Higher-Order Pushdown Trees are Circularly Computable

Jérôme Fortier

Higher-Order Pushdown Trees are Circularly Computable

Jérôme Fortier

1: Laboratoire de Combinatoire et d'Informatique Mathématique (LaCIM), Université du Québec à Montréal.
2: Laboratoire d'Informatique Fondamentale (LIF), Aix-Marseille Université.
`jerome.fortier@lif.univ-mrs.fr`

Résumé

We are interested in the problem of expressiveness of the following operations in the category of sets: finite products and coproducts, initial algebras (induction) and final coalgebras (coinduction). These operations are the building blocks of a logical system that allows circularity in Gentzen-style proofs. Proofs in this system are seen as simple programs, while the cut-elimination process is viewed as a running automaton with a memory device. In this paper, we show that higher-order pushdown trees, those accepted by higher-order pushdown automata, are computable in this setting, by providing an explicit simulation of the automata by cut-elimination.

1. Introduction

Circular proofs were introduced in [11] and recently improved in [5] as a means of expressing the arrows of categories with finite products, finite coproducts, initial algebras and final coalgebras. Such categories are called μ-bicomplete in [12]. If we restrict our attention to the μ-bicomplete category **Set** of sets and functions, the semantical results from [5, 4] mean that we have a way to express, via a logical system, the functions definable purely from induction, coinduction, finite cartesian products and finite disjoint unions (without using the *closedness* of the category, which is the usual interpretation of λ-abstractions). We call such functions *circularly definable* (**CD**). Circular proofs can then be used as a combinatorial tool for tackling the following question: which functions, with respect to other known classes of functions, are the circularly definable ones?

Circular proofs are Gentzen-style proofs in which cyclic reasoning is allowed, given that the cycles satisfy some combinatorial property that we call *guard condition.* The main improvement from [11] to [5] is the sound addition of the cut rule, with which we have a *fullness* property: every arrow from the free μ-bicomplete category is expressible as a circular proof. Using this property and known semantical results from [2, 10], we can conclude, for instance, that the primitive recursive functions $f : \mathbb{N}^k \to \mathbb{N}$ are circularly definable.

In this paper, we are more interested in a syntactical approach to the problem. Our interest is not only in the theoretical question of circular *definability*, but also in the concrete problem of circular *computability.* In [5], we have described a cut-elimination procedure for circular proofs. The procedure works like an automaton with memory, that uses a (finite) proof with cycles as its set of states and produces a possibly infinite proof tree, which is a cut-free unfolded version of the original proof. The Guard condition ensures the productivity of the procedure. In other words, the cut-eliminating automaton is a sort of abstract machine for computing (at least) circularly definable functions, or an interpreter of circular proofs. The functions that are computable by the cut-eliminator, given a finite (not necessarily guarded) pre-proof to work with, are called *circularly computable* (**CC**). In order to understand better which functions are circularly computable we need to understand better the computing mechanism described by the abstract cut-eliminator and its storage device, by comparison with other models of abstract machines.

We focus our attention on circular proofs that compute infinite trees labeled over a signature (Σ-trees) and compare these trees with the possible outputs of higher-order pushdown automata (n-PDAs). The complexity of the memory structure (stacks of stacks of... stacks) of these automata determine a well-established hierarchy of infinite trees known as the Caucal hierarchy [1, 7]. A natural question is then: can we locate the expressive power of the cut-eliminator in the hierarchy?

It turns out, and we will see, that higher-order pushdown automata can always be simulated by the cut-eliminator, but that the converse is not true. In other words, circular computability completely bypasses the Caucal hierarchy. However, our construction does not always give a valid circular proof, suggesting that there may be some gap between circular definability and circular computability, that we have yet to understand.

The core of the paper is divided into two sections. In Section 2, we recall some facts about circular proofs and cut-elimination and we show how they can define and compute Σ-trees. In Section 3, we show how to encode an arbitrary higher-order pushdown automaton into a circular pre-proof, on which cut-elimination simulates the behavior of the automaton.

2. Circular Proofs and Σ-trees

This Section is an informal tutorial about circular proofs and cut-elimination (formal definitions and proofs can be found in [4, 5, 11, 12]). We illustrate the theory mainly with the example of trees over a signature Σ in order to define the classes **CD** and **CC** of *circularly definable* and *circularly computable* trees.

2.1. Directed systems of equations

Fix a set $\mathbb{V}$ of variables. A *directed system of equations* is a collection $\mathcal{S}$ of formal expressions of the form "$X =_{p_X} F_X$", where X ranges over a finite subset $\mathsf{BV}(\mathcal{S}) \subset \mathbb{V}$, called the set of *bound variables* of $\mathcal{S}$. For all $X \in \mathsf{BV}(\mathcal{S})$, $p_X \in \mathbb{N}$ is called the *priority* of X, and F_X is a *term* associated to X. Terms are well-formed expressions constructible from $\mathbb{V}$ using binary function symbols $\times, +$ and constants 0 and 1. We may freely use $\prod$ (resp. $\coprod$) to shorten the notation when many instances of $\times$ (resp. $+$) are nested. An empty product just means the term 1 and an empty coproduct means the term 0. The set $\mathsf{FV}(\mathcal{S})$ of *free variables* of $\mathcal{S}$ consists of all the variables that occur in some F_X, but do not belong to $\mathsf{BV}(\mathcal{S})$.

We use directed systems of equations simply as an alternative syntax to write fixpoint expressions from the lattice μ-calculus [12]. In **Set**, 0 denotes the empty set $\varnothing$ (initial object), 1 denotes a singleton that we may call $\mathbf{1}$ (final object), $\times$ and $+$ are interpreted respectively as cartesian product and coproduct (disjoint union), an *odd priority* means "least solution", an *even priority* means "greatest solution" and the value of the priority grows with the scope of the variable. For instance, the fixpoint expression $\nu X.(\mu N.(1 + N) \times X)$, that the accustomed eye would recognize as denoting the set $\mathbb{N}^{\mathbb{N}}$ of streams of natural numbers, can be transcribed as the following system:

$$\mathcal{S} : \begin{cases} X =_2 N \times X \\ N =_1 1 + N. \end{cases} \tag{1}$$

With respect to the solution of a directed system of equation $\mathcal{S}$ in a μ-bicomplete category $\mathbf{C}$, each term τ denotes a functor $[\![\tau]\!]^{\mathcal{S}}_{V} : \mathbf{C}^{V} \to \mathbf{C}$ for each finite set $V \subseteq \mathbb{V}$ such that $\mathsf{FV}(\mathcal{S}) \subseteq V$

and $\mathsf{BV}(\mathcal{S}) \cap V = \varnothing$. In this paper, we work mainly in the category $\mathbf{C} = \mathbf{Set}$. The exact set V in consideration is not important, given that it is large enough. In fact, taking $V = \varnothing$ works fine in this paper since we will be dealing with terms that only denote sets (constant functors). Since the system $\mathcal{S}$ will be understood from context, we shall then just write $[\![\tau]\!]$. For instance, in (1), we have $[\![N]\!] = \mathbb{N}$ and $[\![X]\!] = \mathbb{N}^{\mathbb{N}}$.

Next, we define a directed system for denoting the central structure of this paper, Σ-trees, which happen to be the outputs of higher-order pushdown automata that we will present in the next section.

