Type-Inhabitation: Formula-Trees vs. Game Semantics[1]

Sandra Alves, Sabine Broda

e-mail: sandra@dcc.fc.up.pt, sbb@dcc.fc.up.pt
LIACC & CMUP & DCC-FC, University of Porto

Technical Report Series: DCC-2014-08
Version 1.0 December 2014

Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL

Tel: 220 402 900 Fax: 220 402 950
http://www.dcc.fc.up.pt/Pubs/

[1]Funded by ERDF (program COMPETE) and FCT project PEst-C/MAT/UI0144/2013

A Short Note on Type-inhabitation: Formula-Trees vs. Game Semantics

Sandra Alves LIACC
Sabine Broda CMUP
Department of Computer Science, University of Porto

January 19, 2015

Abstract

Type-inhabitation is a topic of major importance, due to its close relationship to provability in logical systems and has been studied from different perspectives over the years. In 2000 a new proof method has been presented, evidencing the close relationship between the structure of types and their inhabitants. More recently, in 2011, another method has been given in the context of game semantics. In this paper we clarify the similarities between the two approaches.

1 Introduction

In the simply typed λ-calculus, the problem of associating to a type a term that inhabits it, which is known as type inhabitation, has been a major focus of research over the years. Through the Curry-Howard isomorphism the problem is equivalent to provability of formulas in the implicational fragment of propositional logic [20], and has major implications in the area of proof-theory. Since normal forms in the λ-calculus correspond to Prawitz's [24] notion of normal deduction, algorithms for deciding type-inhabitation can be used for indirectly decide provability. Note that typed λ-calculi derived from the Curry-Howard isomorphism led to the development of theorem assistant tools, such as Coq, where proofs are formalized as programs, which can be checked and executed, and which are valuable tools in the area of formal verification.

The research carried out in the area led to a vast number of results, ranging from the definition of algorithms for generating/counting terms/proofs, to the capture of complexity classes, and the establishment of conditions guaranteeing the uniqueness of normal inhabitants of a given type. For a non-exhausting list of references we point to [4, 9, 14, 27, 26, 19, 22]. The subject has also been studied from the point of view of category-theory, where the uniqueness of type inhabitants for a giving typing was established through the verification of certain syntactic constraints [1, 23, 2]. This result is known as the coherence theorem and has several computational implications. In particular, in the area of computational linguistics, and since the appearance of grammars of lambda-terms [15], knowing that such terms are unique inhabitants of their principal typings has been fundamental in developing efficient parsers for these grammars [25], which in turn have been used in solving problems in text parsing and generation [6].

In [7] a new formal method for exploring type inhabitation was presented, which was later developed in [12], and that clearly evidenced the relationship between the structure of types and their normal inhabitants. This procedure, called *Formula-Tree Method*, represents types by splitting them into atomic parts, which are then used to construct a tree from which the inhabitants of the type can be obtained, and where the process of obtaining the tree is guided by the structure of the type. The method proved to be effective in establishing new results, as well as simplifying existing proofs [10, 8, 13]. Note that, the problem of type inhabitation has been proved to be undecidable for several typed calculi. Although decidable for some well known typed calculi, in general it is a very hard problem: it was proved to be PSPACE-complete for the simply typed λ-calculus [26, 28] and EXPTIME-hard for rank-2 intersection-types [29].

More recently, another method for studying type inhabitation was given, through the use of game semantics [5], that also explores the relation between the structure of types and terms. In that context, type inhabitants are seen as the interpretation of winning strategies in the arena given by a particular typing. From a categorical point of view, this relation has been established between dialogue games and innocent strategies [17]. Although the two method were developed within different backgrounds, there are several similarities between the concepts employed.

The main goal of this paper is to clarify the close relation between the formula-tree method and the method based on game semantics, by exploring the correspondence between:

- the formula-tree of a type and the arena associated to a type;
- proof-trees and winning strategies in a game.

This relation also establishes a direct correspondence between the term scheme associated to the proof-tree of a type and the interpretation of a winning strategy in the arena associated to a type. Furthermore, we refer to previous results that were obtained using both methods.

2 Preliminaries

2.1 The simply typed λ-calculus

In this paper we assume familiarity with basic results on the simply typed λ-calculus as described in [18] or [3]. We denote type-variables (atoms) by $a, b, c, \ldots$ and arbitrary types by lower-case Greek letters $\alpha, \beta, \gamma \ldots$. The sets of type-variables and of simple types are respectively denoted by $\mathcal{A}$ and $\mathcal{T}$. A typing environment Γ is a finite set of type assignments $x : \alpha$, such that whenever $(x : \alpha) \in \Gamma$ and $(x : \beta) \in \Gamma$, one has $\alpha = \beta$. A typing is a pair $< \Gamma, \gamma >$. We say that $< \Gamma, \gamma >$ is a typing of M, or that M is an inhabitant of $< \Gamma, \gamma >$, and write $\Gamma \vdash M : \gamma$, if this formula can be obtained from the inference rules below.

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha} \qquad \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \quad \Delta \vdash N : \alpha}{\Gamma \cup \Delta \vdash MN : \beta}$$

We write $\Gamma, x : \alpha$ to denote the set $\Gamma \cup \{x : \alpha\}$, such that x does not occur in another type assignment in Γ. As standard, we assume $\to$ to be right-associative. If $\vdash M : \gamma$ we also say that M is an inhabitant of γ. One has $\{x_1 : \gamma_1, \ldots, x_n : \gamma_n\} \vdash M : \gamma$ if and only if $\vdash \lambda x_1 \ldots x_n.M : \gamma_1 \to \ldots \to \gamma_n \to \gamma$. Note that, the two methods we consider in this paper, can both be formulated in terms of inhabitation of typings as well as in terms of inhabitation of types. We will focus here on the inhabitation of types, rather than typings. This is standard procedure and often preferable (cf. [28], and [3] pp. 182 and following) when addressing problems related to type checking, typability and inhabitation. A λ-term M is in *(β)-normal form* iff it is of the form $\lambda x_1 \ldots x_n.yQ_1 \ldots Q_m$, such that $m, n \geq 0$ and $Q_1, \ldots, Q_m$ are in normal form.

In the following we will recall some of the less standard results on simple types and their normal inhabitants. These are partly due to Ben-Yelles, cf. [4], and an exhaustive exposition can be found in [18]. A β-normal inhabitant M of a type γ is called a *long* normal inhabitant of γ 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 $(zP_1 \ldots P_n)$, $(n \geq 0)$, that is not in a function position, has atomic type. The finite set of all terms, obtained by η-reducing a λ-term M, is called the *η-family* of M and denoted by $\{M\}_\eta$. It has been shown, cf. [4] and [18], that the η-families of the long normal inhabitants of γ partition the set of normal inhabitants of γ into non-overlapping finite subsets, each η-family containing just one long member. Furthermore, Ben-Yelles, cf. [4] and [18], showed that every normal inhabitant of a typing γ can be η-expanded to one unique (up to α-conversion) long normal inhabitant of γ. A simple expansion-algorithm can be found in [18]. Thus, when studying normal inhabitants of a typing one might focus on the set of its long normal inhabitants from which all normal inhabitants can be obtained by η-reduction. Every type γ can be uniquely written as $\gamma = \gamma_1 \to \ldots \gamma_n \to a$, with $n \geq 0$.

