

Delegation Networks

Frank Drewes

Department of Computing Science, Umeå University
S-901 87 Umeå, Sweden
drewes@cs.umu.se

Abstract. We introduce a generalization of tree-based generators called delegation networks. These make it possible to generate objects such as strings, trees, graphs, and pictures in a modular way by combining tree-based generators of several types. Our main result states that, if all underlying tree generators generate regular tree languages (or finite tree languages), then the tree-generating power of delegation networks is the same as that of context-free tree grammars working in IO mode.

Copyright © 2007

UMINF 07.04

ISSN 0348-0542

1 Introduction

The generation of objects such as strings, trees, graphs, and pictures is an important concept in computer science that has many facets. A huge variety of generating mechanisms have been developed in different areas, for different purposes, and with different characteristics. An important class are rule-based devices, and in particular various sorts of grammars. Several well-known types of such grammars can be formulated as tree-based generators [Dre06], where the syntactic aspects of the generation process are taken care of by a tree generator, i.e., some kind of device generating trees. At the semantic level, the generated trees are viewed as expressions and evaluated by interpreting every symbol as an operation on the domain of interest. The operations that can be used are taken from a (possibly infinite) set of predefined operations. The properties of a class of tree-based devices are thus determined by the sort of tree generator used and the class of predefined operations available.

In this paper, we propose delegation networks as a means to combine several generating devices. A delegation network \mathcal{N} is a finite set of so-called delegating generators. A delegating generator is basically a tree-based generator, but it may delegate the generation of certain parts of the object to another generator in \mathcal{N} . This extends tree-basedness in several ways:

1. Delegation provides a means to modularize the generation of complex objects in a meaningful way.

2. The individual devices that a delegating generator consists of can make use of different classes of tree generators.
3. Similarly, the individual devices may be based on different sets of predefined operations and work on different domains.
4. Finally, part of the generation task can make use of devices that are not tree based at all, by viewing them as predefined operations.

Although picture generation is not the main focus of this paper, it should be pointed out that the main motivation for studying delegation networks is provided by applications in this field. While various interesting rule-based generation mechanisms for pictures have been proposed and studied in theoretical computer science (among them the tree-based ones), their usefulness for practical purposes has been rather limited until now. An important reason for this seems to be the lack of modularity and the fact that it is unclear how different types of picture generators can be combined. In fact, even if we only want to generate a picture of, say, a single plant, the problems become apparent. The branching structure of the plant is rather different in nature from the form of its leaves and blossoms. In addition, the leaves may have a texture which we may wish to generate as well. Another texture may be required for the bark. Thus, while modularity is one aspect, the more important one seems to be that entirely different generation mechanisms appear to be needed. For example, the branching structure of the plant may conveniently be described by the turtle mechanism (see [PL90, Dre06]), whereas collage grammars or iterated function systems could be used to generate blossoms, and yet another mechanism could be employed for the generation of the required textures. However, it is clearly inappropriate to let these components work as stand-alone devices as their results would finally have to be assembled in an ad hoc way to create the desired structure. In fact, since a plant may have many leaves and blossoms, we need a potentially infinite number of instances of the corresponding device. Moreover, the creation of these instances should be triggered by the one that generates the branching structure. At the same time, the latter should determine the placement of the generated leaves and blossoms.

The way in which delegation networks make such a combination of devices possible is conceptually easy. As its major component, a delegating generator internally contains a tree generator each of whose symbols is either a so-called primitive or refers to another delegating generator of the network. For this purpose, the symbols are taken from a many-sorted signature. Thus, every symbol F has a so-called profile $\mathbf{A}_1 \times \dots \times \mathbf{A}_k \rightarrow \mathbf{A}$ determining the domains of its arguments and of the return value. A delegating generator or predefined operation associated with F takes arguments from $\mathbb{A}_1, \dots, \mathbb{A}_k$, the domains corresponding to the formal sorts $\mathbf{A}_1, \dots, \mathbf{A}_k$, and returns results belonging to \mathbb{A} . Usually, this is done in a nondeterministic manner, which means that different possible results may be given for one and the same argument tuple. In particular, the special case $k = 0$ results in a language, i.e., a subset of the domain \mathbb{A} . As primitives can be interpreted arbitrarily, this is where generating mechanisms that are not tree based can enter the picture.

The purpose of this paper is to introduce the notion of delegation networks

in a formally sound manner, and to study a few of their basic properties. The semantics of every delegating generator G belonging to a network \mathcal{N} is obtained recursively and depends on the interpretation π of the primitives used. From an operational point of view, G roughly works as follows. First, its internal tree generator is used to generate a tree. This tree is evaluated by interpreting every primitive according to π , and, recursively, every symbol referring to another delegating generator G' according to the semantics of G' .

A delegating generator $G \in \mathcal{N}$ can also be used to generate a tree language $T_{\mathcal{N}}(G)$. We show that the semantics of G can alternatively be obtained by interpreting the trees in $T_{\mathcal{N}}(G)$ according to π . In other words, a Mezei-and-Wright-like result [MW67] is obtained, which makes it particularly interesting to study the tree-generating power of delegation networks. As our main result, we show that

$$DEL(REG) = Y(REG) = DEL(FIN).$$

Here, the following notation is used. For a class C of tree languages, $DEL(C)$ denotes the class of all tree languages generated by delegation networks whose internal tree generators generate tree languages in C . The two classes used in the equation above are REG and FIN , the regular and finite tree languages, respectively. Finally, Y denotes the so-called YIELD mapping. The class $Y(REG)$ is well-known to coincide with the IO-context-free tree languages, i.e., the tree languages generated by context-free tree grammars in IO derivation mode [ES77, Corollary 4.12].

The structure of the paper is as follows. In the next section, we recall some basic notions and establish the notation used. In Section 3, we formalize the notion of delegation networks, thereafter discussing a small example in Section 4. Section 5 studies some of the basic properties of delegation networks. The main result of this paper is proved in Section 6. Section 7 concludes the paper.

2 Preliminaries

In this section, some notions that are needed to formalize the concept of delegating generators are compiled.

