur at least once (cf. Lemma 4.5) in P. For example, if $P=y(zu)$ is a subterm of $\lambda x\,y\,z.z(y(x(\lambda u.y(zu)))$ there is $\vec{x}=x,y,z,u$ and $\vec{y}_P=z,u$; if we consider $P'=x(\lambda u.y(zu))$, then $\vec{x}=x,y,z$ and $\vec{y}_{P'}=x,z$.

We now prove by induction on the number of applications in P, that the typed sequence $x_1\dots x_n::B$ expands to $Seq(M)$, thus $Seq(M)=x_1\dots x_n\sigma::C_f$, and during this expansion all occurrences of variables in $y_1,\dots,y_k$ and σ, different from x_f, are used in a non-final position.

First suppose that $P=x$ is a variable. Since it has atomic type B, we conclude from Lemma 2.23 that $x=x_n=x_f$, thus $B=C_f$, as well as $Seq(M)=x_1\dots x_n::C_f$. On the other hand, we know from Lemma 4.5 that all of $y_1,\dots,y_k$ have to occur at least once in P. Thus, $y_1,\dots,y_k$ is just x_f. The result becomes trivially true. In fact, $x_1\dots x_n::B$ expands in zero steps to $Seq(M)$, and there are no variables different from x_f in $y_1,\dots,y_k$.

Otherwise, by Lemma 2.23, $P=x(\lambda u_1\dots u_t.Q)$ such that Q, of type D, is not an abstraction and $t\geqslant 0$. Note that the first occurrence of x in P is effective if and only if $x\neq x_n$, i.e. the corresponding occurrence of x is in a non-final position in $x_1\dots x_n$. From the correspondence between variables and primitive parts we conclude that there is the rule $x::B\mapsto u_1\dots u_t::D$ in R. Thus, $x_1\dots x_n::B\Rightarrow x_1\dots x_nu_1\dots u_t::D$. Now, it is sufficient to apply the induction hypothesis to the term Q, typed sequence $\vec{x}_Q=x_1\dots x_nu_1\dots u_t::D$ and $\vec{y}_Q$ defined as before, which means that

$$\vec{y}_Q=\begin{cases} y_1\dots y_ku_1\dots u_t & \text{if } x\notin\{y_1,\dots,y_k\}\\ & \text{or if the first } x \text{ in } P \text{ is ineffective}\\ & \text{and } x=x_n=y_k,\\ y_1\dots y_{l-1}y_{l+1}\dots y_ku_1\dots u_t & \text{otherwise } (x=y_l,\ 1\leqslant l\leqslant k),\end{cases}$$

which finishes the proof. □

Example 4.7. For τ from Example 4.1, we have $\mathscr{R}(\tau) = (\theta, R, x_5 :: B)$ with $\theta = x_1x_2 :: A$ and

$$R = \{x_1 :: A \mapsto x_3x_4 :: A,\ x_2 :: B \mapsto x_5 :: A,\ x_3 :: A \mapsto \varepsilon :: A,\ x_4 :: A \mapsto \varepsilon :: B\}.$$

Recall that $N = \lambda x_1^1 x_2^1.x_1^1(\lambda x_3^1 x_4^1.x_1^1(\lambda x_3^2 x_4^2.x_4^1(x_2^1(\lambda x_5^1.x_3^1(x_3^2(x_4^2 x_5^1))))))$.

In fact, the initial sequence θ expands to $Seq(M) = x_1x_2x_3x_4x_3x_4x_5 :: B$ as shown below:

$\underline{x_1}x_2 :: A \Rightarrow$	$x_1 :: A \mapsto x_3x_4 :: A$	$N = \lambda x_1x_2.x_1P_1$
$\Rightarrow \underline{x_1}x_2x_3x_4 :: A$	$x_1 :: A \mapsto x_3x_4 :: A$	$P_1 = \lambda x_3x_4.x_1P_2$
$\Rightarrow x_1x_2x_3\underline{x_4}x_3x_4 :: A$	$x_4 :: A \mapsto \varepsilon :: B$	$P_2 = \lambda x_3x_4.x_4P_3$
$\Rightarrow x_1\underline{x_2}x_3x_4x_3x_4 :: B$	$x_2 :: B \mapsto x_5 :: A$	$P_3 = x_2P_4$
$\Rightarrow x_1x_2\underline{x_3}x_4x_3x_4x_5 :: A$	$x_3 :: A \mapsto \varepsilon :: A$	$P_4 = \lambda x_5.x_3P_5$
$\Rightarrow x_1x_2x_3x_4\underline{x_3}x_4x_5 :: A$	$x_3 :: A \mapsto \varepsilon :: A$	$P_5 = x_3P_6$
$\Rightarrow x_1x_2x_3x_4x_3\underline{x_4}x_5 :: A$	$x_4 :: A \mapsto \varepsilon :: B$	$P_6 = x_4P_7$
$\Rightarrow x_1x_2x_3x_4x_3x_4x_5 :: B$	$x_f = x_5$	$P_7 = x_5$

where the second column contains the rewriting rule expansion applied and the third column contains its corresponding subterm, as in the proof of Theorem 4.6.

Corollary 4.8. *If τ is a* BB′IW*-inhabited type, then $\mathscr{R}(\tau)$ is a terminating rewriting-system.*

4.2. Termination implies BB′IW-inhabitation

In the following, we show that whenever the initial sequence θ of $\mathscr{R}(\tau)$ expands successfully to a sequence σ, then τ has at least one BB′IW-inhabitant. For this, consider any successful expansion from θ to $\sigma = z_1 \dots z_n x_f :: C_f$ and perform the following annotations on $z_1 \dots z_n x_f$. First, associate new exponents to different occurrences of the same variable in this sequence. Second, insert in this sequence an item $[x_i^j]$ before every subsequence that has been introduced using the occurrence of x_i^j. Third, append the item $[x_f^1]$ at the end of the sequence.

The algorithm T, that we will show to compute a BB′IW-inhabitant $M_\sigma = \mathtt{T}(\tilde{\sigma})$ of τ from this new sequence, denoted by $\tilde{\sigma}$, is given by the following:

- $\mathtt{T}([x]) = x$;
- $\mathtt{T}([x]\rho) = x(\mathtt{T}(\rho))$;
- $\mathtt{T}(x\rho) = \lambda x.\mathtt{T}(\rho)$.

Example 4.9. An annotated version $\tilde{\sigma}$ of $\sigma = x_1x_2x_3x_4x_4x_5x_6 :: A$ from Example 3.5 is

$$x_1^1x_2^1x_3^1[x_2^1]x_4^1[x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]$$

and $M_\sigma = \mathtt{T}(\tilde{\sigma})$ is given by

$$\begin{aligned} M_\sigma &= \mathtt{T}(x_1^1x_2^1x_3^1[x_2^1]x_4^1[x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]) \\ &= \lambda x_1^1.\mathtt{T}(x_2^1x_3^1[x_2^1]x_4^1[x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]) \end{aligned}$$

$$
\begin{aligned}
&= \lambda x_1^1 x_2^1.\mathtt{T}(x_3^1[x_2^1]x_4^1[x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1])\\
&= \lambda x_1^1 x_2^1 x_3^1.\mathtt{T}([x_2^1]x_4^1[x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1])\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\mathtt{T}(x_4^1[x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.\mathtt{T}([x_1^1][x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(\mathtt{T}([x_2^1]x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1])))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\mathtt{T}(x_4^2[x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.\mathtt{T}([x_4^1][x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(\mathtt{T}([x_3^1]x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1])))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\mathtt{T}(x_5^1x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]))))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\lambda x_5^1.\mathtt{T}(x_6^1[x_5^1][x_4^2][x_1^1][x_6^1]))))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\lambda x_5^1x_6^1.\mathtt{T}([x_5^1][x_4^2][x_1^1][x_6^1]))))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\lambda x_5^1x_6^1.x_5^1(\mathtt{T}([x_4^2][x_1^1][x_6^1])))))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\lambda x_5^1x_6^1.x_5^1(x_4^2(\mathtt{T}([x_1^1][x_6^1])))))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\lambda x_5^1x_6^1.x_5^1(x_4^2(x_1^1(\mathtt{T}([x_6^1])))))))))\\
&= \lambda x_1^1 x_2^1 x_3^1.x_2^1(\lambda x_4^1.x_1^1(x_2^1(\lambda x_4^2.x_4^1(x_3^1(\lambda x_5^1x_6^1.x_5^1(x_4^2(x_1^1x_6^1)))))))).
\end{aligned}
$$