Definition 1 Given a type γ, the *polarity* of occurrences of types in γ is defined as follows:

- γ is a positive occurrence in γ;
- if $\alpha \to \beta$ occurs positively (resp. negatively) in γ , then that occurrence of α is negative (resp. positive) and that occurrence of β is positive (resp. negative) in γ.

A term M has a *bound-variable clash* iff M contains an abstractor λx and a (free, bound or binding) occurrence of x that is not in its scope. On the other hand, every term can be α-converted to a term without bound-variable clashes. In this paper we consider λ-terms without bound-variable clashes.

2.2 Tree-domains and labelled trees

We will use $s, s_1, \ldots$ to range over finite sequences of positive natural numbers, ϵ to denote the empty sequence, $s_1 \cdot s_2$ to denote the concatenation of two sequences and $|s|$ to denote the length of a sequence. As usual the set of such sequences is denoted by $\mathbb{N}_+^\star$. Given $s, s' \in \mathbb{N}_+^\star$, we say that s *enables* s', and write $s \vdash s'$, if there is an $i \in \mathbb{N}_+$ such that $s' = s \cdot i$. Then, the reflexive, transitive closure of $\vdash$ allows us to define the notion of *initial segment*, i.e. $s \vdash^\star s'$ iff there exists s'' such that $s' = s \cdot s''$.

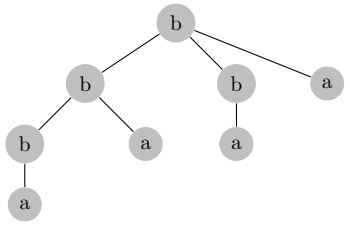
We recall the following formal definitions about labelled trees, cf. [16].

Definition 2 A *tree-domain* is a nonempty set $D_{\mathbf{t}}$ of sequences of natural numbers such that

- if $s \in D_{\mathbf{t}}$ and $s' \vdash^\star s$ then $s' \in D_{\mathbf{t}}$;
- if $s \cdot i \in D_{\mathbf{t}}$ then $s \cdot j \in D_{\mathbf{t}}$ for every j such that $1 \leq j < i$.

An $\mathcal{L}$-*labelled tree* $\mathbf{t}$ is a map from some tree domain $D_{\mathbf{t}}$ into a set $\mathcal{L}$. A tree $\mathbf{t}$ is *finite* iff $D_{\mathbf{t}}$ is finite.

Example 3 For $D_{\mathbf{t}} = \{\epsilon, 1, 2, 3, 1 \cdot 1, 1 \cdot 2, 1 \cdot 1 \cdot 1, 2 \cdot 1\}$ and $\mathcal{L} = \{a, b\}$ the tree $\mathbf{t}$ defined by $\mathbf{t}(\epsilon) = \mathbf{t}(1) = \mathbf{t}(1 \cdot 1) = \mathbf{t}(2) = b$ and $\mathbf{t}(1 \cdot 2) = \mathbf{t}(1 \cdot 1 \cdot 1) = \mathbf{t}(2 \cdot 1) = \mathbf{t}(3) = a$ can also be represented as follows.



We will call elements of $\mathcal{L}$ *labels* of the tree $\mathbf{t}$, and elements of $D_{\mathbf{t}}$ *nodes* of the tree. A node s is a *leaf* node if $s \cdot 1 \notin D_{\mathbf{t}}$, and an *internal* node otherwise. A node s' is a *descendant* of a node s iff $s \vdash^\star s'$ and $s \neq s'$. A node s' is a *direct descendant* of a node s iff $s \vdash s'$. A node s is a (direct) *ancestor* of a node s' iff s' is a (direct) descendant of s. The *depth* of a node s is the length $|s|$ of the sequence s. The *height* of a finite tree $\mathbf{t}$ is $max_{s \in D_{\mathbf{t}}} |s|$. From now on, whenever there are no numbers greater than 9 in the sequences of a tree-domain, we generally omit the concatenation symbol in the sequences. As such, we will represent the domain in the previous example as $D_{\mathbf{t}} = \{\epsilon, 1, 2, 3, 11, 12, 111, 21\}$.

Definition 4 If $\mathbf{t}$ is a tree and s is a node of $\mathbf{t}$, then the *subtree* $\mathbf{t}[s]$ of $\mathbf{t}$ at s is the tree with domain $D_{\mathbf{t}[s]} = \{s' \mid s \cdot s' \in D_{\mathbf{t}}\}$ defined by $\mathbf{t}[s](s') = \mathbf{t}(s \cdot s')$.

Definition 5 If $\mathbf{t_1}, ..., \mathbf{t_n}$ are $\mathcal{L}-$labelled trees and $l \in \mathcal{L}$ then let $l[\mathbf{t_1}, ..., \mathbf{t_n}]$ denote the tree with domain

$$\{\epsilon\} \cup \bigcup_{1 \leq i \leq n} \{i \cdot s \mid s \in D_{\mathbf{t_i}}\}$$

defined by $l[\mathbf{t_1}, ..., \mathbf{t_n}](\epsilon) = l$ and $l[\mathbf{t_1}, ..., \mathbf{t_n}](i \cdot s) = \mathbf{t_i}(s)$, for $1 \leq i \leq n$.

Note that any labelled tree can be written in a unique way as $l[\mathbf{t_1}, ..., \mathbf{t_n}]$ where $n \geq 0$. For sake of readability, when $n = 0$, we generally write l instead of $l[]$.

Definition 6 One can associate with any type γ a tree $\mathbf{t}_\gamma$, labelled by its type-variables, i.e. atomic types, as follows:

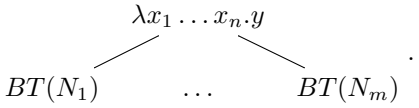
- if γ is an atom a, then $\mathbf{t}_\gamma$ is a;
- if γ is of the form $\gamma_1 \to \ldots \to \gamma_n \to a$, then its tree is $\mathbf{t}_\gamma = a[\mathbf{t}_{\gamma_1}, ..., \mathbf{t}_{\gamma_n}]$.

Example 7 The tree of type $\gamma = ((a \to b) \to a \to b) \to (a \to b) \to a \to b$ is the tree $\mathbf{t}$ from Example 3, and can be written as $b[b[b[a], a], b[a], a]$.

Consider a β-normal form M, of the form $\lambda x_1 \ldots x_n.yN_1 \ldots N_m$, where $n, m \geq 0$ and such that $N_1, \ldots, N_m$ are also terms in β-normal form. The variable y is called the *head* of M and is denoted by $\mathtt{head}(M)$. The Böhm tree[1] of M, denoted by $BT(M)$, is

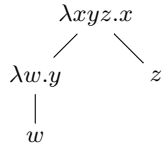
$$\lambda x_1 \ldots x_n.y$$

if $m = 0$ and otherwise



.