2.1 Basic notation

We denote the set of natural numbers (including zero) by \mathbb{N} . For $n \in \mathbb{N}$, the set $\{1, \dots, n\}$ is denoted by $[n]$. The powerset of a set \mathbb{A} is denoted by $\wp(\mathbb{A})$. For a function $f: \mathbb{A} \rightarrow \mathbb{B}$ and $a_1, \dots, a_k \in \mathbb{A}$, we abbreviate the tuple $(f(a_1), \dots, f(a_k))$ by $\vec{f}(a_1, \dots, a_k)$.

The composition of functions $f: \mathbb{A} \rightarrow \mathbb{B}$ and $g: \mathbb{B} \rightarrow \mathbb{C}$ is denoted by $g \circ f$, i.e., $(g \circ f)(a) = g(f(a))$ for all $a \in \mathbb{A}$. In fact, we shall also use the composition $g \circ (f_1, \dots, f_k)$ of functions $f_i: \mathbb{A} \rightarrow \mathbb{B}_i$ ($i \in [k]$) with $g: \mathbb{B}_1 \times \dots \times \mathbb{B}_k \rightarrow \mathbb{C}$, where $h = g \circ (f_1, \dots, f_k)$ is the function $h: \mathbb{A} \rightarrow \mathbb{C}$ given by

$$h(a) = g(f_1(a), \dots, f_k(a))$$

for all $a \in \mathbb{A}$.

2.2 Nondeterministic functions

A nondeterministic function from $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k$ to \mathbb{A} is a function of the form $\varphi: \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_k) \rightarrow \wp(\mathbb{A})$ such that, for all $A_1 \subseteq \mathbb{A}_1, \dots, A_k \subseteq \mathbb{A}_k$ and all $i \in [k]$,

$$\varphi(A_1, \dots, A_k) = \bigcup_{a \in A_i} \varphi(A_1, \dots, A_{i-1}, \{a\}, A_{i+1}, \dots, A_k).$$

We write $\varphi: \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ in order to indicate that φ is a function of this kind, and call $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ its *profile*. If $k = 0$, then f is identified with the set $f() \subseteq \mathbb{A}$, and we write $f: \mathbb{A}$ instead of $f: \twoheadrightarrow \mathbb{A}$, saying that \mathbb{A} is its profile.

A nondeterministic function $\varphi: \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ can be identified with the function $\varphi': \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \rightarrow \wp(\mathbb{A})$ given by $\varphi'(a_1, \dots, a_k) = \varphi(\{a_1\}, \dots, \{a_k\})$ for all $a_1 \in \mathbb{A}_1, \dots, a_k \in \mathbb{A}_k$, because the two determine each other uniquely. In the following, we will therefore write $\varphi(a_1, \dots, a_k)$ instead of $\varphi(\{a_1\}, \dots, \{a_k\})$. In fact, viewing φ as a function that maps $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k$ into $\wp(\mathbb{A})$ explains why it is called nondeterministic: $\varphi(a_1, \dots, a_k)$ should be interpreted as the set of all results one may nondeterministically get by applying φ to the given arguments.

Another observation is that a partial function $\varphi: \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \rightarrow \mathbb{A}$ can be regarded as the special case where $|\varphi(a_1, \dots, a_k)| \leq 1$ for all $(a_1, \dots, a_k) \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_k$. The completely undefined partial function $\varphi: \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ is denoted by \perp , i.e., $\perp(a_1, \dots, a_k) = \emptyset$ for all $a_1 \in \mathbb{A}_1, \dots, a_k \in \mathbb{A}_k$.

If Φ is a set of functions of profile $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$, then $\bigcup \Phi$ denotes the function $\varphi: \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ such that $\varphi(a_1, \dots, a_k) = \bigcup_{\varphi \in \Phi} \varphi'(a_1, \dots, a_k)$. The following fact, which is used later on, states that union distributes over composition.

Fact 2.1 If φ is a function of the form $\varphi = \bigcup \Phi \circ (\bigcup \Phi_1, \dots, \bigcup \Phi_k)$, where $\Phi, \Phi_1, \dots, \Phi_k$ are sets of nondeterministic functions (of appropriate profiles), then

$$\varphi = \bigcup \{\varphi \circ (\varphi_1, \dots, \varphi_k) \mid \varphi \in \Phi \text{ and } \varphi_i \in \Phi_i \text{ for } i \in [k]\}.$$

2.3 Signatures and trees

Let S be a set of *sorts*. A set is *S-sorted* if each element of this set has an associated *profile* $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$, where $\mathbb{A}_1, \dots, \mathbb{A}_k, \mathbb{A} \in S$. Note that these profiles differ from the ones introduced above in that they consist of abstract sorts rather than concrete domains. Despite this fact, we will use similar notational conventions. In particular, we denote the fact that an object G has a certain profile ψ by writing $G: \psi$. An *S-sorted signature* (or simply *signature*) is an *S-sorted* set of function symbols. A function symbol of profile \mathbb{A} (where $\mathbb{A} \in S$) is a *constant symbol*.

For an *S-sorted* signature Σ and $\mathbb{A} \in S$, the set $T_\Sigma^\mathbb{A}$ of all *trees over Σ of sort \mathbb{A}* is defined by simultaneous induction, as usual: if $f: \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ is a symbol in Σ and t_1, \dots, t_k are trees in $T_\Sigma^{\mathbb{A}_1}, \dots, T_\Sigma^{\mathbb{A}_k}$, resp., then $f[t_1, \dots, t_k] \in T_\Sigma^\mathbb{A}$. We identify $f[]$ (where $k = 0$) with f . In this sense, every $f: \mathbb{A}$ in Σ is in $T_\Sigma^\mathbb{A}$.

The set T_Σ of all *trees over* Σ is given by $T_\Sigma = \bigcup_{\mathbf{A} \in S} T_\Sigma^{\mathbf{A}}$. A *tree generator* is any sort of formal device γ that specifies a *tree language* $L(\gamma) \subseteq T_\Sigma$, for some signature Σ .