Definition 2.1. A *signature* is a finite, nonempty set Σ with an arity function $\mathrm{ar} : \Sigma \to \mathbb{N}$. A Σ*-tree* is a tuple $t = (f, t_1 \dots t_r)$ such that $f \in \Sigma$, $r = \mathrm{ar}(f)$ and $t_1 \dots t_r$ are Σ-trees.

Note that Definition 2.1 is somewhat circular and, since it does not rely on a base case, may be iterated infinitely often. It therefore describes trees whose branches are infinite, unless some symbol of arity 0 ends them (which is not a necessity). Let $\mathcal{T}(\Sigma)$ be the system given by the following equations:

$$T =_2 \coprod_{f \in \Sigma} f \qquad , \qquad f =_2 \prod_{j=1}^{\mathrm{ar}(f)} T \quad \text{(for each } f \in \Sigma\text{)}. \tag{2}$$

In order to exhibit an element $t \in [\![T]\!]$, we have to choose a symbol f in the signature and then exhibit an element of the set denoted by $[\![f]\!]$. But doing this just means to choose $\mathrm{ar}(f)$ elements of $[\![T]\!]$. So t is determined by a choice of elements $(f, t_1 \dots t_{\mathrm{ar}(f)})$, as in Definition 2.1. Taking an even priority (greatest solution) means that we allow this process to be iterated infinitely often.

2.2. The proof system

Pre-proofs are built from an intuitionistic sequent calculus with left ($\mathfrak{L}$) and right ($\mathfrak{R}$) rules that depend on a directed system $\mathcal{S}$. The rules are given in Table 1: a hypothesis of the form $\{a_i \vdash b_i\}_{i \in I}$ just means that there is one entry for each $i \in I$. Note that only one term is allowed on each side of the turnstile symbol ($\vdash$).

Identity, Cut, Assumption	$\dfrac{}{a \vdash a}\,\mathtt{Id}$ $\quad \dfrac{a \vdash c \quad c \vdash b}{a \vdash b}\,\mathtt{Cut}$ $\quad \dfrac{}{a \vdash b}\,\mathtt{A}$	
	$\mathfrak{L}$	$\mathfrak{R}$
Products ($\forall I$ finite, $j \in I$)	$\dfrac{a_j \vdash b}{\prod_{i \in I} a_i \vdash b}\,\mathtt{L}\times_j$	$\dfrac{\{a \vdash b_i\}_{i \in I}}{a \vdash \prod_{i \in I} b_i}\,\mathtt{R}\times$
Coproducts ($\forall I$ finite, $j \in I$)	$\dfrac{\{a_i \vdash b\}_{i \in I}}{\coprod_{i \in I} a_i \vdash b}\,\mathtt{L}+$	$\dfrac{a \vdash b_j}{a \vdash \coprod_{i \in I} b_i}\,\mathtt{R}+_j$
Fixpoints ($\forall X \in \mathsf{BV}(\mathcal{S})$)	$\dfrac{F_X \vdash b}{X \vdash b}\,\mathtt{LF}_X$	$\dfrac{a \vdash F_X}{a \vdash X}\,\mathtt{RF}_X$

Table 1: Inference rules over a directed system $\mathcal{S}$

Note that in ordinary sequent calculus, the structure of the proofs is always required to be a finite tree. In order to deal with the semantics of fixpoint expressions, we have to remove that constrtaint.

So, a pre-proof Π is any directed graph with two mappings Rule and Seq of the vertices to rules and sequents respectively, such that the syntax of the rules is locally respected by the sequents.

From an algebraic point of view, these local syntactic constraints establish a system of equations in which the unknowns are the vertices of the pre-proof. The rules determine equations between these variables, according to the standard categorical semantics of inference rules (see [8] and [5, 4]). We say that Π is *resolvable* in a category **C** if a unique solution to the system exists. In that case, we denote the solution at vertex a v such that $\textsc{Seq}(v) = a \vdash b$ by $[\![v]\!]_\Pi : [\![a]\!] \to [\![b]\!]$ or, if Π is well understood, simply by $[\![v]\!]$. The assumption rule allows us to introduce variables in those systems that can eventually be substituted by suitable functions: we use it as a place-holder for further constructions in Section 3.2.

Of course, not every pre-proof should be called a *proof*, since some of them are not resolvable. Consider, for instance, the pre-proof over the system $\mathcal{S}$ of (1) given in Figure 1.

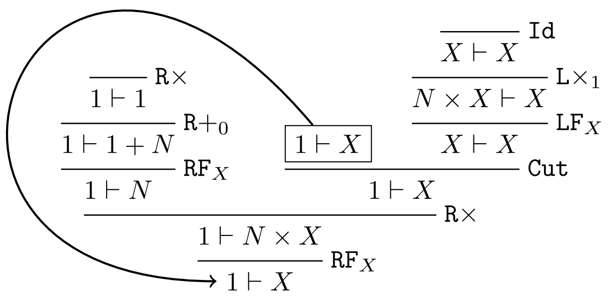

Figure 1: A bad pre-proof

From a computational point of view, we interpret proofs as programs defining functions. So what is the function $s : \mathbf{1} \to \mathbb{N}^{\mathbb{N}}$ denoted by the root of that pre-proof? By extracting the corresponding system of equations, one finds that it must satisfy the following equation:

$$s = \texttt{Cons}(0, \texttt{tail}(s)).$$

But the solution to that equation is far from being unique. In fact, any stream that starts with 0 is a solution, so the pre-proof is not a productive definition of any particular stream.

In order to make sure that *proofs* denote precise functions, we ask valid proofs to satisfy a global condition, inspired by the theory of parity games [12].

Definition 2.2. We say that an infinite path γ in a pre-proof Π has a *right ν-trace* (resp. *left μ-trace*) if it has a suffix γ' such that the following properties hold:

- for all $v \in \gamma'$, if $\textsc{Rule}(v) = \texttt{Cut}$, then the next vertex in γ' is the right (resp. left) premise of v;
- there is some $X \in \mathsf{BV}(\mathcal{S})$ such that the rule $\texttt{RF}_X$ (resp. $\texttt{LF}_X$) is used infinitely often in γ';
- the maximal value of p_X for any such X is an even (resp. odd) number.

Guard condition. *Every infinite path in* Π *has a left μ-trace or a right ν-trace.*

The guard condition, in the view that proofs denote programs, is a productivity constraint. It says, intuitively, that after a while, the program either reads an inductive input or produces a coinductive output, while keeping track of what it is doing.

Definition 2.3. A *proof* is a pre-proof Π that satisfies the guard condition. It is a *circular proof* if Π is finite, and an *arboreal proof* if Π is a (possibly infinite) tree. A pre-proof is *ground* if it does not use the assumption rule, and *cut-free* if it does not use the cut rule.

For instance, a quick analysis of the cycles suffices to see that the pre-proof given in Figure 3 is a ground circular proof, while the one given in Figure 1 does not satisfy the guard condition. In [5], we proved the following result.

Theorem 2.4 (Soundness and Fullness)**.** *Let* Π *be a ground circular proof. Then* Π *is resolvable in any chosen* μ*-bicomplete category. In particular every* $v \in \Pi$ *denotes an arrow* $[\![v]\!]_\Pi$*. Conversely, every arrow of the free* μ*-bicomplete category is definable as the solution at some vertex of a ground circular proof.*

Since the category of sets is μ-bicomplete [12] and because of the richness of such categories, many functions are definable by circular proofs. We focus our attention on the case of functions of the form $\mathbf{1} \to [\![T]\!]$ that denote a choice of a particular Σ-tree.