Example 8 For $D_{\mathbf{t}} = \{\epsilon, 1, 2, 11\}$ and $\mathcal{L} = \{\lambda xyz.x, \lambda w.y, w, z\}$ the tree $\mathbf{t}$ defined by $\mathbf{t}(\epsilon) = \lambda xyz.x$, $\mathbf{t}(1) = \lambda w.y$, $\mathbf{t}(2) = z$, and $\mathbf{t}(11) = w$ is the Böhm tree of the term $M = \lambda xyz.x(\lambda w.yw)z$ and can also be represented as follows.



It is well known that, given a (long) β-normal inhabitant M of a type γ, there is exactly one deduction for $\vdash M : \gamma$, and that in this unique deduction every variable and subterm is given a type. Furthermore, if M has no bound-variable clashes, then all occurrences of a variable x in this deduction correspond to exactly one occurrence of a subtype γ_x in γ. This particular occurrence of γ_x as well as the corresponding node s_x in the tree $\mathbf{t}_\gamma$ of γ, such that $\mathbf{t}_\gamma[s_x] = \mathbf{t}_{\gamma_x}$ can be easily determined. A simple algorithm for computing γ_x and s_x can be found in [12].

Example 9 The term M from Example 8 is a long normal inhabitant of γ from Example 7. For variables x and w one has, for instance, $\gamma_x = (a \to b) \to a \to b$ and $s_x = 1$, $\gamma_w = a$ and $s_w = 111$.

3 The Formula-Tree Method

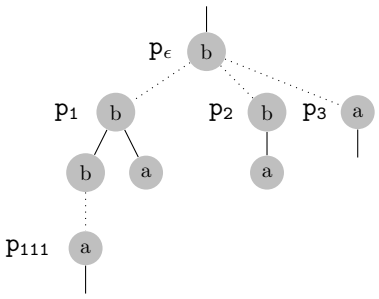
In this section we recall the Formula-tree proof method for type-inhabitation [12].

3.1 Formula-trees and Proof-trees

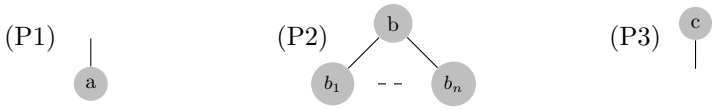
Given a type γ, its *formula-tree*, denoted by FT_γ, is obtained from $\mathbf{t}_\gamma$ by splitting $\mathbf{t}_\gamma$ into small parts p_s, that are called *primitive parts*, and which are formed by the nodes s of odd length and their direct descendants. Additionally, one considers a root-part consisting of the root-node of FT_γ. In order to distinguish this part from the parts resulting from nodes with no direct descendants, i.e. from leaf nodes, an edge is added in the top of the type-variable in the former, and an edge below the type-variables in the latter.

[1] Note, that we restrict the definition of Böhm trees to the case of β-normal forms.

Example 10 Consider again γ as in Examples 3 and 7. It's formula-tree FT_γ is the following.



We conclude that *primitive parts* 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. Now, it is easy to see that FT_γ forms a tree-like structure consisting of primitive parts. Here, the primitive part p_ϵ in the top is the only part of the form (P1). Any other primitive part p_s is of the form (P2) or (P3), and s is the position of its head-variable in $\mathbf{t}_\gamma$. Furthermore, p_s descends directly from (i.e. is enabled by) the ith tail-variable of another primitive part $p_{s'}$ of arity n, for some integer $1 \leq i \leq n$, such that $s = s' \cdot i \cdot j$ for some $j \geq 1$.

Definition 11 A *proof-tree* for γ is a tree PT, labelled with primitive parts of FT_γ, that satisfies the following conditions.

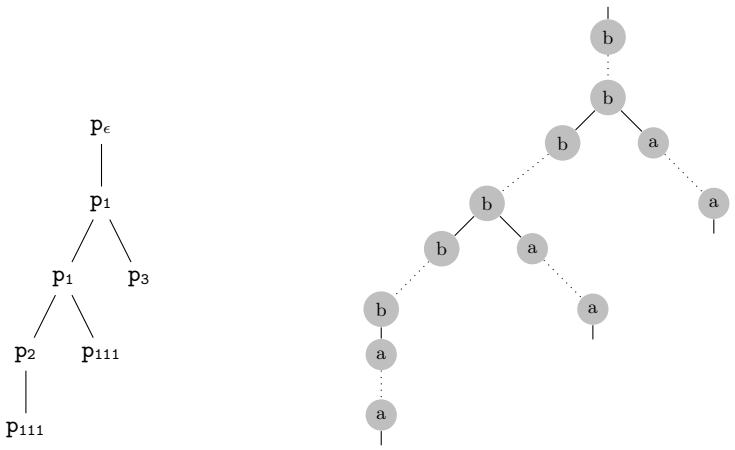
i. p_ϵ labels the root of PT.

ii. Every occurrence of a primitive part p_s in PT, with $n \geq 0$ tail-variables $a_1, \ldots, a_n$, has exactly n direct descendants $p_{s_1}, \ldots, p_{s_n}$ with head-variables $a_1, \ldots, a_n$ respectively.

iii. Finally, if p_s labels a node s' in PT and the primitive part p_s descends directly from (i.e. is enabled by) the ith tail-variable of another primitive part $p_{s''} \neq p_\epsilon$ in FT_γ, i.e. $s = s'' \cdot i \cdot j$, then there is an ancestor s''' of s' in PT labelled by $p_{s''}$ and the node s' is in the ith subtree rooted in s'''.

It follows from this definition, that constructing a proof-tree for a type γ is a kind of puzzle/game consisting of primitive parts that can be 'put together', while respecting the hierarchy imposed on them by the formula-tree of γ. The game always starts with primitive part p_ϵ. In the first step it is necessary to link the tail-variable a of p_ϵ to some 'available' primitive part p_s, whose head-variable is also a. If p_s has no tail-variable, the game is won. Otherwise, it is necessary to pursue until there are no tail-variables in any leaf node (which is a primitive part) of the proof-tree constructed so far. In [11] one can find a tool where the potentialities of the Formula-tree proof method can be explored.

Example 12 The following is a proof-tree for the type γ from previous examples and is constructed from its formula-tree FT_γ given in Example 10.



It has been shown in [12], that for every normal inhabitant M of a type γ there is one unique proof-tree of γ. Conversely, every proof-tree of γ represents a finite set of normal inhabitants, all of them with the same principal type (which is obviously not necessarily γ). The corresponding transformation algorithms are given in the next subsection.