In the following, we want to represent complex operations by trees. For this purpose, we need a way to distinguish symbols which represent the parameters of such an operation. Let $\mathbf{A}_1, \dots, \mathbf{A}_k \in S$ be sorts, for some $k \in \mathbb{N}$. An (S -sorted) *parameter signature of type* $\mathbf{A}_1 \times \dots \times \mathbf{A}_k$ is a pair (X, pos) , where X is an S -sorted signature containing only constant symbols, and pos is a mapping $pos: X \rightarrow [k]$, such that every symbol $x \in X$ has the profile $\mathbf{A}_{pos(x)}$. Later on, such a symbol will correspond to the $pos(x)$ -th parameter of a generated operation. Note that pos is not required to be injective, i.e., several $x \in X$ may correspond to the same parameter position. Unless there is reason to suspect confusion, we will denote a parameter signature by X , without explicitly mentioning pos . In all situations in which the union of a parameter signature and an “ordinary” signature plays a role, the two are assumed to be disjoint. This fact will not be mentioned explicitly.

2.4 Evaluation

Let Σ be an S -sorted signature. A Σ -*interpretation* σ associates with every sort $\mathbf{A} \in S$ a domain \mathbf{A}_σ and with every function symbol $f: \mathbf{A}_1 \times \dots \times \mathbf{A}_k \rightarrow \mathbf{A}$ in Σ a function $\sigma(f): \mathbf{A}_{1\sigma} \times \dots \times \mathbf{A}_{k\sigma} \rightarrow \mathbf{A}_\sigma$. In the literature, the pair $((\mathbf{A}_\sigma)_{\mathbf{A} \in S}, (\sigma(f))_{f \in \Sigma})$ is also called a (nondeterministic) Σ -algebra. If σ is clear from the context, we denote the domain \mathbf{A}_σ that it assigns to a given sort \mathbf{A} by \mathbb{A} . Given another S -sorted signature $\Sigma' \supseteq \Sigma$, a Σ' -interpretation σ' is a Σ' -*extension of* σ if σ is the restriction of σ' to Σ , i.e., $\mathbf{A}_{\sigma'} = \mathbf{A}_\sigma$ for all $\mathbf{A} \in S$, and $\sigma'(f) = \sigma(f)$ for all $f \in \Sigma$.

Given a Σ -interpretation σ and a parameter signature X of type $\mathbf{A}_1 \times \dots \times \mathbf{A}_k$, we can evaluate the trees in $T_{\Sigma \cup X}$. The *value* of $t \in T_{\Sigma \cup X}^{\mathbf{A}}$ (with respect to σ) is denoted by $val_\sigma^X(t)$. It is the function $\varphi: \mathbb{A}_1 \times \dots \times \mathbb{A}_k \rightarrow \mathbb{A}$ given as follows:

- If $t \in X$, then $\varphi(A_1, \dots, A_k) = A_{pos(t)}$ for all $A_1 \subseteq \mathbb{A}_1, \dots, A_k \subseteq \mathbb{A}_k$.
- Otherwise, if $t = f[t_1, \dots, t_n]$, then $\varphi = \sigma(f) \circ \vec{val}_\sigma^X(t_1, \dots, t_n)$.¹

Given a set $T \subseteq T_{\Sigma \cup X}^{\mathbf{A}}$, we let $val_\sigma^X(T) = \bigcup_{t \in T} val_\sigma^X(t)$.

Thus, val_σ^X is the unique Σ -homomorphism from $T_{\Sigma \cup X}$, the free term algebra over Σ generated by X , to the Σ -algebra given by σ . In the literature, $val_\sigma(t)$ is sometimes called a *derived operation*. During the remainder of this paper, we shall generally drop the superscript X in the notation val_σ^X , thus writing $val_\sigma(t)$ and $val_\sigma(T)$. This slight ambiguity should not cause any problems because the parameter signature in question will always be clear from the context.

3 Delegation networks

Let us now define the main notion of this paper, the delegation network. Such a delegation network is a finite S -sorted set of devices called delegating generators. In other words, each delegating generator has a certain profile. Semantically,

¹Recall that $\vec{val}_\sigma^X(t_1, \dots, t_k)$ abbreviates $(val_\sigma^X(t_1), \dots, val_\sigma^X(t_k))$.

the idea is that a delegating generator of profile $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k \rightarrow \mathbf{A}$ generates a set of trees whose evaluation yields a function of profile $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k \rightarrow \mathbb{A}$. The generator consists of two components, namely a parameter signature X and a tree generator γ that generates trees of sort \mathbf{A} . Each symbol in a tree generated by γ is either a parameter symbol, another delegating generator from the network or a so-called primitive.

The formal definition of delegation networks reads as follows.

Definition 3.1 (delegation network and delegating generator)

Let Π be an S -sorted signature for some set S of sorts. A *delegating network with primitives in Π* is a finite set \mathcal{N} of S -sorted *delegating generators*. Each delegating generator of profile $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k \rightarrow \mathbf{A}$ in \mathcal{N} is a pair $G = (X, \gamma)$, where

- X is a finite parameter signature of type $\mathbf{A}_1 \times \cdots \times \mathbf{A}_k$, and
- γ is a tree generator that generates trees in $\mathbb{T}_{\Sigma_{\mathcal{N}} \cup X}^{\mathbf{A}}$.

Here, $\Sigma_{\mathcal{N}} = \Pi \cup \{F_G \mid G \in \mathcal{N}\}$, where $F_G: \psi$ is a new symbol for every $G: \psi$ in \mathcal{N} .

In the following, the distinction between $G \in \mathcal{N}$ and the corresponding symbol $F_G \in \Sigma_{\mathcal{N}}$ is dropped. Thus, $\Sigma_{\mathcal{N}} = \Pi \cup \mathcal{N}$, using every $G \in \mathcal{N}$ in a double role as a delegating generator and a symbol.

The semantics of \mathcal{N} depends on the interpretation of primitives. Given a Π -interpretation π , the resulting semantics of \mathcal{N} is a certain $\Sigma_{\mathcal{N}}$ -extension of π , which we denote by $\pi_{\mathcal{N}}$. Intuitively, for a delegating generator G as in the definition, we obtain $\pi_{\mathcal{N}}(G)$ by evaluating the trees in $L(\gamma)$. To see what this means, consider a tree $t \in L(\gamma)$. According to Definition 3.1, every non-parameter symbol $F: \psi$ occurring in t is either a primitive or a delegating generator of profile ψ . Thus, both cases yield an appropriate interpretation of F (using recursion if $F \in \mathcal{N}$), which can be used to evaluate t in the way defined earlier.

However, unfortunately, the situation is not as simple as it intuitively might seem, because delegation networks can be cyclic. This invalidates the simple inductive definition the previous paragraph may have suggested. For this reason, we generate the semantics of a delegation network in a stepwise manner.