Definition 2.5. A Σ-tree t is *circularly definable* if there is a ground (finite) circular proof Π over a system $\mathcal{S} \supseteq \mathcal{T}(\Sigma)$ and a vertex $v \in \Pi$ such that $[\![v]\!]_\Pi : \mathbf{1} \to [\![T]\!]$ has value t in **Set**. Let **CD** denote the set of circularly definable trees.

One reason for focusing on functions of the form $\mathbf{1} \to [\![T]\!]$ is that the analysis of cut-elimination procedure (see Section 2.3 below) is simpler on them, although they are general enough to encode wide classes of interesting functions. For instance, by results in [2, 10] combined with Theorem 2.4, the class of streams $f \in \mathbb{N}^\mathbb{N}$ definable by circular proofs includes all the primitive recursive functions in one variable. But streams can be encoded by their *comb-tree*, $\textsc{Comb}(f)$, defined in the usual way: the positions in the stream (or inputs) are encoded by an infinite branch whose nodes are labeled by a 2-ary symbol a, and from such a symbol a at depth n, we put a branch of $f(n)$ 1-ary symbols s (successor) followed by one 0-ary symbol z (zero). For instance, Figure 2 shows the comb-tree of the identity function.

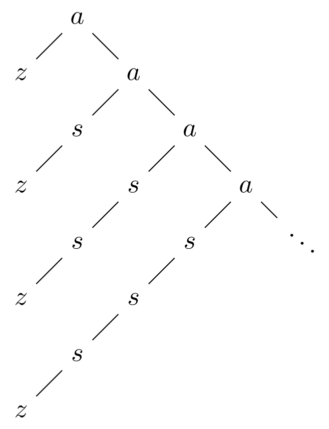

Figure 2: The comb-tree of identity

In Figure 3, we give a circular proof whose interpretation is the function $\textsc{Comb} : \mathbb{N}^\mathbb{N} \to [\![T]\!]$. An argument for claiming this can be achieved with the help of cut-elimination (see 2.3 below). Essentially, the left rules of the proof are "reading rules" that interact with similar right rules of a stream given input by an essential cut-reduction, while the right rules are "writing rules", on which cut-elimination is productive. In Figure 3, the underlying directed system is $\mathcal{S} \cup \mathcal{T}(\Sigma)$ (as in equations (1) and (2)) with Σ as above. We conclude that for all primitive recursive $f \in \mathbb{N}^\mathbb{N}$, $\textsc{Comb}(f) \in \mathbf{CD}$.

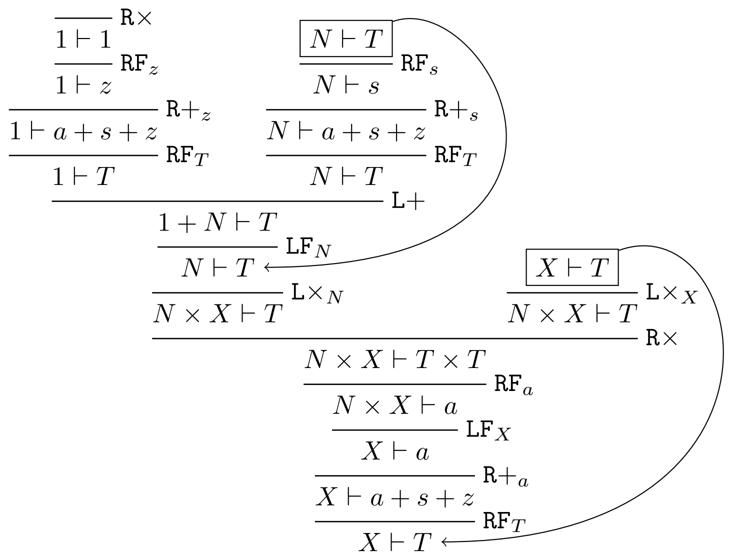

Figure 3: A circular proof for turning streams into comb trees

A cardinality argument shows, however, that many Σ-trees are not circularly definable. For those, we need arboreal proofs. Figure 4 shows how to translate a Σ-tree $t = (f, t_1 \ldots t_r)$ into its proof representation Ψ_t. Note that each Ψ_t is resolvable in **Set**.

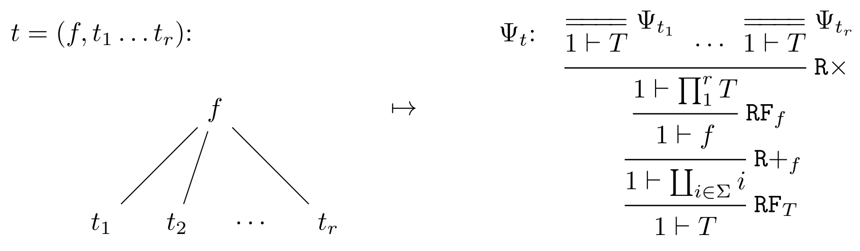

Figure 4: Proof representation of Σ-trees

In that Figure (and many others later in this paper), we allow ourselves to use a previously defined proof Π, with root of the form $a \vdash b$ and assumptions $\mathtt{A}_0 \ldots \mathtt{A}_k$ of the form $a_i \vdash b_i$, as if it was a new rule (or subroutine) denoted as follows:

$$\dfrac{a_0 \vdash b_0 \ \cdots\ a_k \vdash b_k}{a \vdash b}\,\Pi \qquad \text{or} \qquad \dfrac{\{a_i \vdash b_i\}_{i=1\ldots k}}{a \vdash b}\,\Pi$$

On the other hand, it should be clear that any ground and cut-free arboreal proof of the sequent $1 \vdash T$ is Ψ_t for some Σ-tree t, except for possible uses of `Id` instead of $\mathtt{R}\times$ to justify the sequent $1 \vdash 1$. This is just an instance of a more general fact.

Lemma 2.6. *Let Π_1, Π_2 be two ground, cut-free arboreal proofs with no occurrence of the rule* `Id`*. Let r_i be the root of Π_i, and suppose that $\textsc{Seq}(r_1) = \textsc{Seq}(r_2) = 1 \vdash \tau$ for some term τ. Assume also that Π_1 and Π_2 both are resolvable. Then, if $[\![r_1]\!]_{\Pi_1} = [\![r_2]\!]_{\Pi_2}$, we can conclude $\Pi_1 = \Pi_2$.*

Proof. By the principle of coinduction (see [6]), it is sufficient to show that the relation $\sim$ defined by

$$\Pi_1 \sim \Pi_2 \quad \Longleftrightarrow \quad \textsc{Seq}(r_1) = \textsc{Seq}(r_2) \text{ and } [\![r_1]\!] = [\![r_2]\!]$$

is a *bisimulation* on the set of proofs for which the left-hand side of the sequent at the root is 1. That is, we want to show that if $\Pi_1 \sim \Pi_2$, then the following is true:

- $\textsc{Rule}(r_1) = \textsc{Rule}(r_2)$;
- r_1 and r_2 have the same degree;
- Given i, j, let Π_i^j be the sub-proof of Π_i generated by the j-th son of r_i. Then for all j, $\Pi_1^j \sim \Pi_2^j$.