3.2 From long normal inhabitants to proof-trees and back

Informally, the proof-tree $\mathbf{t}_M$ of a long normal inhabitant M, without bound-variable clashes, is a tree which, after removing its top node p_ϵ, has the same structure as the Böhm tree of M and such that every node, labelled by some expression $\lambda x_1 \dots x_n.y$ in the Böhm tree, is labelled by the node p_{s_y}.

Definition 13 Consider a closed long normal inhabitant M of a type γ without bound-variable clashes. Its *proof-tree* $\mathbf{t}_M$ is obtained from the Böhm tree $BT(M)$ by replacing every label of the form $\lambda x_1 \dots x_n.y$ by the sequence/position p_{s_y} and adding an extra node with label p_ϵ at the root of this tree.

Example 14 Consider the inhabitant $M = \lambda xyz.x(\lambda w.yw)z$ of γ from example 7. In order to compute $\mathbf{t}_M$ note that $s_x = 1$, $s_y = 2$, $s_z = 3$ and $s_w = 111$, and that the Böhm tree $BT(M)$ of M is the following, from which we obtain the proof-tree on the right:

We now describe how, given a proof-tree of a type γ, we compute the corresponding finite set of long normal inhabitants of γ.

Definition 15 Consider a proof-tree $\mathbf{t}$ for a type γ. The corresponding *term-scheme* $N_{\mathbf{t}}$ is the term whose Böhm-tree is obtained from $\mathbf{t}$ by:

- first substituting every node s that descends from the ith tail-variable of another primitive part $p_{s'}$ in $\mathbf{t}$, by $\lambda x_{s_1} \dots x_{s_k}.x_s$, with $k \geq 0$, where $s_1, \dots, s_k$ are all sequences $s' \cdot i \cdot 1, \dots, s' \cdot i \cdot k \in D_{\mathbf{t}_\gamma}$;
- then, removing the top node labelled with p_ϵ.

Finally, we obtain the finite set, $\texttt{Terms}(\mathbf{t})$ of long normal inhabitants of γ from $N_{\mathbf{t}}$ by renaming all variables in abstraction sequences with identical names and renaming the free occurrences of these variables in the scope of these abstraction sequences in all possible ways[2].

[2]The precise algorithm can be found in [12].

Example 16 Consider again the proof-tree $\mathbf{t}$ for the type γ in example 12. We have

$$N_{\mathbf{t}} = \lambda x_1 x_2 x_3.x_1(\lambda x_{111}.x_1(\lambda x_{111}.x_2 x_{111})x_{111})x_3, \quad \text{and}$$

$$\begin{array}{rcl} \texttt{Terms}(\mathbf{t}) & = & \{\lambda x_1 x_2 x_3.x_1(\lambda x_{111}.x_1(\lambda x'_{111}.x_2 x_{111})x_{111})x_3, \\ & & \ \ \lambda x_1 x_2 x_3.x_1(\lambda x_{111}.x_1(\lambda x'_{111}.x_2 x'_{111})x_{111})x_3\} \\ & =_\alpha & \{\lambda xyz.x(\lambda u.x(\lambda v.yu)u)z,\ \lambda xyz.x(\lambda u.x(\lambda v.yv)u)z\ \}. \end{array}$$

4 Type Inhabitation through Game Semantics

In this section, we describe a method for studying type-inhabitation, based on game-semantics, introduced in [5]. We begin recalling some notions from game-semantics, in the context of type inhabitation. The definitions and results in this section are the ones presented in [5], following the restriction in [21]. However, at times we adapt notation according to what we defined in Section 2. Note also that, for the reasons given in Section 2, our presentation of this method is in terms of inhabitation of types, contrary to [5], where it was presented in terms of inhabitation of typings. This choice simplifies the exposition and also makes the establishment of the relationship between the two methods easier.

4.1 Arenas, Games and Winning Strategies

In the following, we consider an *arena* associated to a type as a labelled tree. For a given type γ, a *move* in the arena of γ is a finite sequence s of natural numbers, i.e. $s \in \mathbb{N}_+^\star$.

Definition 17 Let γ be a simple type. The *arena* associated to the type γ, denoted by $A_\gamma = (M_\gamma, \tau_\gamma)$, where M_γ is a set of *moves* and $\tau_\gamma : M_\gamma \longrightarrow \mathcal{A}$ is a typing function mapping moves to atomic types, is defined by $M_\gamma = D_{\mathbf{t}_\gamma}$ and $\tau_\gamma(s) = \mathbf{t}_\gamma(s)$, where $\mathbf{t}_\gamma$ is the tree associated with type γ.

Example 18 Consider again type $\gamma = ((a \to b) \to a \to b) \to (a \to b) \to a \to b$, from Example 3. The arena A_γ is given by: $M_\gamma = \{\epsilon, 1, 11, 12, 111, 2, 21, 3\}$ and $\tau_\gamma(\epsilon) = \tau_\gamma(1) = \tau_\gamma(11) = \tau_\gamma(2) = b$ and $\tau_\gamma(12) = \tau_\gamma(111) = \tau_\gamma(21) = \tau_\gamma(3) = a$.

Definition 19 Let (M, τ) be a given arena. The function $pl : M \to \{O, P\}$, which associates moves in the arena to players, where P and O represent proponent and opponent, respectively, is defined by $pl(\epsilon) = O$ and $pl(s_2) = \overline{pl}(s_1)$, if $s_1 \vdash s_2$, where $\overline{pl}$ is called the inverse function of pl and is such that, $pl(s) = O$ iff $\overline{pl}(s) = P$, for every move $s \in M$ and $\overline{\overline{pl}} = pl$.

Note, that one has $pl(s) = P$ iff $s \in M$ is a sequence of odd length.

Finite sequences of moves $m_1 \dots m_n$ will be denoted by $S, S_1, \dots$ and we write $S_1 \sqsubseteq S_2$, if a sequence of moves S_1 prefixes the sequence S_2.

Definition 20 A finite sequence of moves of the form $s_1 \dots s_n$ is said to be *justified* iff for each s_i, with $1 \leq i \leq n$, there exists a move s_j, such that $j < i$, which enables s_i, i.e. $s_j \vdash s_i$.

A justified sequence is represented as $(s_1, 1, 0) \cdots (s_n, n, l)$, with (s, i, j) denoting the move in position i, which is justified by the move in position j. Obviously, not all moves in a justified sequence are relevant for player P. These are discarded in the P-view of the sequence.

Definition 21 Let S be a finite sequence of moves. The *P-view* of S, denoted by $\ulcorner S \urcorner$, is inductively defined as:

$$\begin{array}{ll} \ulcorner \epsilon \urcorner = \epsilon & \\ \ulcorner S \cdot (s, i, j) \urcorner = \ulcorner S \urcorner \cdot (s, i, j) & \textit{if } s \textit{ is a } P\textit{-move} \\ \ulcorner S_1 \cdot (s_1, j, k) \cdot S_2 \cdot (s_2, i, j) \urcorner = \ulcorner S_1 \urcorner \cdot (s_1, j, k) \cdot (s_2, i, j) & \textit{if } s_2 \textit{ is a } O\textit{-move} \end{array}$$

Definition 22 A justified sequence $S = s_1 \dots s_n$ is a *legal position* if it satisfies the following conditions:

- $s_1 = \epsilon$;
- for $S = S_1 \cdot (s, i, j) \cdot (s', i+1, k) \cdot S_2$, one has $pl(s) = \overline{pl}(s')$;
- $\ulcorner S \urcorner = S$.