Definition 3.2 (semantics of delegating generators) Let \mathcal{N} be a delegation network with primitives in Π , and let π be a Π -interpretation. For all $i \in \mathbb{N}$, let $\pi_{\mathcal{N},i}$ be the $\Sigma_{\mathcal{N}}$ -extension of π such that, for every $G = (X, \gamma) \in \mathcal{N}$,

- $\pi_{\mathcal{N},0}(G) = \perp$ and
- $\pi_{\mathcal{N},i+1}(G) = \text{val}_{\pi_{\mathcal{N},i}}(L(\gamma))$ for $i \in \mathbb{N}$.

Now, the *semantics of \mathcal{N} (with respect to π)* is the $\Sigma_{\mathcal{N}}$ -extension $\pi_{\mathcal{N}}$ of π such that $\pi_{\mathcal{N}}(G) = \bigcup_{i \in \mathbb{N}} \pi_{\mathcal{N},i}(G)$ for every $G \in \mathcal{N}$. If G is of profile \mathbf{A} for some sort \mathbf{A} , then $\pi_{\mathcal{N}}(G) \subseteq \mathbb{A}$ is called the *language generated by G* .

4 An example

Let us now discuss an example that, although rather simple, can be used to illustrate the possibilities provided by delegation networks. For examples such as this one, and even more for future applications, it is appropriate to discard the (theoretically convenient) distinction between primitives and their interpretation. Throughout this section, we will therefore assume that the signature Π of primitives comes with a fixed interpretation π . As a consequence, we may identify every symbol $F \in \Pi$ with $\pi(F)$, thus simplifying our notation by writing $F(\dots)$ instead of $\pi(F)(\dots)$.

From an implementation point of view, one may think of delegating generators as a nondeterministic device working as follows. Let $G = (X, \gamma)$ be a delegating generator of profile $\mathbf{A}_1 \times \dots \times \mathbf{A}_k \rightarrow \mathbf{A}$ that belongs to a delegation network \mathcal{N} , and let $a_1 \in \mathbf{A}_1, \dots, a_k \in \mathbf{A}_k$ be arguments. We may then nondeterministically “execute” G in order to produce an element of the set $\pi_{\mathcal{N}}(G)(a_1, \dots, a_k)$. For this purpose, we first run γ , which produces a tree $t \in \mathbf{T}_{\Sigma_{\mathcal{N}} \cup X}^{\mathbf{A}}$. This tree is evaluated in a bottom-up manner by assigning a value to each of its subtrees. To each occurrence of a variable $x \in X$, the value $a_{\text{pos}(x)}$ is assigned. Now, consider a subtree $s = F[s_1, \dots, s_l]$ with $F \in \Sigma_{\mathcal{N}}$, and suppose that we have already assigned values b_1, \dots, b_l to s_1, \dots, s_l , resp. There are two cases.

- If $F \in \Pi$, we choose nondeterministically any element of $F(b_1, \dots, b_l)$ as the value of s .
- If $F \in \mathcal{N}$, we execute (recursively) an instance of F with arguments b_1, \dots, b_l and consider the result to be the value of s .

Note that the base case is the one where $F \in \Pi$ for all non-parameter symbols F occurring in t . Of course, a naive implementation may lead to an infinitely descending recursion because of the third case, if the delegation structure is cyclic. Thus, some care must be taken in an implementation.

As a side remark, one may point out that delegation networks are well suited for parallel and even for distributed implementations. This is because many tree generators generate their output trees in some stepwise manner. If γ is of this kind, one may create an instance of F each time an occurrence of $F \in \mathcal{N}$ is generated. The execution of this instance may be started even if γ is still running, and it may run in parallel with all other instances of delegating generators created in this way.

Our example makes use of two domains, namely the set \mathbb{P} of two-dimensional pictures (i.e., subsets of \mathbb{R}^2) and the set \mathbb{N} of natural numbers. We denote the corresponding sorts by \mathbf{P} and \mathbf{N} , resp. Besides the constant 0 and the successor function s on \mathbb{N} , two types of primitives are used, which are explained next.

Picture operations A *picture operation* in the sense of [Dre06, Chapter 4] maps \mathbb{P}^k to \mathbb{P} and is denoted by $\langle f_1 \dots f_k, p \rangle$, where f_1, \dots, f_k are affine transformations, and $p \in \mathbb{P}$. It is defined by

$$\langle f_1 \dots f_k, p \rangle(p_1, \dots, p_k) = p \cup \bigcup_{i \in [k]} f_i(p_i)$$

for all $p_1, \dots, p_k \in \mathbb{P}$.

Cellular automata The concept of cellular automata (CA) was developed by von Neumann, with contributions by Ulam, Zuse, and others, in the middle of the last century. A (two-dimensional) CA ca is a parallel device that operates on an infinite two-dimensional array of cells C_{ij} ($i, j \in \mathbb{Z}$). Geometrically, we identify the cell C_{ij} with the unit square whose lower left corner has the coordinates (i, j) . The cellular automaton consists of two components. The first is a set $Q = \{0, \dots, k\}$ of states, where $k > 0$. At each moment in time, every cell contains one of these states. The second component of ca is a transition function of the form $\Delta: Q^{3 \times 3} \rightarrow Q$, where $\Delta\left(\begin{smallmatrix} 000 \\ 000 \\ 000 \end{smallmatrix}\right) = 0$. Initially, the so-called inactive state 0 is assigned to all cells C_{ij} except C_{00} , which is assigned the state 1. In each step, all cells synchronously update their states according to Δ and the states of cells in their neighbourhood. More precisely, each cell C_{ij} changes its state to $\Delta(N)$, where N is the 3×3 array of states of its neighbouring cells (including C_{ij} itself), i.e., the states of the cells C_{pq} such that $i - 1 \leq p \leq i + 1$ and $j - 1 \leq q \leq j + 1$. For example, after the first step, the state of C_{00} will be $\Delta\left(\begin{smallmatrix} 000 \\ 010 \\ 000 \end{smallmatrix}\right)$, and the state of C_{11} will be $\Delta\left(\begin{smallmatrix} 000 \\ 000 \\ 100 \end{smallmatrix}\right)$ since C_{11} has C_{00} as its lower left neighbour. Note that the number of active cells will always remain finite, because $\Delta\left(\begin{smallmatrix} 000 \\ 000 \\ 000 \end{smallmatrix}\right) = 0$.