Verifying these properties is a simple case-check on the value of $\textsc{Rule}(r_1)$. Keep in mind that since $\textsc{Seq}_{\mathtt{L}}(r_1) = \textsc{Seq}_{\mathtt{L}}(r_2) = 1$ and the proofs do not contain occurrences of `Cut` or `Id`, then $\textsc{Rule}(r_1), \textsc{Rule}(r_2) \in \mathfrak{R}$. $\square$

2.3. Cut-elimination

In [5], we gave an algorithm for eliminating cuts in circular proofs, although their finiteness cannot always be preserved in the process. Given a ground (not necessarily finite) proof Π, cut-elimination on Π can be described as a sort of automaton whose configurations are what we call *multicuts*. A multicut is just a finite composable sequence $M = [u_1, u_2 \dots u_m]$ of vertices of Π. We can think of multicuts of size m as m-ary instances of a generalized cut rule `MCut` as follows, where $\textsc{Seq}(u_i) = a_{i-1} \vdash a_i$:

$$\frac{a_0 \vdash a_1 \quad a_1 \vdash a_2 \quad \dots \quad a_{m-1} \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut}.$$

The automaton, that we call *cut-eliminator*, builds up a cut-free proof by “pushing” cuts towards infinity. To do it, it needs to have $\textsc{Rule}(u_1) \in \mathfrak{L}$ or $\textsc{Rule}(u_m) \in \mathfrak{R}$ in order to operate commutative cut reductions, such as the following:

$$\frac{a_0 \vdash \dots \vdash a_{m-1} \qquad \dfrac{a_{m-1} \vdash F_X}{a_{m-1} \vdash X}\,\mathtt{RF}_X}{a_0 \vdash X}\,\mathtt{MCut} \quad \mapsto \quad \dfrac{\dfrac{a_0 \vdash \dots \vdash a_{m-1} \quad a_{m-1} \vdash F_X}{a_0 \vdash F_X}\,\mathtt{MCut}}{a_0 \vdash X}\,\mathtt{RF}_X .$$

Whenever this is not the case, the automaton can modify the content of M by performing one of the following internal operations:

- merge the hypothesis of a cut u_i into M:

$$\frac{a_0 \vdash \dots \qquad \dfrac{a_i \vdash c \quad c \vdash a_{i+1}}{a_i \vdash a_{i+1}}\,\mathtt{Cut} \qquad \dots \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut} \quad \mapsto \quad \frac{a_0 \vdash \dots\ a_i \vdash c \quad c \vdash a_{i+1}\ \dots \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut};$$

- remove an instance of the identity rule:

$$\frac{a_0 \vdash\ \dots\ \vdash b \qquad \dfrac{}{b \vdash b}\,\mathtt{Id} \qquad b \vdash\ \dots\ \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut} \quad \mapsto \quad \frac{a_0 \vdash\ \dots\ \vdash b \quad b \vdash\ \dots\ \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut};$$

- perform an essential cut reduction between two successive vertices u_i,u_{i+1} such that $\textsc{Rule}(u_i) \in \mathfrak{R}$ and $\textsc{Rule}(u_{i+1}) \in \mathfrak{L}$ (we illustrate it with the case of fixpoint rules, but a similar reduction applies to product and coproduct rules):

$$\frac{a_0 \vdash \dots \quad \dfrac{a_{i-1} \vdash F_X}{a_{i-1} \vdash X}\,\mathtt{RF}_X \quad \dfrac{F_X \vdash a_{i+1}}{X \vdash a_{i+1}}\,\mathtt{LF}_X \quad \dots \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut} \quad \mapsto \quad \frac{a_0 \vdash \dots a_{i-1} \vdash F_X \quad F_X \vdash a_{i+1} \dots \vdash a_m}{a_0 \vdash a_m}\,\mathtt{MCut}.$$

If a multicut M' is obtained from M by one of these internal operations, we write $M \rightarrowtail M'$ (or $M \overset{*}{\rightarrowtail} M'$ if an arbitrary number of steps are necessary).

The problem with internal operations is that they do not allow the cut-eliminator to produce a part of an infinite proof tree by “pushing” the cut away: they rather “pull” everything above to the ground. Since Π may contain infinite paths, it may not seem clear that this process eventually halts. This is one of the main results in [5], that we could rephrase as: *there is no infinite $\rightarrowtail$-chain*, given that Π satisfies the guard condition. Thus, the guard condition is sufficient for ensuring that, given a multicut M over Π, cut-elimination produces an arboreal cut-free proof $\textsc{ce}_\Pi(M)$. This proof, in turn, is resolvable: the solution of the corresponding system at $v \in \textsc{ce}_\Pi(M)$ is obtained by composition of the solutions (in Π) of the vertices in the multicut at the moment when v is produced.

Definition 2.7. A Σ-tree t is *circularly computable* if there exists a multicut M over a ground and finite pre-proof Π, such that $\textsc{ce}_\Pi(M) = \Psi_t$. Let $\mathbf{CC}$ denote the set of circularly computable trees.

Note that the definition of $\textsc{ce}_\Pi(M)$ could be ambiguous, since the cut-eliminator is not deterministic: it chooses between producing something on the left or on the right, given that both options are available. However, in our case, since the sequent at the root of Ψ_t is $1 \vdash T$ and no left rule can justify a sequent of that form, then the cut-eliminator always justifies all its productions by right rules.

Proposition 2.8. $\mathbf{CD} \subseteq \mathbf{CC}$.

Proof. Let $t \in \mathbf{CD}$, so that there is a ground circular proof Π and $v \in \Pi$ such that $[\![v]\!]_\Pi : \mathbf{1} \to [\![T]\!]$ has value t.

Note that the only way, for the cut-eliminator, to justify a node by the rule `Id` is to reach a state where its multicut is $[u]$, for some $u \in \Pi$ such that $\textsc{Rule}(u) = \texttt{Id}$. But since the left-hand side of the production must be 1, then $\textsc{Seq}(u) = 1 \vdash 1$. Hence, by replacing Π by a proof Π' in which every sequent $1 \vdash 1$ is justified by $\texttt{R}\times$, we can make sure that $\textsc{ce}_{\Pi'}([v])$ does not involve the identity rule. This operation does not change the system of equations since $[\![1]\!]$ is the final object. Hence, $[\![v]\!]_{\Pi'}$ has value t.

But $\textsc{ce}_{\Pi'}([v])$ is resolvable and the solution at the root is $[\![v]\!]_{\Pi'}$. Hence, by Lemma 2.6, $\textsc{ce}_{\Pi'}([v]) = \Psi_t$ and therefore $t \in \mathbf{CC}$. □

Note that it is not clear whether Proposition 2.8 is true or not, since there are some finite pre-proofs on which cut-elimination is productive, even if they do not satisfy the Guard condition. Indeed, the construction that we are about to describe, of a pre-proof that simulates the behavior of a given higher-order pushdown automaton, provides such examples.

3. Higher-Order Pushdown Automata

We use the model of higher-order pushdown automata from [7]. Such an automaton manipulates a higher-order stack in a way that is completely determined by its top symbol. Hence, they are not accepting or rejecting words given as input, but rather accepting some Σ-tree. Our goal is to describe a simulation of higher-order pushdown automata by cut-elimination, in order to conclude that the accepted Σ-trees are circularly computable.