A game is generally defined by associating a set of positions to an arena. In this context, a *game* will be an arena with an associated set of prefix-closed legal positions. Because a legal position is a P-view, each occurrence in the legal position of an O-move different from ϵ, is justified by the immediate preceding move, which corresponds to a P-move. Therefore, the following notation can be used:

- if an occurrence of an O-move s is in position $2i+1$ (from left-to-right) in sequence S, then it is denoted by (s, i);
- if an occurrence of a P-move s in S, is justified by a preceding occurrence (s', i) of an O-move s', then it is denoted by (s, i).

However, when convenient we sometimes will omit references in these sequences, writing $s_1, \ldots, s_n$ instead of $(s_1, i_1), \ldots, (s_n, i_n)$.

Definition 23 Consider an arena $A = (M, \tau)$ and let Σ be a finite non-empty set of prefix-closed legal positions in A. We say that Σ is a *typing strategy* if:

a) Any sequence $S \in \Sigma$ is non-empty and of length even.

b) If $S \cdot s \cdot s_1, S \cdot s \cdot s_2 \in \Sigma$, then $s_1 = s_2$.

c) If $S \cdot s \cdot s' \in \Sigma$, then $\tau(s) = \tau(s')$.

We consider $\mathsf{max}(\Sigma) \subseteq \Sigma$ as the set of sequences which are maximal with respect to $\sqsubseteq$. A typing strategy Σ in an arena $A = (M, \tau)$, is a *winning strategy* if:

d) For all $S \cdot s \in \mathsf{max}(\Sigma)$, there is no O-move s', such that $s \vdash s'$.

e) If $S \cdot s_1 \in \Sigma$ and $S \cdot s_1 \cdot s$ is a legal position in A, then there is a P-move s_2 such that $S \cdot s_1 \cdot s \cdot s_2 \in \Sigma$.

Example 24 Consider again type γ as before and its arena given in Example 18. A winning strategy for γ is the following.

$$\begin{array}{ll} \Sigma = \{ & (\epsilon,0)\cdot(1,0), \\ & (\epsilon,0)\cdot(1,0)\cdot(11,1)\cdot(1,0), \\ & (\epsilon,0)\cdot(1,0)\cdot(11,1)\cdot(1,0)\cdot(11,2)\cdot(2,0), \\ & (\epsilon,0)\cdot(1,0)\cdot(11,1)\cdot(1,0)\cdot(11,2)\cdot(2,0)\cdot(21,3)\cdot(111,2), \\ & (\epsilon,0)\cdot(1,0)\cdot(11,1)\cdot(1,0)\cdot(12,2)\cdot(111,1), \\ & (\epsilon,0)\cdot(1,0)\cdot(12,1)\cdot(3,0) \ \} \end{array}$$

4.2 From winning strategies to long normal inhabitants and back

In [5] it was shown that it is possible to establish a bijection between the set of winning strategies in the arena of a type, and the set of its long normal inhabitants, through an interpretation of winning strategies as λ-terms.

The *preceding relation* $\prec$ on sequences is defined by $s_1 \prec s_2$ iff there exists $i, j \in \mathbb{N}_+$ and $s \in \mathbb{N}_+^\star$, such that $s_1 = s \cdot i$, $s_2 = s \cdot j$ and $i < j$.

Definition 25 Let $A = (M, \tau)$ be an arena and Σ a strategy on A. The *arborescent reading* of Σ (which is a tree representation of $\mathsf{max}(\Sigma)$), is denoted $\mathbb{T}_\Sigma$, and defined inductively as $\mathbb{T}_\Sigma = s_1 \cdot s_2[\mathbb{T}_{\Sigma_1}, \ldots, \mathbb{T}_{\Sigma_p}]$ with:

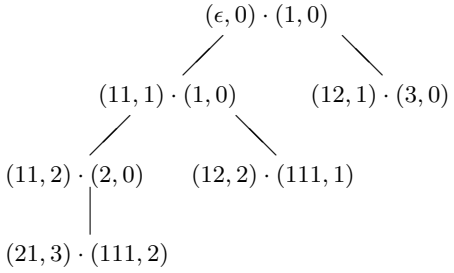
- every $S \in \mathsf{max}(\Sigma)$ is of the form $s_1 \cdot s_2 \cdot S'$;

- $\{s'_1, \ldots, s'_p\} = \{s' \mid s_2 \vdash s'\}$ and $s'_i \prec s'_j$ iff $i < j$;
- $\Sigma_i = \{s'_i \cdot S' \mid s_1 \cdot s_2 \cdot s'_i \cdot S' \in \Sigma\}$.

Example 26 The arborescent reading of the strategy Σ in Example 24, is:

$$\mathbb{T}_\Sigma = (\epsilon, 0) \cdot (1, 0)[(11, 1) \cdot (1, 0)[(11, 2) \cdot (2, 0)[(21, 3) \cdot (111, 2)], (11, 2) \cdot (111, 1)], (12, 1) \cdot (3, 0)]$$

which can be pictured as:



The interpretation of a strategy in an arena is defined on the arborescent reading of the strategy.

Definition 27 Let $\mathbb{T}_\Sigma$, be the arborescent reading of a strategy Σ in an arena $A = (M, \tau)$. The *interpretation of* Σ on A, $[\![\Sigma]\!] = [\![\mathbb{T}_\Sigma, \emptyset]\!]$ is defined inductively on $\mathbb{T}_\Sigma$:

$$[\![(s, i) \cdot (s', j)[\mathbb{T}_{\Sigma_1}, \ldots, \mathbb{T}_{\Sigma_p}], V]\!] = \lambda x_1 \ldots x_p . x [\![\mathbb{T}_{\Sigma_1}, W]\!] \ldots [\![\mathbb{T}_{\Sigma_q}, W]\!]$$

where s is a O-move, s' is a P-move, $W = V \cup \{((s \cdot k, i), x_k) \mid s \cdot k \in M\}$ with $((s', j), x) \in W$ and x_k is a fresh variable for $1 \leq k \leq p$.