Theorem 4.10. *Consider an arity-*1 *formula* τ *with rewriting-system* $\mathscr{R}(\tau)=(\theta,R,x_f::C_f)$ *and a typed sequence* σ *such that* θ *expands successfully to* σ. *Then*, M_σ is a $\mathsf{BB'IW}$-*inhabitant of* τ.

Proof. Consider an annotated version $\tilde{\sigma}$ of σ. We begin arguing that for every such sequence $\tilde{\sigma}$, if $\tilde{\sigma}=\rho_1 x[x_1]\ldots[x_n]\rho_2$, $n\geqslant 0$, then $x\in FV(\mathtt{T}(\rho_2))$. For this note that items of the form $[z]$ in $\tilde{\sigma}$ are introduced due to the use of z during the expansion of θ to σ. Since every variable in σ, but x_f, has to be used at least once in a non-final position during the expansion (cf. Definition 3.3), there has to be at least one item $[x]$ in $[x_1]\ldots[x_n]\rho_2$. Now, even if $x=x_i$ for some $1\leqslant i\leqslant n$, then this expansion with x was performed in a final position and there must be another item $[x]$ in ρ_2. Thus, in any case there has to be at least one item $[x]$ in ρ_2 leading to a free (variables have been renamed in $\tilde{\sigma}$, thus there is no other unbracketed x in $\tilde{\sigma}$) occurrence of x in $\mathtt{T}(\rho_2)$.

Now, in order to prove that M_σ is a $\mathsf{BB'IW}$-term, i.e. $M_\sigma\in HRM_\varepsilon$ (cf. Proposition 2.3), we show that during the calculation of M_σ, for every computed subterm $M=\mathtt{T}(\rho)$ in the scope of variables $\vec{z}=z_1,\ldots,z_s$, $s\geqslant 1$, $M\in HRM_{\vec{z}}$ as well as $Ind_{\vec{z}}(M)=z_s$. We prove this result by induction on the length of ρ, which is the remaining subsequence of $\tilde{\sigma}$, which during the computation of $M_\sigma=\mathtt{T}(\tilde{\sigma})$ is consumed from left to right.

For $\rho=[x]$, there is $x=x_f^1=z_s$ and the result is true.

For $\rho=[x]\rho'$, one has $M=\mathtt{T}([x]\rho')=x(M')$, where $M'=\mathtt{T}(\rho')$ with $M'\in HRM_{\vec{z}}$ and $Ind_{\vec{z}}(M')=z_s$. On the other hand, the item $[x]$ appears in $\tilde{\sigma}$ due to the use of x during

the expansion. So, in $\tilde{\sigma}$ there must be an unbracketed occurrence of x on the left of this item $[x]$. Since every unbracketed occurrence of a variable gives origin to an abstraction, we conclude that x is one of $z_1,\ldots,z_s$, and consequently $x \leqslant z_s$. Thus, it follows from $Ind_{\vec{z}'}(x) = x \prec z_s = Ind_{\vec{z}'}(M')$ that $M \in HRM_{\vec{z}'}$ as well as $Ind_{\vec{z}'}(M) = z_s$.