3.1. The Algebra of Stacks

Definition 3.1. Fix a finite alphabet Γ. A 0-stack is an element of Γ. For $n \geq 1$, an n-stack is a finite list of $(n-1)$-stacks. A n-stack is *well-formed* if $n = 0$ or if it is a *nonempty* list of well-formed $(n-1)$-stacks. The *top symbol* of a well-formed n-stack is defined by $\texttt{top}(a) = a$ if $a \in \Gamma$, $\texttt{top}[s_\ell, s_{\ell-1} \dots s_1] = \texttt{top}(s_\ell)$ otherwise.

This is essentially the definition from [7], although in the latter, n-stacks (also called *higher-order pushdown stores*) are always well-formed. This will make no difference in our modeling of automata, since the operations we will define on them are partial functions, the non-well-formed case being treated as an error. In [7], we also assume that Γ contains a special symbol $\perp$ to denote the bottom of the stack. We will also use this convention later in our construction, although it is not necessary to define the n-stacks algebraically.

First, we need to describe n-stacks as directed systems of equations, in order to use them in circular proofs. Since n-stacks are lists of $(n-1)$-stacks, the reader who is familiar with initial algebras might suggest to iterate the Kleene star $S^* = \mu X.(1 + S \times X)$. But this will not work since cartesian products behave inaccurately on the left side of circular proofs. Indeed, the structure of the rules $\mathtt{L}\times_i$ only allows us to read the head or the tail of such a list, destroying the rest of the data in the process.

We circumvent this problem by taking the free monoid in the category $\mathcal{V} = \mathrm{End}(\mathbf{Set})$ of endofunctors of **Set**, rather than doing it in **Set** itself. Recall from [9, Chapter VII] that $\mathcal{V}$ is a strict monoidal category where the tensor is given by functorial composition $\circ$. We distinguish functorial composition from composition of ordinary functions, that we write in prefix notation: $(f \cdot g)(x) = g(f(x))$. A monoid in $\mathcal{V}$ is called a *monad.*

For any definable functor $F \in \mathcal{V}$, one can show (see [2]) that the functor defined by $\widehat{F}(X) := \mu Y.(X + F(Y))$ has the structure of a free monad over F. Therefore, we define the following family of functors.

$$S_0(X) := \coprod_{a \in \Gamma} X \qquad , \qquad S_n(X) := \widehat{S}_{n-1}(X)$$

So for $n \geq 1$, $S_n(X)$ is an initial algebra, whose structure map is

$$\alpha^n_X = \{\mathtt{nil}^n_X, \mathtt{cons}^n_X\} \ : \ X + S_{n-1}S_nX \longrightarrow S_nX.$$

For any symbol $a \in \Gamma$, let $\mathtt{in}^a_X : X \to S_0(X)$ denote the canonical injection.

Lemma 3.2. *For all $n \in \mathbb{N}$ and $X \in$ **Set**, S_nX is isomorphic to the set of pairs (s, x) such that s is an n-stack and $x \in X$. In particular, S_n1 is isomorphic to the set of n-stacks.*

Proof. We proceed by induction on n. For $n = 0$, this is just a consequence the usual definition of the coproduct in **Set**:

$$S_0X = \coprod_{a \in \Gamma} X = \big\{(a, x) : a \in \Gamma, x \in X\big\}.$$

For $n \geq 1$, we define a function $\varphi^n_X : S_nX \to W \times X$, where W is the set of finite words over the following alphabet:

$$A_n = \Gamma \cup \{\mathtt{cons}_k \ , \ \mathtt{nil}_k\}_{k=1\dots n}.$$

Let $z \in S_nX$. If $z = \mathtt{nil}^n_X(x)$ for some $x \in X$, let $\varphi_n(z) = (\mathtt{nil}_n, x)$. Otherwise, $z = \mathtt{cons}^n_X(y)$ for some $y \in S_{n-1}S_nX$. By induction, let $(w, z') := \varphi^{n-1}_{S_nX}(y)$. Note that $z' \in S_nX$ and w is a word over $A_{n-1} \subset A_n$. We define $\varphi^n_X(z) := \mathtt{cons}_n \cdot w \cdot \varphi^n_X(z')$. That may look like a circular definition (and it is!), but by initiality of the S_k's, the computation of $\varphi^n_X(z)$ must halt, the base case being $z = \mathtt{nil}^n_X(x)$ for some $x \in X$.

So we have $\varphi^n_X = (s, x)$ for some $s \in W$ and $x \in X$. Note that s is independent of x, since x is only used in the base case. It follows that φ^n_X is injective. Its first component is, therefore, an isomorphism into its image set $L_n \subseteq W$. By construction, the L_k's must satisfy the following equations:

$$\begin{aligned} L_0 &= \Gamma \\ L_k &= (\mathtt{cons}_k \cdot L_{k-1})^* \cdot \mathtt{nil}_k \,. \end{aligned}$$

Hence, $L_n \cong L^*_{n-1}$ and, by induction, L_{n-1} is isomorphic to the set E of $(n-1)$-stacks. So we have $L_n \cong E^*$ and, by Definition 3.1, E^* is the set of n-stacks. ☐

A visual interpretation of the proof of Lemma 3.2 is achievable through an interpretation of directed systems of equations as parity games, as described in [12]. Indeed, let $Z_k = (S_k \circ \ldots \circ S_n)(X)$ for $0 \leq k \leq n$. Then by construction of the S_k's, the Z_k's must satisfy the following directed system of equations.

$$\mathcal{Z} : \begin{cases} Z_{n+1} = \ X \\ \quad Z_k =_1 Z_{k+1} + Z_{k-1} \qquad (0 \leq k \leq n) \\ \quad Z_0 =_1 \coprod_{a \in \Gamma} Z_1 \end{cases}$$

Following the conversion proposed in [12], we can rewrite this system as a graph with one vertex for each variable of $\mathcal{Z}$ and an edge $X \to Y$ if and only if Y occurs in F_X (see Figure 5).

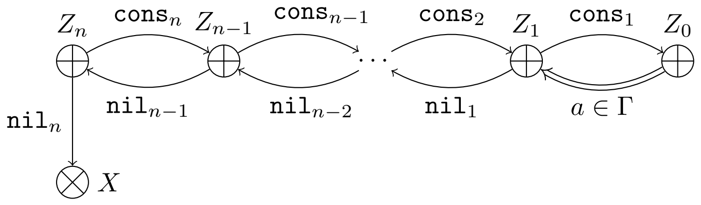

Figure 5: Parity game board associated to S_nX

Still following the interpretation in [12] (and skipping the details), S_nX is then isomorphic to the set of pairs (s, x) such that s is a path from Z_n to X in the graph, and x is a choice of element in X. The set of those paths s is exactly the set L_n in the proof of Lemma 3.2. It is just easier to memorize it with the help of the graph than it is with only the formula defining L_n.

Now that we are given an algebraic description of n-stacks, we can express some basic operations in this setting. The reason for doing this is to observe that all these definitions use nothing but the μ-bicomplete structure of **Set**, so by Theorem 2.4, they are all definable by circular proofs.

We start with the *bottom n-stacks*. Assume that there is a special symbol $\perp \in \Gamma$. We define $\perp_0 = \perp \in \Gamma$ and $\perp_{n+1} = [\perp_n]$. The $\perp_n$'s are particular stacks that we can assimilate to functions $\perp_n : 1 \to S_n(1)$. It suffices to take $\perp_n = b^n_1$ where $b^n_X : X \to S_nX$ is defined by $b^0_X = \mathtt{in}^{\perp}_X$, and $b^{n+1}_X = \mathtt{nil}^n_X \cdot b^{n-1}_{S_nX} \cdot \mathtt{cons}^n_X$.