Now, to turn a cellular automaton ca as above into a primitive, we view it as an operation $ca: \mathbb{N} \times \mathbb{P}^k \rightarrow \mathbb{P}$, where $ca(n, p_1, \dots, p_k)$ is obtained as follows. First, ca is executed n steps. Then, for every cell whose current state is $i \in [k]$, a copy of p_i is horizontally and vertically scaled and translated in such a way that the cell C_{ij} becomes its bounding box². The resulting picture is the union of all these transformed copies of p_1, \dots, p_k .

Note that one could alternatively view ca as a nondeterministic primitive $ca': \mathbb{P}^k \rightarrow \mathbb{P}$, where $ca'(p_1, \dots, p_k) = \{ca(n, p_1, \dots, p_k) \mid n \in \mathbb{N}\}$. Obviously, this would not allow for as much control as the variant used here.

In this example, we use only one nontrivial cellular automaton ca , where $k = 3$. Rather than defining ca formally, let us use a delegation network consisting of a single generator to show how ca behaves. The primitives are ca , 0 , s , and pictures p_i , $1 \leq i \leq 3$. The p_i are hollow squares with different figures placed inside: a square with indented edges, a triangle, and a hollow circle, resp. The delegating generator is $G = (\emptyset, \gamma)$, where γ is a tree generator (e.g., a regular tree grammar; see below) with $L(\gamma) = \{ca[s^n[0], p_1, p_2, p_3] \mid n \in \mathbb{N}\}$. The resulting pictures for $0 \leq n \leq 6$ are shown in Figure 1.

Now, let us discuss a delegation network that makes use of ca in a more interesting way. It consists of three delegating generators, G , CA , and IFS , where G uses the other two via delegation. Moreover, IFS delegates to itself. All tree generators employed are regular tree grammars. Readers who have never encountered this type of tree generator before, may first wish to look up Definition 6.1.

²For the purpose of this example, we may disregard the degenerated cases where such a scaling is impossible.

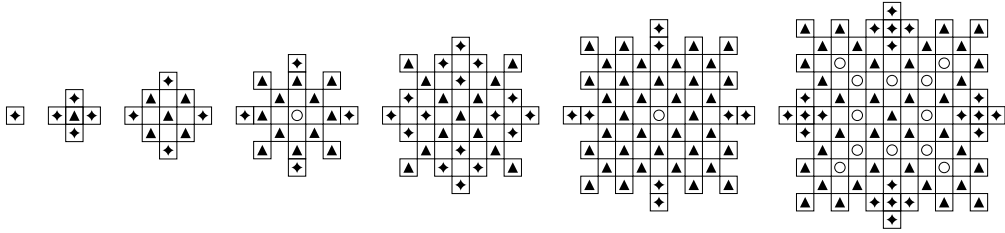


Figure 1: Initial steps of the cellular automaton ca used as an example



Figure 2: Pictures generated by IFS , if applied to \blacklozenge (up to scaling)

Let us first discuss IFS , which is a delegating generator of profile $P \rightarrow P$ that implements an iterated function system. Its tree generator generates the finite tree language $\{x, IFS[F[x, x, x]]\}$, using a single variable x of profile P . Here, F is a picture operation of the form $\langle f_1 f_2 f_3, p \rangle$. The transformations f_1, f_2, f_3 and the picture p are chosen in such a way that

$$F(\blacklozenge, \blacklozenge, \blacklozenge) = \blacklozengetree.$$

Because of the recursion (note that IFS delegates to itself), $\pi_{\mathcal{N}}(IFS)(\blacklozenge)$ yields the set of pictures indicated in Figure 2.

Now, let us discuss CA . Its tree generator γ_{CA} is the regular tree grammar with the nonterminals C_0 and C , where C_0 is the initial one, and the rules

$$\begin{aligned} C_0 &\rightarrow scale[0, C], \\ C &\rightarrow ca[n, C, x_0, y_0], \\ C &\rightarrow ca[n, x, y, z]. \end{aligned}$$

Here, the parameter signature is of type $\mathbb{N} \times P^5$, the corresponding variables being n, x_0, y_0, x, y, z (one for each parameter position). The cellular automaton ca is as defined above, while $scale$ is the trivial cellular automaton with states 0, 1 and no rules at all. Thus, the effect of $scale(0, p)$ is to scale and translate p in such a way that its bounding box becomes the unit square having its lower left corner at the origin.

Clearly, γ_{CA} generates all trees

$$t_m^{CA} = scale[0, \underbrace{ca[n, ca[n, \dots ca[n, ca[n, x, y, z], x_0, y_0] \dots, x_0, y_0], x_0, y_0}],$$

m times

where $m \in \mathbb{N}$. Note that the recursion takes place in the argument corresponding to state 1 of ca (i.e., the squarish one in Figure 1), and all instances of ca perform the same number of steps, as determined by the parameter n .