For $\rho = x\rho'$, there is $M = \mathtt{T}(x\rho') = \lambda x.M'$, where $M' = \mathtt{T}(\rho')$ with $M' \in HRM_{\vec{z}x}$ and $Ind_{\vec{z}x}(M') = x$, hence $x \in FV(M')$. We conclude that $M = \lambda x.M' \in HRM_{\vec{z}}$. On the other hand, since M is in the scope of variables $z_1,\ldots,z_s$, there must be unbracketed occurrences of variables $z_1,\ldots,z_s$ (in this order, possibly separated by bracketed occurrences of variables) in $\tilde{\sigma}$ on the left of $x\rho'$. Thus, $\tilde{\sigma}$ is of the form $\rho^{\star} z_s[x_1]\ldots[x_n]x\rho'$, $n \geqslant 0$, for some sequence $\rho^{\star}$ (containing unbracketed occurrences of $z_1,\ldots,z_{s-1}$). We conclude from the remark in the beginning of the proof that $z_s \in FV(M)$, hence $Ind_{\vec{z}}(M) = z_s$.

Correct typing is guaranteed by the fact that the term M computed by $\mathtt{T}$ is (modulo α-conversion) in the set $\mathtt{Terms}(\mathtt{PT}_\sigma)$ which has been shown (cf. Proposition 2.20) to be a set of closed long normal inhabitants of τ. Here $\mathtt{PT}_\sigma$ denotes the proof-tree

$$\begin{array}{l} p_0 \\ p_{i_1} \\ \vdots \\ p_{i_n} \\ p_f \end{array},$$

where $[x_{i_1}^{j_1}]\ldots[x_{i_n}^{j_n}][x_f^1]$ is the subsequence of bracketed items in $\tilde{\sigma}$.

Note that if $[x_{i_1}^{j_1}]\ldots[x_{i_n}^{j_n}][x_f^1]$ is the subsequence of bracketed items in $\tilde{\sigma}$, then $\tilde{\sigma}$ has to be of the form $x_{k_1}^{l_1}\ldots x_{k_s}^{l_s}[x_{i_1}^{j_1}]\vec{x}_{i_1}^{j_1}[x_{i_2}^{j_2}]\vec{x}_{i_2}^{j_2}\ldots[x_{i_n}^{j_n}][x_f^1]$, where $\theta = x_{k_1}\ldots x_{k_s}$, and where each subsequence $\vec{x}_i^j$ is the sequence of variables introduced by the use of variable x_i^j in the expansion of σ and marked by the item $[x_i^j]$ in $\tilde{\sigma}$. The sequence following $[x_{i_n}^{j_n}]$ has to empty (but for the type $a \to a$). Otherwise the variables in this sequence would still have to be used leading to further bracketed items in $\tilde{\sigma}$. Also note that if a variable with index s introduces a sequence of variables respectively with indexes $s_1,\ldots,s_m$, then in the formula-tree of the formula the descendents of the primitive part labelled with s are, respectively, labelled with $s_1,\ldots,s_m$. Thus, $M = \mathtt{T}(\tilde{\sigma}) = \lambda x_{k_1}^{l_1}\ldots x_{k_s}^{l_s}.x_{i_1}^{j_1}(\lambda\vec{x}_{i_1}^{j_1}.x_{i_2}^{j_2}(\lambda\vec{x}_{i_2}^{j_2}.\ldots.(x_{i_n}^{j_n}x_f^1)\ldots))$.

On the other hand, when applying the algorithm $\mathtt{Terms}$ to $\mathtt{PT}_\sigma$, then in the first step the application $x_0(x_{i_1}(x_{i_2}\ldots(x_{i_n}x_f)\ldots))$ is formed. Then, before the argument of each of $x_0, x_{i_1},\ldots,x_{i_n}$ an abstraction sequence $\lambda\vec{x}_0, \lambda\vec{x}_{i_1},\ldots,\lambda\vec{x}_{i_n}$ is introduced and x_0 is deleted from the top, leading to the term-scheme $\lambda\vec{x}_0.x_{i_1}(\lambda\vec{x}_{i_1}.x_{i_2}(\lambda\vec{x}_{i_2}.\ldots.(x_{i_n}(\lambda\vec{x}_{i_n}.x_f))\ldots))$. Also here the abstraction sequence that is introduced before the argument of an variable with index s is the sequence of variables indexed with the labels of the primitive parts that in the formula-tree descend from the primitive part labelled with s. It follows from the remark in the footnote, which guarantees that no primitive parts descend in the formula-tree from the primitive part labelled with i_n, that $\vec{x}_{i_n}$ is in fact an empty sequence and that $\lambda\vec{x}_0 = \lambda x_{k_1}^{l_1}\ldots x_{k_s}^{l_s}$. Thus, the term-scheme can in fact be obtained from M erasing the superscripts of variables and on the other hand M will be one of the terms obtained from this term-scheme in the third step of the algorithm $\mathtt{Terms}$. □