Another operation is to push a given symbol $a \in \Gamma$ on top of a 1-stack. We call this the *symbol push* operation. In practice, it is used only with $a \neq \perp$.

$$\mathtt{spush}^a_1[a_\ell, a_{\ell-1} \ldots a_1] = [a, a_\ell, a_{\ell-1} \ldots a_1].$$

This is just $p^a_{1;1}$ where $p^a_{1;X}$ is defined by the following composition.

$$S_1X \xrightarrow{\mathtt{in}^a_{S_1X}} \coprod_{a \in \Gamma} S_1X = S_0S_1X \xrightarrow{\mathtt{cons}^1_X} S_1X \hookrightarrow X + S_1X.$$

Here, the extra summand X is added to the codomain, as it will be for the other operations, in order to treat the possible error case of non-well-formed stacks (there is no such possibility for $\mathtt{spush}^a_1$). To define the next operations on n-stacks, we need another property of the functors S_n: they are *cocommutative comonads*.

Lemma 3.3. *For each set X, there are functions* definable by circular proofs $\Upsilon^n_X : S_nX \to X$ *and* $\Delta^n_X : S_nX \to S_nS_nX$ *(called* destroy *and* double*) such that for all $(s, x) \in S_nX$, $\Upsilon^n_X(s, x) = x$ and $\Delta^n_X(s, x) = (s, (s, x))$.*

Proof. The definition of Υ^n_X and Δ^n_X, along with the fact that they provide a cocommutative comonadic structure (whence the equations provided in the statement of the Lemma), can be found (in a more general context) in [2]. One can check that this construction uses only the μ-bicompleteness of **Set**. Therefore, it follows by Fullness (Theorem 2.4) that Υ^n_X and Δ^n_X are definable by circular proofs. □

On any stack level, one can perform push and pop operations, where the term *push* refers to a full copy of the leftmost $(n-1)$-stack and *pop* refers to the destruction of that $(n-1)$-stack. Formally:

$$\begin{aligned} \mathtt{push}^n_n[s_\ell, s_{\ell-1}\dots s_1] &= [s_\ell, s_\ell, s_{\ell-1}\dots s_1];\\ \mathtt{pop}^n_n[s_\ell, s_{\ell-1}\dots s_1] &= [s_{\ell-1}\dots s_1].\end{aligned}$$

Each of those two operations is a function of the form $p^n_{n;1}$, where the family of functions $p^n_{n;X} : S_nX \to X + S_nX$ is of the form $p^n_{n;X} = (\alpha^n_X)^{-1} \cdot (\mathtt{id} + f)$. The difference between $\mathtt{push}^n_n$ and $\mathtt{pop}^n_n$ is the choice of function $f : S_{n-1}S_nX \to S_nX$ in that expression. For $\mathtt{pop}^n_n$, since we want to *destroy* some information, we take $f = \Upsilon^{n-1}_{S_nX}$. For $\mathtt{push}^n_n$, we want to *double* the first $(n-1)$-stack, so we let f be the following composition:

$$S_{n-1}S_nX \xrightarrow{\Delta^{n-1}_{S_nX}} S_{n-1}S_{n-1}S_nX \xrightarrow{S_{n-1}(\mathtt{cons}^n_X)} S_{n-1}S_nX \xrightarrow{\mathtt{cons}^n_X} S_nX.$$

We also need the following level $k < n$ operations on n-stacks:

$$\begin{aligned} \mathtt{spush}^a_n[s_\ell, s_{\ell-1}\dots s_1] &= [\mathtt{spush}^a_{n-1}(s_\ell), s_{\ell-1}\dots s_1];\\ \mathtt{push}^k_n[s_\ell, s_{\ell-1}\dots s_1] &= [\mathtt{push}^k_{n-1}(s_\ell), s_{\ell-1}\dots s_1];\\ \mathtt{pop}^k_n[s_\ell, s_{\ell-1}\dots s_1] &= [\mathtt{pop}^k_{n-1}(s_\ell), s_{\ell-1}\dots s_1].\end{aligned}$$

The shape of these three definitions is the same. The functions we are defining are of the form $p^z_{n;1}$, where $p^z_{n;X} : S_nX \to X + S_nX$ is either already defined above, or can be reached by induction on n with $p^z_{n;X} = (\alpha^n_X)^{-1} \cdot (\mathtt{id} + f)$, where f is the following composition

$$S_{n-1}S_nX \xrightarrow{p^z_{n-1;S_nX}} S_nX + S_{n-1}S_nX \xrightarrow{\Upsilon^n_X + \mathtt{id}} X + S_{n-1}S_nX \xrightarrow{\alpha^n_X} S_nX.$$

The set of *level n operations* is then defined as

$$\mathcal{O}_n = \{\mathtt{spush}^a_n : a \in \Gamma \setminus \{\bot\}\} \cup \{\mathtt{push}^k_n, \mathtt{pop}^k_n : 1 \le k \le n\}.$$

3.2. From automata to proofs

Definition 3.4. A *level n pushdown automaton* (or n-PDA for short) is a tuple $\mathcal{A} = \langle Q, \Sigma, \Gamma, q_0, \delta\rangle$, where Q is a finite set of states with an initial state $q_0 \in Q$, Σ is a signature, Γ is a finite stack alphabet and $\delta : Q \times \Gamma \to \mathcal{I}_\mathcal{A}$ is the *transition function.* The codomain $\mathcal{I}_\mathcal{A}$ of δ is the set of *admissible instructions*, consisting of the expressions of one of the two following forms: (q, φ), where $q \in Q$ and $\varphi \in \mathcal{O}_n$; or $(f, p_1 \dots p_r)$, where $f \in \Sigma$, $r = \mathrm{ar}(f)$ and $p_1 \dots p_r \in Q$.

A *configuration* of $\mathcal{A}$ is a pair (q, s) where $q \in Q$ and s is an n-stack. Let $\mathcal{C}_\mathcal{A}$ be the set of configurations of $\mathcal{A}$. We write $(q, s) \to_\mathcal{A} (q', s')$ if $\delta(q, \mathtt{top}(s)) = (q', \varphi)$ for some $\varphi \in \mathcal{O}_n$ such that $s' = \varphi(s)$. The relation $\twoheadrightarrow_\mathcal{A}$ is the reflexive transitive closure of $\to_\mathcal{A}$. The *initial configuration* of $\mathcal{A}$ is $(q_0, \bot_n)$.

Let $t = (f, t_1 \dots t_r)$ be a Σ-tree and $(q, s) \in \mathcal{C}_\mathcal{A}$. A *run of t from* (q, s) is a partial function $\varrho : T \rightharpoonup \mathcal{C}_\mathcal{A}$ defined at t, with the following property. Let $\varrho(t) = (q_t, s_t)$. Then $(q, s) \twoheadrightarrow_\mathcal{A} (q_t, s_t)$ and $\delta(q_t, \mathtt{top}(s_t)) = (f, p_1 \dots p_r)$ for some states $p_1 \dots p_r \in Q$ such that for $1 \le i \le r$, ϱ is a run of t_i from

(p_i, s_t). A tree t is *accepted* by $\mathcal{A}$ if and only if there is a run of t from $(q_0, \bot_n)$. Note that, since our automata are deterministic, any n-PDA accepts at most one tree.