Finally, the delegating generator G is of profile P . Its tree generator is the

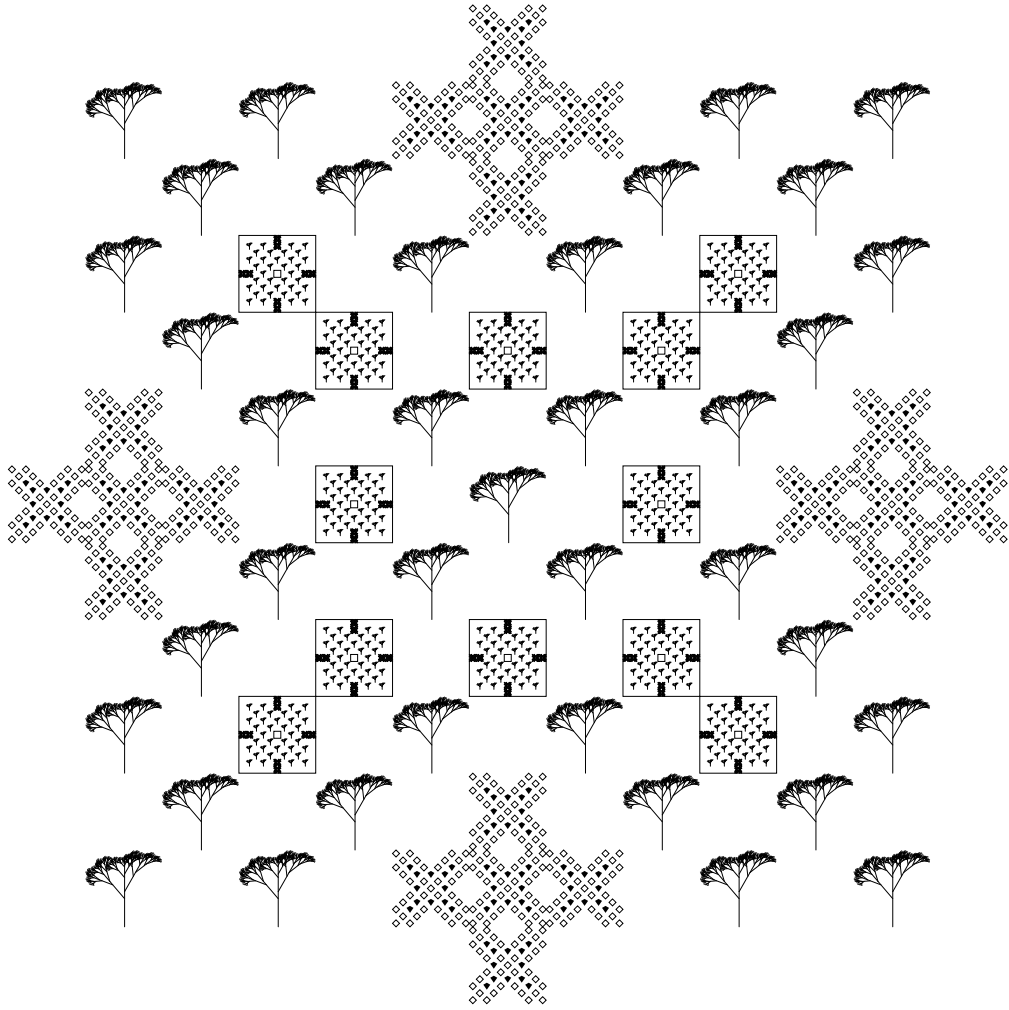


Figure 3: A picture generated by the delegating generator G

regular tree grammar γ_G with nonterminals S (initial) and A , and the rules

$$\begin{aligned}
 S &\rightarrow CA[A, IFS, frame[S], \emptyset, \diamond, \blacklozenge], \\
 S &\rightarrow \emptyset, \\
 A &\rightarrow s[A], \\
 A &\rightarrow 0.
 \end{aligned}$$

Here, \emptyset and $frame$ denote the empty picture and the picture operation that does not transform its argument at all, but adds a hollow unit square to it.

One of the pictures in the language generated by G is shown in Figure 3. Abbreviating $s^n[0]$ by n , the tree in $L(\gamma_G)$ that gives rise to this picture is

$$CA[6, IFS, frame[CA[5, IFS, frame[\emptyset], \emptyset, \diamond, \blacklozenge]], \emptyset, \diamond, \blacklozenge],$$

using t_2^{CA} “inside” both occurrences of CA . It is instructive to compare Figure 3 with Figure 1. In particular, one should note the different levels at which the structures of the two rightmost pictures of Figure 1 appear in Figure 3:

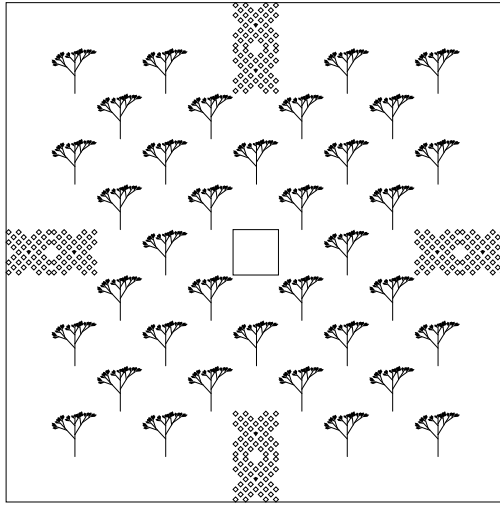


Figure 4: A closeup of one of the framed parts of Figure 3

the rightmost one (corresponding to a call $ca(6, \dots)$) appears at two levels of recursion – although with different subpictures as parameters. Likewise, the one to its left (corresponding to a call $ca(5, \dots)$) occurs at two levels of recursion inside the framed subpictures (see the closeup in Figure 4).

5 Basic properties

In this section, we discuss the semantics of delegating generators from several different perspectives. We start with the observation that Definition 3.1 provides a fixed-point construction. To make this precise, let us turn the set of all functions of profile $\mathbb{A}_1 \times \dots \times \mathbb{A}_k \rightarrow \mathbb{A}$ into a complete lattice by defining $f \leq g$ if and only if $f(a_1, \dots, a_k) \subseteq g(a_1, \dots, a_k)$, for all $a_1 \in \mathbb{A}_1, \dots, a_k \in \mathbb{A}_k$. This extends to interpretations of S -sorted signatures in the obvious way: if σ and σ' are two Σ -interpretations such that $\mathbf{A}_\sigma = \mathbf{A}_{\sigma'}$ for all $\mathbf{A} \in S$, then $\sigma \leq \sigma'$ means that $\sigma(f) \leq \sigma'(f)$ for all $f \in \Sigma$.

Now, consider a delegation network \mathcal{N} and a $\Sigma_{\mathcal{N}}$ -extension σ of π . By rephrasing the construction in Definition 3.1, we get an operator $iterate_\pi$ on $\Sigma_{\mathcal{N}}$ -interpretations, where $iterate_\pi(\sigma) = \sigma'$ is given by

$$\sigma'(F) = \begin{cases} val_\sigma(L(\gamma)) & \text{if } F = (X, \gamma) \in \mathcal{N}, \\ \pi(F) & \text{otherwise.} \end{cases}$$

Using these definitions, the following theorem is obtained.

Theorem 5.1 For every delegation network \mathcal{N} and every Π -interpretation π , $\pi_{\mathcal{N}}$ is the least fixed point of $iterate_\pi$. More precisely,

1. $iterate_\pi(\pi_{\mathcal{N}}) = \pi_{\mathcal{N}}$ and
2. $\pi_{\mathcal{N}} \leq \sigma$ for all $\Sigma_{\mathcal{N}}$ -interpretations σ for which $iterate_\pi(\sigma) = \sigma$.