Example 28 Consider again the arborescent reading of $\mathbb{T}_\Sigma$ from Example 26:

$$\underbrace{(\epsilon, 0) \cdot (1, 0)\,[(11, 1) \cdot (1, 0)\,\underbrace{[(11, 2) \cdot (2, 0)[(21, 3) \cdot (111, 2)]}_{\mathbb{T}_{\Sigma_2}}, (11, 2) \cdot (111, 1)], (12, 1) \cdot (3, 0)]}_{\mathbb{T}_{\Sigma_1}}$$

Then, with $W = \{((1, 0), x), ((2, 0), y), ((3, 0), z)\}$, $W_1 = W \cup \{((111, 1), u)\}$ and $W_2 = W_1 \cup \{((111, 2), v)\}$:

$$\begin{array}{rcll} [\![\Sigma]\!] & = & [\![\mathbb{T}_\Sigma, \emptyset]\!] & = \lambda xyz.x [\![\mathbb{T}_{\Sigma_1}, W]\!]\ [\![(12, 1) \cdot (3, 0)[\,], W]\!] \\ & & & = \lambda xyz.x (\lambda u.x [\![\mathbb{T}_{\Sigma_2}, W_1]\!]\ [\![(11, 2) \cdot (111, 1)[\,], W_1]\!])\ z \\ & & & = \lambda xyz.x (\lambda u.x (\lambda v.y [\![(21, 3) \cdot (111, 2)[\,], W_2]\!])\ u) z \\ & & & = \lambda xyz.x (\lambda u.x (\lambda v.yv) u) z \end{array}$$

It was proved in [5], that given an arena A associated to a typing $< \Gamma, \gamma >$, and a winning strategy Σ on A, then $< \Gamma, \gamma >$ is a typing pair for $[\![\Sigma]\!]$. Given a typing judgment $\Gamma \vdash M : \gamma$, it is also possible to define, by induction on M, a winning strategy Σ, such that $[\![\Sigma]\!] =_{\beta\eta} M$ (details can be found in the Appendix of [5]).

5 Winning Strategies and Proof Trees

In this section, we establish the close relationship that exists between winning strategies and proof-trees, presenting two transformation algorithms between the two. We begin by simplifying[3] some notions of the game-semantic approach, with the following lemmas, that follow directly from the definitions given in the previous section.

Lemma 29 Let $A = (M, \tau)$ be an arena and $S = s_1 \ldots s_n$ a finite sequence of moves in A. S is a legal position if and only if the following conditions hold:

[3]For instance, there will be no longer need for concepts such as function pl, P-view of a sequence, etc.

a) every sequence S begins with $s_1 = \epsilon$ and alternates O-moves (moves of even length) with P-moves (moves of odd length);

b) if $S' \cdot s_i \cdot s_{i+1} \sqsubseteq S$, where s_i is a P-move, then $s_i \vdash s_{i+1}$ (i.e. every O-move different from s_1 is enabled by the immediate preceding P-move in S);

c) for every P-move s_i in S there is some O-move s_j, with $j < i$, and such that $s_j \vdash s_i$ (i.e. every P-move in S is enabled by some preceding O-move).

Proof Straightforward from the definition of $\ulcorner S \urcorner$ and Definitions 20 and 22. •

Lemma 30 Let $A = (M, \tau)$ be an arena and Σ a non-empty prefix-closed set of legal positions of even length in A. Then, Σ is a winning strategy if and only if the following conditions hold.

a) if $S \cdot s_i \in \Sigma$, where s_i is a P-move, then for every $s_j \in M$ such that $s_i \vdash s_j$, there is some $S_j \in \Sigma$ such that $S \cdot s_i \cdot s_j \sqsubseteq S_j$;

b) if $S \cdot s_1 \in \Sigma$ and s_1 is a P-move, then there is no other $S \cdot s_2 \in \Sigma$ such that $s_1 \neq s_2$;

c) if $S \cdot s_i \cdot s_{i+1} \in \Sigma$, where s_i is an O-move (and s_{i+1} a P-move), then $\tau(s_i) = \tau(s_{i+1})$.

Proof Note first, that both Definition 23 as well as Lemma 30 are restricted to sequences of even length. Furthermore, conditions b) and c) in Lemma 30 are equivalent to conditions b) and c) in Definition 23. Finally, since all sequences in Σ are of even length, condition a) in Lemma 30 is equivalent to conditions d) and e) in Definition 23. •

Comparing this alternative definition of winning strategies with the one, given in Section 3, for proof-trees, the similarity of both methods becomes obvious; and in particular it is straightforward to define transformation algorithms, from one approach to the other and back.

Definition 31 Let γ be a type and Σ a winning strategy in $A_\gamma = (M_\gamma, \tau_\gamma)$. The tree $\mathtt{PT}_\Sigma$ is obtained by substituting in the arborescent reading $\mathbb{T}_\Sigma$ every label $(\_, \_) \cdot (\mathbf{s}, \_)$ by $p_{\mathbf{s}}$ and adding an additional root-node with label p_ϵ.

Example 32 It is easy to see, that the proof-tree $\mathtt{PT}_\Sigma$ corresponding to γ and Σ from Examples 24 and 26, is the proof-tree given in Example 12.

Proposition 33 If Σ is a winning strategy in an arena $A_\gamma = (M_\gamma, \tau_\gamma)$, then $\mathtt{PT}_\Sigma$ is a proof-tree for γ.

Proof $\mathtt{PT}_\Sigma$ is obtained by substituting in the arborescent reading $\mathbb{T}_\Sigma$ every label $(\_, \_) \cdot (\mathbf{s}, \_)$ by $p_{\mathbf{s}}$ and adding an additional root-node with label p_ϵ.

Since, in every label $(s', i), (s, j)$ in $\mathbb{T}_\Sigma$, s' and s are respectively an O-move and a P-move, it follows that the root-node of $\mathtt{PT}_\Sigma$ is labelled by p_ϵ and every other node in $\mathtt{PT}_\Sigma$ is labelled by p_s for some $s \in M_\gamma = D_{\mathbf{t}_\gamma}$ of odd length, and consequently corresponds to exactly one primitive part in the formula-tree of γ.

In order to show that $\mathtt{PT}_\Sigma$ is a proof-tree for γ, we show that all conditions in Definition 11 are satisfied. Condition i. follows directly from the definition of $\mathtt{PT}_\Sigma$. Condition ii. follows from conditions a), b) and c) in Lemma 30. Finally condition iii. is guaranteed by condition c) of Lemma 29. •

We now present the inverse transformation, that given a proof-tree $\mathtt{PT}$ for γ constructs a finite set of winning strategies in the arena $A_\gamma = (M_\gamma, \tau_\gamma)$. In the first step, we compute a tree $\mathbb{T}_{\mathtt{PT}}$, where each node is labelled with two pairs $(s^O, m) \cdot (s^P, \{n_1, \ldots, n_k\})$, where s^O and s^P are respectively an O-move and a P-move, m is the reference of this node, and $\{n_1, \ldots, n_k\}$ is the set of all references to nodes that can enable the P-move s^P in this path.