5. Decidability for arity-1 formulas

In order to prove decidability for arity-1 formulas, it remains to show decidability of termination of rewriting-systems. We begin with a result on sequences of n-tuples of non-negative integers.

5.1. The finiteness of non-ascending $\mathbb{N}^n$-sequences

Let $\sigma = \bar{\mathsf{a}}_1, \ldots, \bar{\mathsf{a}}_i, \ldots$ be a sequence of n-tuples on non-negative integers, $\bar{\mathsf{a}}_i = \langle a_i^1, \ldots, a_i^n \rangle \in \mathbb{N}^n$, where $a_i^j \in \mathbb{N}$ represents the jth position of tuple $\bar{\mathsf{a}}_i$.

We define a partial order on n-tuples as follows:

$$\bar{\mathsf{a}} \leqslant \bar{\mathsf{b}} \quad \text{iff } a^j \leqslant b^j \quad \text{for } 1 \leqslant j \leqslant n.$$

The sequence σ is *non-ascending* iff for any $i < k$ we have $\bar{\mathsf{a}}_i \not\leqslant \bar{\mathsf{a}}_k$. Note that in a non-ascending sequence it follows that $\bar{\mathsf{a}}_i \neq \bar{\mathsf{a}}_k$ for $i \neq k$.

We want to prove that any non-ascending sequence of n-tuples in $\mathbb{N}^n$ is finite. This result is, in fact, equivalent to Kripke's Lemma, cf. [1], and to some other known results such as the infinite division principle by Meyer, cf. [9], or to Dickson's Lemma in number theory, cf. [6], that can be obtained from Hilbert's Finite Basis Theorem, cf. [7]. These powerful results have been used in various areas among which in decidability proofs for some fragments of relevant systems, cf. [11].

Theorem 5.1. *Let $\sigma = \bar{\mathsf{a}}_1, \ldots, \bar{\mathsf{a}}_i, \ldots$ be a non-ascending sequence of tuples in $\mathbb{N}^n$. Then σ is finite.*

Proof. Suppose, for the sake of contradiction, that σ is infinite. There cannot be any bounding value on the elements of the sequence, otherwise all elements of the sequence σ would be inside an n-dimensional cube, but there are only finitely many distinct elements inside an n-dimensional cube in $\mathbb{N}^n$.

So at least one of the dimensions is unbounded. Without loss of generality, suppose the first dimension is unbounded (otherwise we reorder the dimensions). Let us form a new sequence $\sigma^{(1)}$, a subsequence of σ,

$$\sigma^{(1)} = \bar{\mathsf{a}}_1^{(1)}, \ldots, \bar{\mathsf{a}}_i^{(1)}, \ldots$$

by deleting from σ all the elements in which the unbound first dimension is not strictly greater than the largest value so far of the first dimension. As a result, the infinite sequence that consists of the first dimension of $\sigma^{(1)}$ is strictly ascending.

If there is another unbounded dimension in $\sigma^{(1)}$, which without loss of generality we can suppose is the second dimension, we repeat this process above of extracting a sequence $\sigma^{(2)}$, where the second dimension is strictly ascending. Note that in this process the first dimension remains strictly ascending.