Proof Straightforward, using standard arguments. ■

Now, let us consider a delegation network \mathcal{N} with primitives in Π . Clearly, \mathcal{N}

can be used to generate (operations on) trees in T_Π . We just have to choose a Π -interpretation τ such that $\mathbf{A}_\tau = T_\Pi^{\mathbf{A}}$ for all $\mathbf{A} \in S$. In particular, we can make a free construction by defining $\tau(F)(t_1, \dots, t_k) = F[t_1, \dots, t_k]$, for every symbol $F: \mathbf{A}_1 \times \dots \times \mathbf{A}_k \rightarrow \mathbf{A}$ in Π and all trees t_1, \dots, t_k of sorts $\mathbf{A}_1, \dots, \mathbf{A}_k$. In the following, we shall continue to denote this specific Π -interpretation by τ .

Notation 5.2 (parameter substitution) Let X be a parameter signature of type $\mathbf{A}_1 \times \dots \times \mathbf{A}_k$, and consider trees $t \in T_{\Pi \cup X}$, $t_1 \in T_{\Pi \cup X'}^{\mathbf{A}_1}, \dots, t_k \in T_{\Pi \cup X'}^{\mathbf{A}_k}$, where X' is any parameter signature. Obviously, $val_\tau(t)(t_1, \dots, t_k)$ is the tree obtained by substituting $t_{pos(x)}$ for each occurrence of a parameter $x \in X$ in t . This type of substitution is called *parameter substitution*; we abbreviate $val_\tau(t)(t_1, \dots, t_k)$ by $t(t_1, \dots, t_k)$. Similarly, for $T \subseteq T_{\Pi \cup X}$, $T(t_1, \dots, t_k)$ denotes $val_\tau(T)(t_1, \dots, t_k)$.

The following lemma, stating that evaluation distributes over parameter substitution, is well known. In fact, it is a consequence of the earlier mentioned fact that val_σ is a homomorphism.

Lemma 5.3 Let X, X' be parameter signatures, where X is of type $\mathbf{A}_1 \times \dots \times \mathbf{A}_k$. For all trees $t \in T_{\Pi \cup X}$, $t_1 \in T_{\Pi \cup X'}^{\mathbf{A}_1}, \dots, t_k \in T_{\Pi \cup X'}^{\mathbf{A}_k}$,

$$val_\sigma(t(t_1, \dots, t_k)) = val_\sigma(t) \circ \vec{val}_\sigma(t_1, \dots, t_k).$$

By definition, $\tau_{\mathcal{N}}$ associates a mapping $\tau_{\mathcal{N}}(G): T_\Pi^{\mathbf{A}_1} \times \dots \times T_\Pi^{\mathbf{A}_k} \rightarrow T_\Pi^{\mathbf{A}}$ with every $G = (X, \gamma) \in \mathcal{N}$ of profile $\mathbf{A}_1 \times \dots \times \mathbf{A}_k \rightarrow \mathbf{A}$. In fact, $\tau_{\mathcal{N}}(G) = val_\tau(T)$ for a suitable set $T \subseteq T_\Sigma^X$. We are now going to show how such a set T can be constructed by iteratively replacing symbols by trees. The kind of replacement required for this is defined next.

Definition 5.4 (IO-replacement) Consider signatures Σ, Σ' where $\Sigma' \subseteq \Sigma$, and let X be a parameter signature. A Σ' -assignment is a mapping α that assigns to each $f: \mathbf{A}_1 \times \dots \times \mathbf{A}_k \rightarrow \mathbf{A} \in \Sigma'$ a tree language $\alpha(f) \subseteq T_{\Sigma \cup X'}^{\mathbf{A}}$, where X' is any parameter signature of type $\mathbf{A}_1 \times \dots \times \mathbf{A}_k$.

For every tree $t = f[t_1, \dots, t_k] \in T_{\Sigma \cup X}$, the *IO-replacement* of symbols in t according to α yields

$$t\alpha = \begin{cases} \{s(t'_1, \dots, t'_k) \mid s \in \alpha(f) \text{ and } t'_i \in t_i\alpha \text{ for } i \in [k]\} & \text{if } f \in \Sigma' \\ \{f[t'_1, \dots, t'_k] \mid t'_i \in t_i\alpha \text{ for } i \in [k]\} & \text{otherwise.} \end{cases}$$

For $T \subseteq T_{\Sigma \cup X}$, we let $T\alpha = \bigcup_{t \in T} t\alpha$.

Thus, IO-replacement yields all trees obtained by replacing each occurrence of symbols $f \in \Sigma'$ in t by a tree in $\alpha(f)$. Note that this replacement is done in an inside-out fashion: if a tree in $\alpha(f)$ is nonlinear in the sense that there are two distinct occurrences of parameter symbols x, y with $pos(x) = pos(y)$, then both copies of the corresponding parameter will be identical because they are copied *after* the process has recursively been carried out on them.

Notation 5.5 A Σ' -assignment α may be denoted by a set-like expression of the form $\llbracket f \leftarrow T \mid \psi \rrbracket$, where ψ is a condition specifying both Σ' and $T = \alpha(f)$ for every $f \in \Sigma'$.

Similar to Lemma 5.3, we have the following result stating that evaluation distributes over IO-replacement.