Next we now show how to convert an n-PDA $\mathcal{A} = \langle Q, \Sigma, \Gamma, q_0, \delta\rangle$ into a finite pre-proof $\Pi(\mathcal{A})$. The general idea is to reproduce the graph structure of $\mathcal{A}$, while replacing each vertex by a *gadget* that simulates it. For this, we keep the stack on the left of the turnstile symbol, and the tree t accepted by $\mathcal{A}$ on the right. In order to deal with possible errors cases, we see t as a Σ'-tree, where $\Sigma' = 1 + \Sigma$. That allows us to define a proof $\mathtt{ERR}_X$ for any term X in the following way.

$$\mathtt{ERR}_X = \cfrac{\cfrac{\cfrac{}{X \vdash 1}\,\mathtt{R}\times}{X \vdash 1 + \coprod_{f\in\Sigma} f}\,\mathtt{R}+_0}{X \vdash T}\,\mathtt{RF}_T$$

First we need to encode the transition function δ. The encoding depends on the shape of $\delta(q, a)$ and is shown in Table 2.

$\delta(q,a) = (\varphi, p)$ for $\varphi \in \mathcal{O}_n$	$\delta(q,a) = (f, p_1 \dots p_r)$ for $f \in \Sigma$
$\cfrac{\cfrac{}{S_n1 \vdash 1 + S_n1}\,\varphi \quad \cfrac{\cfrac{}{1 \vdash T}\,\mathtt{ERR}_1 \quad \cfrac{}{S_n1 \vdash T}\,\mathtt{A}}{1 + S_n1 \vdash T}\,\mathtt{L}+}{S_n1 \vdash T}\,\mathtt{Cut}$	$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{S_n1 \vdash T}\,\mathtt{A}_1 \quad \cdots \quad \cfrac{}{S_n1 \vdash T}\,\mathtt{A}_r}{S_n1 \vdash \prod_1^r T}\,\mathtt{R}\times}{S_n1 \vdash f}\,\mathtt{RF}_f}{S_n1 \vdash 1 + \coprod_{i\in\Sigma} i}\,\mathtt{R}+_f}{S_n1 \vdash T}\,\mathtt{RF}_T$

Table 2: Proof encoding of $\delta(q, a)$

In $\mathcal{A}$, the choice of transition to take depends on the top symbol of the n-stack. Hence, we want a proof $\mathtt{TOP}$ with as many assumptions as the cardinality of Γ, that we can use to branch on different cases.

The construction of $\mathtt{TOP}$ is in two steps. First, we need to "dig" in the n-stack in order to access the top symbol. We show how in Table 3, where a new piece of notation is introduced: given a proof Π in which the variable X occurs free (i.e. no fixpoint rule is applied at X) and some term τ, $\Pi[X/\tau]$ is the proof in which τ is substituted for every occurence of X. Since reading is destructive in circular proofs, we also need a proof to "fill" such an input by putting the lost symbol back and building a well-formed stack (see Table 3 again).

We can now define $\mathtt{TOP}$ as follows:

$$\mathtt{TOP} = \cfrac{\left\{ \cfrac{\cfrac{}{(S_1 \cdots S_n)1 \vdash S_n1}\,\mathtt{FILL}_n^a[X := 1] \quad \cfrac{}{S_n1 \vdash T}\,\mathtt{A}_a}{(S_1 \cdots S_n)1 \vdash T}\,\mathtt{Cut} \right\}_{a\in\Gamma}}{S_n1 \vdash T}\,\mathtt{DIG}_n[X := 1]$$

For all $q \in Q$, generate the following proof Π_q. Let $\surd_q$ be the root of Π_q and $@_q^{a,i}$ the assumption node labeled by $\mathtt{A}_{a,i}$.

$$\Pi_q = \cfrac{\left\{ \cfrac{\cfrac{}{S_n1 \vdash T}\,\mathtt{A}_{a,1} \quad \dots \quad \cfrac{}{S_n1 \vdash T}\,\mathtt{A}_{a,r}}{S_n1 \vdash T}\,\delta(q,a) \right\}_{a\in\Gamma}}{S_n1 \vdash T}\,\mathtt{TOP}$$

$\mathtt{DIG}_0$: $\dfrac{\dfrac{\left\{ \dfrac{}{X \vdash T}\,\mathtt{A}_a \right\}_{a\in\Gamma}}{\coprod_\Gamma X \vdash T}\,\mathtt{L+}}{S_0 X \vdash T}\,\mathtt{LF}_{S_0X}$	$\mathtt{DIG}_k$: $\dfrac{\dfrac{\dfrac{}{X \vdash T}\,\mathtt{ERR}_X \qquad \dfrac{\left\{ \dfrac{}{(S_1\cdots S_k)X \vdash T}\,\mathtt{A}_a \right\}_{a\in\Gamma}}{S_{k-1}S_kX \vdash T}\,\mathtt{DIG}_{k-1}[X := S_kX]}{X + S_{k-1}S_kX \vdash T}\,\mathtt{L+}}{S_kX \vdash T}\,\mathtt{LF}_{S_kX}$
$\mathtt{FILL}^a_0$: $\dfrac{\dfrac{\dfrac{}{X \vdash X}\,\mathtt{Id}}{X \vdash \coprod_\Gamma X}\,\mathtt{R+}_a}{X \vdash S_0X}\,\mathtt{RF}_{S_0X}$	$\mathtt{FILL}^a_k$: $\dfrac{\dfrac{\dfrac{}{(S_1\cdots S_k)X \vdash S_{k-1}S_kX}\,\mathtt{FILL}^a_{k-1}[X := S_kX]}{(S_1\cdots S_k)X \vdash X + S_{k-1}S_kX}\,\mathtt{R+}_1}{(S_1\cdots S_k)X \vdash S_kX}\,\mathtt{RF}_{S_kX}$

Table 3: Categorical programming at work

Let also $\Pi_\perp$ be the following proof. Call its root $\surd_\perp$ and its assumption $\oslash_\perp$.

$$\Pi_\perp = \dfrac{\dfrac{}{1 \vdash S_n 1}\,\perp_n \qquad \dfrac{}{S_n 1 \vdash T}\,\mathtt{A}}{1 \vdash T}\,\mathtt{Cut}$$

The pre-proof $\Pi(\mathcal{A})$ is then obtained from the union of $\Pi_\perp$ with all the Π_q by linking $\oslash_\perp$ to $\surd_{q_0}$ rather than by justifying with an assumption and, for all $q \in Q$ and $a \in \Gamma$: if $\delta(q,a) = (p,\varphi)$, link $\oslash^{a,1}_q$ to $\surd_p$ in a similar fashion; otherwise, $\delta(q,a) = (f, p_1 \dots p_r)$ and then, for $i = 1 \dots r$, link $\oslash^{a,i}_q$ to $\surd_{p_i}$.

We can now state this paper's main theorem: trees accepted by n-PDAs are circularly computable.

Theorem 3.5. *Let $\mathcal{A}$ be an n-PDA that accepts $t \in T$. Then* $\textsc{ce}_{\Pi(\mathcal{A})}([\surd_\perp]) = \Psi_t$.

Proof sketch. The key of the proof is to lift the local behavior of a run $\varrho : T \rightharpoonup \mathcal{C}_\mathcal{A}$ to the set of configurations of the cut-eliminator. These configurations simply are multicuts on $\Pi(\mathcal{A})$.