We repeat this process as long as there is an unbounded dimension. Suppose we have made this process m times, $1 \leqslant m \leqslant n$, and we end up with an infinite subsequence

$\sigma^{(m)}$ of σ.

$$\sigma^{(m)} = \bar{a}_1^{(m)}, \ldots, \bar{a}_i^{(m)}, \ldots.$$

At this point, we have m dimensions of strictly ascending values and $n-m$ bounded dimensions in $\sigma^{(m)}$. Take a bounded dimension, say dimension $m+1$. Since $\sigma^{(m)}$ is infinite, there is a value c_{m+1} in dimension $m+1$ that repeats infinitely. Now form a new subsequence $\sigma^{(m+1)}$ of $\sigma^{(m)}$ by deleting all values for which dimension $m+1$ is different from c_{m+1}, so that dimension $m+1$ is constant in $\sigma^{(m+1)}$.

Repeat this process for all bounded dimensions. We end up with an infinite sequence $\sigma^{(n)}$, in which m dimensions are strictly ascending and $m-n$ dimensions are constant. Since $\sigma^{(n)}$ is a subsequence of σ, this contradicts the fact that σ is non-ascending.

Therefore, the initial sequence σ cannot be infinite, which finishes the proof. □

The following lemma, similar to König's lemma, will be used in the proof of Theorem 5.4, which leads to our main result.

Lemma 5.2. *Let $\mathbb{L}$ be a set and $\leqslant$ be a binary relation defined on $\mathbb{L}$. Let $\mathcal{T}$ be any finitely branching tree whose nodes are labelled by elements of $\mathbb{L}$. Consider the following recursive procedure, where n is a node of $\mathcal{T}$ and $S \subseteq \mathbb{L}$.*

```
proc Visit(n,S):
    let l be the label of node n
    if there exists l' ∈ S s.t. l' ⩽ l
        then return
    else
        for each descendent n' of n
                        do Visit(n', S ∪ {l})
```

If for any node n in $\mathcal{T}$ the call `Visit(n,∅)` *does not terminate, then there exists an infinite sequence $(l_i)_i$ in $\mathbb{L}$ such that for all $0 \leqslant i < j$, one has $l_i \not\leqslant l_j$.*

Proof. The existence of a sequence $(l_i)_i$ is easily established as follows: assume that `Visit(`n_0`,∅)` does not terminate for some node n_0. Then, since the tree is finitely branching, there is at least one descendent n_1 of n_0 such that `Visit(`n_1`,{`l_0`})` also does not terminate, where l_0 is the label of n_0. But then there is also a direct descendent n_2 of n_1, such that `Visit(`n_2`,{`l_0, l_1`})` does not terminate, where l_1 is the label of n_1 and $l_0 \not\leqslant l_1$. Similarly, we can conclude that there is a descendent n_3 of n_2, such that `Visit(`n_3`,{`l_0, l_1, l_2`})` does not terminate, where l_2 is the label of n_2 and such that $l_0 \not\leqslant l_1$, $l_0 \not\leqslant l_2$ and $l_1 \not\leqslant l_2$. Since `Visit(`n`,∅)` does not terminate, it is possible to repeat this procedure indefinitely. □

5.2. Deciding termination of rewriting-systems

The following Lemma will be used in order to prove decidability of termination of rewriting-systems.

Lemma 5.3. *Consider a rewriting-system $\mathscr{R}$ and a typed sequence σ, such that σ expands successfully with $\mathscr{R}$. Then, there is at least one successful expansion sequence $\sigma = \sigma_1, \ldots, \sigma_n$ such that for all i, j with $1 \leqslant i < j \leqslant n$, the typed sequences σ_i and σ_j do not satisfy at least one of the following conditions*:

- *the type-variable in σ_i and σ_j is the same*;
- *the last term-variable in σ_i and σ_j is the same*;
- *in σ_i and σ_j the same term-variables occur*;
- *for every variable x such that k_x occurrences of x in σ_i have not been used in a non-final position in the expansion of σ to σ_i, there are at least k_x occurrences of x in σ_j that have not been used in a non-final position in the expansion of σ to σ_j.*

Proof. Consider a successful expansion sequence $\sigma = \sigma_1, \ldots, \sigma_i, \ldots, \sigma_j, \ldots, \sigma_{j+k}$ from σ to σ_{j+k} of length $j + k$, where $i, j, k > 0$, $j > i$ and such that

- the type-variable in σ_i and σ_j is the same;
- the last term-variable in σ_i and σ_j is the same;
- in σ_i and σ_j the same term-variables occur;
- for every variable x such that k_x occurrences of x in σ_i have not been used in a non-final position in the expansion of σ to σ_i, there are at least k_x occurrences of x in σ_j that have not been used in a non-final position in the expansion of σ to σ_j.