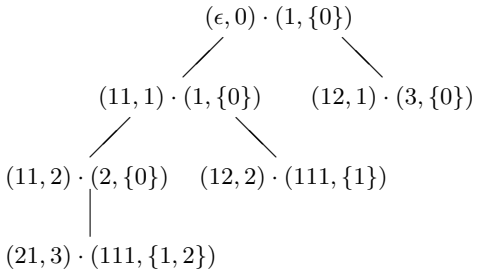
The construction of $\mathbb{T}_{\mathtt{PT}}$ is done top-down, creating for each node p_s in $\mathtt{PT}$ that descends from the i-th tail-variable of another primitive part $p_{s'}$ in $\mathtt{PT}$[4], a node labelled by $(s' \cdot i, \mathsf{depth}) \cdot (s, \mathsf{Ref})$, where

[4]Note that there will be no node created for the top-node p_ϵ in $\mathtt{PT}$.

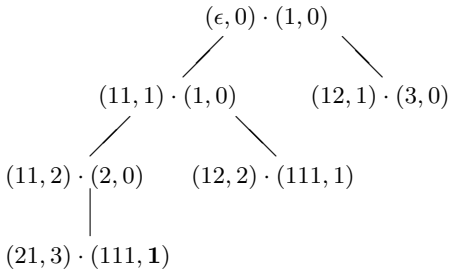
- depth is the depth of this node in $\mathbb{T}_{\mathtt{PT}}$ and
- Ref is the set containing all references d, such that there is some ancestor of this node with label $(s^-, \mathsf{d}), (\_, \_)$ with $s^- \vdash s$, as well as the reference depth, whenever $s' \cdot i \vdash s$.

Then, the set of winning strategies $\mathtt{WS}(\mathtt{PT})$ is the set of strategies Σ with arborescent reading $\mathbb{T}_\Sigma$ consistent with $\mathbb{T}_{\mathtt{PT}}$; meaning that $\mathbb{T}_\Sigma$ has the same structure as $\mathbb{T}_{\mathtt{PT}}$ and such that each node labelled with $(s^O, m) \cdot (s^P, \{n_1, \ldots, n_k\})$ in $\mathbb{T}_{\mathtt{PT}}$ is labelled with $(s^O, m) \cdot (s^P, n_i)$ in $\mathbb{T}_\Sigma$, where $i \in \{1, \ldots, k\}$.

Example 34 Consider again the proof-tree $\mathtt{PT}$ from Example 12 for type γ as before. We have, $\mathbb{T}_{\mathtt{PT}} =$



Thus, $\mathtt{WS}(\mathtt{PT}) = \{\Sigma_1, \Sigma_2\}$, where Σ_1 is the winning strategy from Example 24 and Σ_2 has the following arborescent reading:



Note also, that these two strategies represent precisely the two λ-terms computed in Example 16.

Proposition 35 If $\mathtt{PT}$ is a proof-tree for γ, then $\mathtt{WS}(\mathtt{PT})$ is a finite set of winning strategies in the arena $A_\gamma = (M_\gamma, \tau_\gamma)$. Furthermore,

$$\mathtt{WS}(\mathtt{PT}) = \{\ \Sigma \mid \Sigma \text{ is a winning strategy in } A_\gamma \text{ such that } \mathtt{PT}_\Sigma = \mathtt{PT}\ \}.$$

Proof First, we note that each $\Sigma \in \mathtt{WS}(\mathtt{PT})$ is a set of legal positions in A_γ. In fact, by construction, every sequence in S begins with $s_1 = \epsilon$ and alternates O-moves with P-moves. Also every O-move $s' \cdot i$ is enabled by the immediate preceding P-move s' in S. Furthermore, by the definition of Ref, every P-move in S is enabled by some preceding O-move.

The conditions in Lemma 30, guaranteeing that Σ is in fact a winning strategy follow easily from $\mathtt{PT}$ being a proof-tree. And finally, any winning strategy Σ such that $\mathtt{PT}_\Sigma = \mathtt{PT}$ is such that $\mathbb{T}_\Sigma$ is consistent with $\mathbb{T}_{\mathtt{PT}}$. •

5.1 Some considerations on the expressiveness of both methods

We saw that the notions of proof-tree and winning strategy are essentially the same, but for references to enabling variables, which are present in winning strategies and missing in proof-trees. In fact, a winning strategy represents exactly one long normal inhabitant and consequently a finite family of normal inhabitants of a type. On the other hand, a proof-tree represents a finite set of long normal inhabitants, corresponding to a possibly bigger finite family of normal inhabitants, which share important properties such as principality, etc. (cf. [7]).

As such, it is natural that, in the past, both methods have been used for similar purposes. For instance, the formula-tree method was used in 2000, cf. [7], to characterize principal typings of β-normal terms. An equivalent characterization in terms of game-semantics was given in 2011,

cf. [5]. Also, both methods have been used (respectively in 2002 and 2011) to present a concise proof of Aoto's theorem, which states that negatively non-duplicating types, have at most one normal inhabitant (cf. [10] and [5]).

In fact, it seems as if most notions in the game-semantics approach translate easily to the formula-tree approach. For instance, in [5], given a long inhabitant N of a type (typing), a binary relation I_{ij}^N on the variables in N is defined by $xI_{ij}^N y$ if and only if there is a subterm of N of the form $xN_1 \dots N_{i-1}(\lambda x_1 \dots x_{j-1} y x_{j+1} \dots x_n.N')N_{i+1} \dots N_m$. This relation is used to define another binary relation $\approx_N$ on variables in N. Given two variables x and y, one has $x \approx_N y$ iff

- $x = y$;
- there are two variables z_1 and z_2 in N such that $z_1 I_{ij}^N x$, $z_2 I_{ij}^N y$ and $z_1 \approx_N z_2$.

This relation is then recursively extended to a relation $\approx_N$ on subterms of N, defining $N_1 \approx_N N_2$ iff

- $N_1 = x_1$, $N_2 = x_2$ and $x_1 \approx_N x_2$;
- $N_1 = \lambda x_1.P_1$, $N_2 = \lambda x_2.P_2$, $x_1 \approx_N x_2$ and $P_1 \approx_N P_2$;
- $N_1 = x_1 P_1 \dots P_n$, $N_2 = x_2 Q_1 \dots Q_n$, $x_1 \approx_N x_2$ and $P_i \approx_N Q_i$, for $i = 1, \dots, n$.

Example 36 Consider $N = \lambda xyz.y(\lambda u.x(y(\lambda v_1.u))(y(\lambda v_2.v_2))(y(\lambda v_3.z))$. For the subterms $N_1 = y(\lambda v_1.u)$, $N_2 = y(\lambda v_2.v_2)$ and $N_3 = y(\lambda v_3.z)$, one has $N_1 \approx_N N_2$, but $N_1 \not\approx_N N_3$.

Finally, based on this relation, a syntactic characterization of the inhabitants of negatively non-duplicating types, called first-order copying terms is provided: a long normal inhabitant for its principal type is first-order copying if every two subterms N_1 and N_2 are assigned the same type iff $N_1 \approx_N N_2$.

Example 37 The term N from the previous example is a long normal inhabitant for its principal type $(a \to a \to a \to b) \to ((b \to b) \to a) \to b \to a$. All subterms N_1, N_2 and N_3 are assigned the same type a. Since $N_1 \not\approx_N N_3$, one concludes that the term N is not first-order copying, and consequently no inhabitant of a negatively non-duplicating type.