We say that a multicut $M = [u_1 \dots u_{m-1}, u_m]$ is *good* if the left-hand side of $\textsc{Seq}(u_1)$ is 1 and $u_m = \surd_q$ for some $q \in Q$. Note that any good multicut is then associated to a state q_M, and for $1 \leq i < m$, u_i is part of a *valid* sub-proof of $\Pi(\mathcal{A})$. Since the right-hand side of $\textsc{Seq}(u_{m-1})$ is $S_n(1)$, then M defines a unique stack $s_M = [\![u_1]\!] \cdots [\![u_{m-1}]\!]$. So if $\mathcal{G}_\mathcal{A}$ is the set of good multicuts, we can define a function $\psi : \mathcal{G}_\mathcal{A} \to \mathcal{C}_\mathcal{A}$ by $\psi(M) = (q_M, s_M)$.

Let $M \in \mathcal{G}_\mathcal{A}$ and $(q,s) = \psi(M)$, such that s is well-formed. A technical analysis of the construction of Π_q leads to the following facts:

1. For all $(q', s') \in \mathcal{C}_\mathcal{A}$ such that $(q,s) \to_\mathcal{A} (q',s')$, there is $M' \in \mathcal{G}_\mathcal{A}$ such that $M \mathrel{\overline{✂}} M'$ and $\psi(M') = (q', s')$;

2. If $\delta(q, \mathtt{top}(s)) = (f, p_1 \dots p_r)$, then there are $M'_1 \dots M'_r \in \mathcal{G}_\mathcal{A}$ such that for all i, $\psi(M'_i) = (p_i, s)$. Also, $\textsc{ce}_{\Pi(\mathcal{A})}(M)$ has the same shape as Ψ_t (see Figure 4), except that Ψ_{t_i} on the Figure is replaced by $\textsc{ce}_{\Pi(\mathcal{A})}(M'_i)$.

Now, let $t = (f, t_1 \dots t_r) \in [\![T]\!]$ and $(q,s) \in \mathcal{C}_\mathcal{A}$ be such that there is a run ϱ of t from (q,s). Suppose there is a multicut $M \in \mathcal{G}_\mathcal{A}$ such that $\psi(M) = (q,s)$. Using facts 1 and 2 above, a rather simple induction shows that $\textsc{ce}_{\Pi(\mathcal{A})}(M) = \Psi_t$. In particular, if $M = [u, \surd_{q_0}]$ such that $[\![u]\!] = \perp_n$, then t is the tree accepted by $\mathcal{A}$. Since $[\surd_\perp] ✂ M$, we conclude that $\textsc{ce}_{\Pi(\mathcal{A})}([\surd_\perp]) = \textsc{ce}_{\Pi(\mathcal{A})}(M) = \Psi_t$. $\square$

Note that the converse of Theorem 3.5, which states that circularly computable trees are accepted by some n-PDA, is not true. For instance, let $f : \mathbb{N} \to \mathbb{N}$ be the function assigning to $x \in \mathbb{N}$ a tower of x exponents: that is, let $h_0(x) = x$, $h_{k+1}(x) = 2^{h_k(x)}$, and define $f(x) = h_x(1)$. Then f is primitive recursive and therefore, by the discussion in Section 2.2, $\textsc{Comb}(f) \in \mathbf{CD} \subseteq \mathbf{CC}$. However, if $\textsc{Comb}(f)$ were accepted by some n-PDA, then as a consequence of [3, Th. 9.3], there would exist a polynomial p such that for x large enough, $f(x) \leq (h_{2n} \circ p)(x)$, a contradiction.

4. Conclusion

The work done in this paper provides effective links between language theory and the theory of initial algebras and final coalgebras. Indeed, our constructive encoding of n-PDAs via finite pre-proofs provides an algebraic description of higher-order trees as solutions (in the category of sets) to a certain class of systems of equations with initial algebras and final coalgebras, as described in Section 2 and, in more details, in [5]. On the other hand, the cut-eliminator is an alternative abstract machine for computing them.

It is important to note, however, that in general (except for the case $n = 0$), there is no cause for believing that $\Pi(\mathcal{A})$ satisfies the Guard condition (hence that higher-order trees are in $\mathbf{CD}$). The Guard condition on $\Pi(\mathcal{A})$ is equivalent to the assertion that every cycle in $\mathcal{A}$ goes through an instruction of the form $(f, p_1 \dots p_r)$. Of course, we did not use this in the proof of Theorem 3.5: productivity of cut-elimination was ensured by the fact that $\mathcal{A}$ accepts a tree.

So either our construction of $\Pi(\mathcal{A})$ is not smart enough, or there is a difference between $\mathbf{CC}$ and $\mathbf{CD}$ that we have yet to understand. That leaves open the question of measuring the expressive power of these two classes. For instance, can one encode higher-order pushdown trees into *valid* circular proofs? If $\mathbf{CD} \neq \mathbf{CC}$ and μ-bicomplete categories are the semantical world for $\mathbf{CD}$, then what is the semantical world of $\mathbf{CC}$? Can we formulate a (weaker) guard condition that characterize the pre-proofs on which cut-elimination is productive?

Acknowledgements. The author would like to thank Luigi Santocanale for proposing the problem and for his constant support and helpful discussions towards solving it. Thanks also for the referees of a previous version of this paper for their helpful remarks on how to improve it.

Bibliographie

[1] Caucal D. : On infinite transition graphs having a decidable monadic theory. In: Proc. 23rd International Colloquium on Automata, Languages and Programming, LNCS, vol. 1099, pp. 194–205. Springer, Heidelberg (1996)

[2] Cockett J. R. B., Santocanale, L. : Induction, Coinduction and Adjoints. Electr. Notes Theor. Comput. Sci. 69, 101–119 (2003)

[3] Damm W. : The IO and OI hierarchies. Theoretical Computer Science, vol. 20, pp. 95–207. (1982)

[4] Fortier, J. : Puissance expressive des preuves circulaires. Ph.D. Thesis, Université du Québec à Montréal and Aix-Marseille Université. (2014)

[5] Fortier, J., Santocanale, L. : Cuts for Circular Proofs: Semantics and Cut-Elimination. In: Ronchi Della Rocca, S. (ed.) Computer Science Logic 2013 (CSL'13). LIPIcs, vol. 23, pp. 248–262. Dagstuhl (2013)

[6] Jacobs, B., Rutten, J. : A Tutorial on (Co)Algebras and (Co)Induction. EATCS Bulletin, vol. 62, pp. 62–222. (1997)

[7] Knapik, T., Niwiński, D., Urzyczyn, P. : Higher-Order Pushdown Trees Are Easy. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002, LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)

[8] Lambek, J., Scott, P. J. : Introduction to higher order categorical logic. Cambridge studies in advanced mathematics 7. (1986)

[9] MacLane, S. : Categories for the Working Mathematician (2nd ed.). GTM vol. 5, Springer–Verlag, New York (1998)

[10] Paré, R., Román, L. : Monoidal Categories With Natural Numbers Object. Studia Logica, vol. 48, Issue 3, pp. 361–376 (1989)

[11] Santocanale, L. : A Calculus of Circular Proofs and its Categorical Semantics. In: Nielsen, M., Engberg, U. (eds.) FoSSaCS 2002, LNCS, vol. 2303, pp. 357–371. Springer, Heidelberg (2002)

[12] Santocanale, L. : μ-Bicomplete Categories and Parity Games. RAIRO – Theoretical Informatics and Applications, vol. 36.2, pp. 195–227. (2002)