Then, another successful expansion sequence $\sigma = \sigma_1, \ldots, \bar{\sigma}_i = \sigma_i, \bar{\sigma}_{i+1}, \ldots, \bar{\sigma}_{i+k}$ of length $i + k < j + k$ can be obtained by the following. For $l = 0, \ldots, k - 1$, suppose that σ_{j+l+1} was obtained from σ_{j+l} using rule $x_l :: A_l \mapsto \vec{x}_l :: B_l$. Then expand $\bar{\sigma}_{i+l}$ with this same rule and using the leftmost occurrence of x_l in $\bar{\sigma}_{i+l}$ that has not been used in a non-final position in the expansion of σ to $\bar{\sigma}_{i+l}$, if any, or otherwise any other occurrence of x_l in $\bar{\sigma}_{i+l}$.

Now repeat the whole process to this new, shorter successful expansion sequence, as long as necessary. □

Theorem 5.4. *Termination of rewriting-systems over typed sequences is decidable.*

Proof. Let $\mathscr{R} = (\theta, R, x_f :: C_f)$ be a rewriting-system in which occur $n > 0$ term-variables $x_1, \ldots, x_n$ (after renaming if necessary) and $m > 0$ type-variables $A_1, \ldots, A_m$. Given an expansion sequence $\sigma_1, \sigma_2, \ldots$ we associate with each typed sequence σ_i in this expansion a tuple $L_i = (\bar{\mathsf{a}}_i, V_i, x^i, A^i)$, with $\bar{\mathsf{a}}_i = \langle a_i^1, \ldots, a_i^n \rangle \in \mathbb{N}^n$, $V_i \subseteq \{x_1, \ldots, x_n\}$, $x^i \in \{x_1, \ldots, x_n\}$ and $A^i \in \{A_1, \ldots, A_m\}$ by the following:

- For $j = 1, \ldots, n$ let a_i^j be the number of occurrences of x_j in σ_i that have not been used in a non-final position during the expansion of σ_1 to σ_i;
- let V_i be the set of variables occurring in σ_i;
- let x^i be the last variable in σ_i;
- and finally let A^i be the type-variable in σ_i.

We call a typed sequence σ_j in this expansion a repetition of another typed sequence σ_i with $i < j$ if and only if $\bar{\mathsf{a}}_i \leqslant \bar{\mathsf{a}}_j$, $V_i = V_j$, $x^i = x^j$ and $A^i = A^j$, in which case we write that $L_i \leqslant L_j$. We know from Lemma 5.3 that if $\mathscr{R}$ is terminating, then there is at least one repetition-free successful expansion sequence starting with the initial sequence θ. On the other hand, there are only a finite number of distinct V_k's, x^k's as well as A^k's.

Thus, it follows from Theorem 5.1 that there are no infinite, repetition-free expansion sequences, i.e. there is no infinite sequence of tuples $(L_i)_i$ such that for any $0 \leqslant i < j$ one has $L_i \not\leqslant L_j$.

Consider the finitely branching tree $\mathscr{T}$ defined by the following: the top-node is labelled by the tuple L_0 corresponding to the initial sequence $\sigma_0 = \theta$. Furthermore, the descendents of a node labelled with a tuple L_i are all (finitely many) nodes labelled with tuples $L_{i_1}, \ldots, L_{i_{n_i}}$, corresponding to all the sequences $\sigma_{i_1}, \ldots, \sigma_{i_{n_i}}$ that can be obtained in one expansion-step from σ_i.