Now, it can be easily seen that, in terms of proof-trees $\approx_N$ translates directly to the identity on trees. As such, one has $N_1 \approx_N N_2$ iff $\mathbf{t}_{N_1} = \mathbf{t}_{N_2}$. Consequently, the characterization of the inhabitants of negatively non-duplicating typings, given in [5], can be expressed in the formula-tree approach, replacing $N_1 \approx_N N_2$ by $\mathbf{t}_{N_1} = \mathbf{t}_{N_2}$.

Example 38 In fact, for the subterms in the previous example the relation $\approx_N$ is easy to establish by observing their corresponding proof-trees.

$$\mathbf{t}_{N_1} = \mathbf{t}_{N_2} = \begin{array}{c} \mathtt{p_2} \\ | \\ \mathtt{p_{21}} \end{array} \qquad \text{and} \qquad \mathbf{t}_{N_3} = \begin{array}{c} \mathtt{p_2} \\ | \\ \mathtt{p_3} \end{array}$$

On the other hand, results that depend on the absence/presence of references to enabling variables, cannot be transferred directly from one world to the other. As an example, it was shown in [27] that it is possible to describe the set of normal inhabitants of a type, using an infinitary extension of the concept of context-free grammar, which allows for an infinite number of non-terminal symbols as well as production rules. The set of normal inhabitants corresponds then to the set of terms generated by this, possibly infinitary, grammar plus all terms obtained from those by η-reduction. Later, using the formula-tree approach, it has been shown, cf. [12], that for every type γ there is in fact a finite context-free grammar G_γ from which all normal inhabitants of γ can be obtained. The existence of this grammar relies on the absence of references and it seems that there is no straightforward counterpart to the construction of a finite grammar in terms of game-semantics.

6 Conclusions

In this paper we revisited two tools for studying inhabitation in the simply typed λ-calculus. Both methods explore the close relationship between the structure of types and their inhabitants and both have been used to obtain new results, as well as significantly simplifying previous proofs of other, already known, results.

Although defined in two very different contexts, the two approaches are very closely related. In this paper we highlighted the links between these two approaches, which up to now were considered separately, by gradually refining the notation, in order to easily define simple translation algorithms between the two methods.

In spite of the fact that the two approaches are equivalent, they are not completely identical and we briefly discussed how different problems may benefit from the features of one method or the other.

7 Bibliography

References

[1] T. Aoto. Uniqueness of normal proofs in implicational intuitionistic logic. *J. of Logic, Lang. and Inf.*, 8(2):217–242, 1999.

[2] A.A. Babaev and S.V. Solov'ev. A coherence theorem for canonical morphisms in cartesian closed categories. *Journal of Mathematical Sciences*, 20:2263–2279, 1982.

[3] H.P. Barendregt. Lambda Calculi with Types. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Clarendon Press, Oxford, 1992.

[4] Ch. Ben-Yelles. *Type Assignment in the Lambda-Calculus: Syntax and Semantics*. PhD thesis, University College of Swansea, September 1979.

[5] P. Bourreau and S. Salvati. Game semantics and uniqueness of type inhabitance in the simply-typed λ-calculus. In *TLCA'11*, volume 6690 of *LNCS*, pages 61–75. Springer, 2011.

[6] Pierre Bourreau and Sylvain Salvati. A datalog recognizer for almost affine -cfgs. In Makoto Kanazawa, András Kornai, Marcus Kracht, and Hiroyuki Seki, editors, *MOL*, volume 6878 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2011.

[7] S. Broda and L. Damas. On the structure of normal λ-terms having a certain type. In *Proc. 7th WoLLIC'2000*, pages 33–43, 2000.

[8] S. Broda and L. Damas. A context-free grammar representation for normal inhabitants of types in TA_λ. In *EPIA'01*, volume 2258 of *LNCS*, pages 321–334. Springer, 2001.

[9] S. Broda and L. Damas. Counting a type's (principal) inhabitants. *Fundam. Inform.*, 45(1-2):33–51, 2001.

[10] S. Broda and L. Damas. Studying provability in implicational intuitionistic logic: the formula tree approach. *ENTCS*, 67:131–147, 2002.

[11] S. Broda and L. Damas. Formula Tree Lab. `http://www.dcc.fc.up.pt/~sbb/FTLab/ftlab/ftlab.html`, 2003.

[12] S. Broda and L. Damas. On long normal inhabitants of a type. *J. Log. and Comput.*, 15:353–390, June 2005.

[13] S. Broda, L. Damas, M. Finger, and P. Silva e Silva. The decidability of a fragment of BB'IW-logic. *Theor. Comput. Sci.*, 318(3):373–408, 2004.

[14] M.W. Bunder. Proof finding algorithms for implicational logics. *Theoretical Computer Science*, 232(12):165 – 186, 2000.

[15] Philippe de Groote. Towards abstract categorial grammars. In *ACL*, pages 148–155. Morgan Kaufmann Publishers, 2001.

[16] I. Guessarian. *Algebraic Semantics*, volume 99 of *LNCS*. Springer, 1981.

[17] R. Harmer, M. Hyland, and P. Melliès. Categorical combinatorics for innocent strategies. In *LICS*, pages 379–388, 2007.

[18] J.R. Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.

[19] S. Hirokawa. Infiniteness of proof (α) is polynomial-space complete. *Theor. Comput. Sci.*, 206(1-2):331–339, 1998.

[20] W.A. Howard. The formulas-as-types notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.

[21] A.D. Ker, H. Nickau, and C.-H. Luke Ong. Innocent game models of untyped lambda-calculus. *Theor. Comput. Sci.*, 272(1-2):247–292, 2002.

[22] Y. Komori and S. Hirokawa. The number of proofs for a BCK-formula. *J. Symb. Log.*, 58(2):626–628, 1993.

[23] G.E. Mints. Closed categories and the theory of proofs. *Journal of Mathematical Sciences*, 15:45–62, 1981.

[24] D. Prawitz. *Natural deduction: a proof-theoretical study*. PhD thesis, Almqvist & Wiksell, 1965.

[25] S. Salvati. *Problmes de filtrage et problmes danalyse pour les grammaires catgorielles abstraites*. PhD thesis, Institut National Polytechnique de Lorraine, 2005.

[26] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci.*, 9:67–72, 1979.

[27] M. Takahashi, Y. Akama, and S. Hirokawa. Normal proofs and their grammar. *Information and Computation*, 125(2):144–153, 1996.

[28] P. Urzyczyn. Inhabitation in typed lambda-calculi (a syntactic approach). In *TLCA'97*, volume 1210 of *LNCS*, pages 373–389. Springer, 1997.

[29] P. Urzyczyn. Inhabitation of low-rank intersection types. In *TLCA'09*, volume 5608 of *LNCS*, pages 356–370. Springer, 2009.