Finally, change procedure `Visit` to immediately stop if a successful, repetition-free expansion sequence is reached, and apply it to $\mathscr{T}$. Lemma 5.2 and the remark above about the L_i's guarantee that this call terminates, thus making it a decision procedure. □

Now we are able to conclude our main result.

Theorem 5.5. $\mathsf{BB'IW}$-*inhabitation is decidable for arity*-1 *formulas.*

Proof. It follows from Theorems 4.6 and 4.10 that an arity-1 formula τ is $\mathsf{BB'IW}$-inhabited if and only if the corresponding rewriting-system $\mathscr{R}(\tau)$ is terminating. The decidability of the latter has been shown in Theorem 5.4. □

6. Conclusions

We have shown the decidability of the arity-1 fragment of $\mathsf{BB'IW}$-logic using the termination of a special kind of rewriting-system. The decision procedure is based on the finiteness of non-ascending $\mathbb{N}^n$-sequences. The complexity of the decision procedure is bound to be quite costly, but yet undetermined. In fact, the results in [13] suggest that it is primitive recursive in the Ackermann function.

The method proposed here gets considerably more complicated if applied to the whole fragment, for then the rewriting-system will not be expanding a sequence but a tree. The relationship between the structure of $\mathsf{BB'IW}$-λ-terms and these tree-expanding rewriting-systems still has to be investigated.

And finally, we have not ruled out the possibility of the whole $\mathsf{BB'IW}$-logic being undecidable. That undecidability, if proven, will have to depend on formulas that are not arity-1.

Acknowledgements

We would like to thank the anonymous referees for numerous comments which contributed to enhance the presentation.

The work presented in this paper has been partially supported by funds granted to *LIACC* through *Programa de Financiamento Plurianual*, *Fundação para a Ciência e Tecnologia* and *Programa POSI*.

References

[1] A.R. Anderson, N.D. Belnap, Entailment, Vol. I, Princeton University Press, USA, 1975.
[2] H. Barendregt, Lambda calculi with types, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Background: Computational Structures, Handbook of Logic in Computer Science, Vol. 2, Oxford Science Publications, 1992, pp. 117–309.
[3] C.-B. Ben-Yelles, Type-assignment in the lambda-calculus; syntax and semantics, Ph.D. Thesis, Mathematics Department, University of Wales, Swansea, UK, 1979.
[4] S. Broda, L. Damas, On the structure of normal λ-terms having a certain type, Proc. 7th WoLLIC'2000, 2000, pp. 33–43, copy in http://www.dcc.fc.up.pt/∼sbb/wollic00.ps.
[5] M.W. Bunder, Proof finding algorithms for implicational logics, Theoret. Comput. Sci. 232 (2000) 165–186.
[6] L.E. Dickson, Finiteness of the odd perfect primitive abundant numbers with n distinct prime factors, Amer. J. Math. 35 (1913) 413–422.
[7] D. Hilbert, Über die Theorie der algebraischen Formen, Math. Ann. 36 (1890) 473–534.
[8] J.R. Hindley, Basic Simple Type Theory, in: Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, 1997.
[9] R.K. Meyer, H. Ono, The Finite Model Property for BCK and BCIW, Studia Logica 53 (1) (1994) 107–118.
[10] D. Prawitz, Natural Deduction, Almqvist and Wiksell, Sweden, 1965.
[11] J. Riche, R.K. Meyer, Kripke, Belnap, Urquhart and relevant decidability & complexity, in: G. Gottlob, E. Grandjean, K. Seyr (Eds.), Computer Science Logic, CSL'98, Lecture Notes in Computer Science, Vol. 1584, Springer, Berlin, 1998, pp. 224–240.
[12] P. Trigg, J.R. Hindley, M.W. Bunder, Combinatory abstraction using **B**, **B′** and friends, Theoret. Comput. Sci. 135 (1994) 405–422.
[13] A. Urquhart, The complexity of decision procedures in relevance logic, in: J.M. Dunn, A. Gupta (Eds.), Truth or Consequences, Essays in Honor of Nuel Belnap, Kluwer Academic Publishers, Dordrecht, 1990, pp. 61–76.