Lemma 5.6 Let Σ, Σ' be signatures with $\Sigma' \subseteq \Sigma$, let X be a parameter signature, and let α be a Σ' -assignment. For every Σ -interpretation σ and every tree $t \in T_{\Sigma \cup X}$,

$$\text{val}_\sigma(t\alpha) = \text{val}_{\sigma'}(t),$$

where, for every symbol $f \in \Sigma$,

$$\sigma'(f) = \begin{cases} \text{val}_\sigma(\alpha(f)) & \text{if } f \in \Sigma' \\ \sigma(f) & \text{otherwise.} \end{cases}$$

Proof We proceed by induction on the structure of t . If $t \in X$, then the statement is obviously true. Now, let $t = f[t_1, \dots, t_k]$, and assume that $f \in \Sigma'$, because the statement follows directly from the induction hypothesis if $f \notin \Sigma'$. Using the relevant definitions, Lemma 5.3, and Fact 2.1, we get

$$\begin{aligned} \text{val}_\sigma(t\alpha) &= \text{val}_\sigma(\{s(t'_1, \dots, t'_k) \mid s \in \alpha(f) \text{ and } t'_i \in t_i\alpha \text{ for } i \in [k]\}) \\ &= \bigcup \{ \text{val}_\sigma(s(t'_1, \dots, t'_k)) \mid s \in \alpha(f) \text{ and } t'_i \in t_i\alpha \text{ for } i \in [k] \} \\ &= \bigcup \{ \text{val}_\sigma(s) \circ \vec{\text{val}}_\sigma(t'_1, \dots, t'_k) \mid s \in \alpha(f) \text{ and } t'_i \in t_i\alpha \text{ for } i \in [k] \} \\ &= \bigcup \{ \text{val}_\sigma(s) \mid s \in \alpha(f) \} \circ \bigcup \{ \vec{\text{val}}_\sigma(t'_1, \dots, t'_k) \mid t'_i \in t_i\alpha \text{ for } i \in [k] \} \\ &= \text{val}_\sigma(\alpha(f)) \circ \vec{\text{val}}_{\sigma'}(t_1, \dots, t_k) \\ &= \sigma'(f) \circ \vec{\text{val}}_{\sigma'}(t_1, \dots, t_k) \\ &= \text{val}_{\sigma'}(t), \end{aligned}$$

which completes the proof. \blacksquare

Note that Lemma 5.6 remains correct if t is replaced by a subset T of $T_{\Sigma \cup X}$, because $\text{val}_\sigma(T\alpha) = \text{val}_\sigma(\bigcup_{t \in T} t\alpha) = \bigcup_{t \in T} \text{val}_\sigma(t\alpha)$ and $\text{val}_{\sigma'}(T) = \bigcup_{t \in T} \text{val}_{\sigma'}(t)$.

Intuitively, one may think of a delegating generator G as a device that produces (functions on) objects belonging to some domain by generating trees that evaluate to these objects. Delegation means that the trees may contain occurrences of other delegating generators $F \in \mathcal{N}$ that evaluate to functions on the type of objects considered. If we disregard the evaluation step, we may say that F generates a tree that resides “within” the corresponding node of the tree generated by G . This recursive structure of trees within trees can be flattened using IO-replacement, thus resulting in a tree language $T_{\mathcal{N}}(G)$ generated by G . The following definition makes this precise.

Definition 5.7 Let \mathcal{N} be a delegation network. The family $(T_{\mathcal{N}}(G))_{G \in \mathcal{N}}$ consists of the smallest tree languages $T_{\mathcal{N}}(G)$ such that, for all $G = (X, \gamma) \in \mathcal{N}$,

$$T_{\mathcal{N}}(G) = L(\gamma) \llbracket F \leftarrow T_{\mathcal{N}}(F) \mid F \in \mathcal{N} \rrbracket.$$

It should be clear that Definition 5.7 is sound. This is because $T_{\mathcal{N}}(G)$ can be constructed by a process similar to the one in Definition 3.1. For this, let $T_{\mathcal{N},0}(G) = \emptyset$ and $T_{\mathcal{N},i+1}(G) = L(\gamma)[[F \leftarrow T_{\mathcal{N},i}(F) \mid F \in \mathcal{N}]]$. Then one can easily verify that $T_{\mathcal{N}}(G) = \bigcup_{i \in \mathbb{N}} T_{\mathcal{N},i}(G)$. It should be noted that $T_{\mathcal{N}}$ can be interpreted as an \mathcal{N} -assignment. To emphasize this view, this particular \mathcal{N} -assignment is denoted by $\alpha_{\mathcal{N}}$, i.e., $\alpha_{\mathcal{N}}(G) = T_{\mathcal{N}}(G)$ for all $G \in \mathcal{N}$. Thus, the equality defining $T_{\mathcal{N}}(G)$ can be rewritten as $T_{\mathcal{N}}(G) = L(\gamma)\alpha_{\mathcal{N}}$.

The construction of $T_{\mathcal{N}}(G)$ makes proofs by induction on i possible. To make this more convenient, we formulate an induction principle which avoids the necessity to deal with i explicitly. For this, we say that a property Φ of tree languages is *positively continuous* if the following holds: For all sequences of tree languages $T_0 \subseteq T_1 \subseteq \dots$, if each T_i ($i \in \mathbb{N}$) has property Φ , then $\bigcup_{i \in \mathbb{N}} T_i$ has property Φ as well.

Lemma 5.8 (induction principle for delegation networks) Let \mathcal{N} be a delegation network. For every $G \in \mathcal{N}$, let Φ_G be a positively continuous property of tree languages. Then $T_{\mathcal{N}}(G)$ has property Φ_G for all $G \in \mathcal{N}$, provided that the following hold:

1. For every $G \in \mathcal{N}$, the empty tree language has property Φ_G .
2. Let α be any \mathcal{N} -assignment such that each of the tree languages $\alpha(G)$, $G \in \mathcal{N}$, has property Φ_G . Then $L(\gamma)\alpha$ has also property Φ_G , for every $G = (X, \gamma) \in \mathcal{N}$.

Proof Obvious from the preceding discussion. ■

To finish this section, we show that the semantics of a delegating generator $G \in \mathcal{N}$ (with respect to any interpretation of primitives) can be obtained by a strategy which may be called lazy evaluation: rather than evaluating every tree as soon as it is produced by a generator to which G has delegated part of its job, we work with trees all the way, thus generating $T_{\mathcal{N}}(G)$, and evaluate the trees afterwards. In other words, we have obtained a result in the style of Mezei and Wright [MW67].

Theorem 5.9 Let \mathcal{N} be a delegation network with primitives in Π , and let $G = (X, \gamma) \in \mathcal{N}$. For every Π -interpretation π , it holds that

$$\pi_{\mathcal{N}}(G) = \text{val}_{\pi}(T_{\mathcal{N}}(G)).$$

Proof Clearly, evaluation of sets of trees distributes over infinite unions. As a consequence, $\text{val}_{\pi}(T_{\mathcal{N}}(G)) = \bigcup_{i \in \mathbb{N}} \text{val}_{\pi}(T_{\mathcal{N},i}(G))$, where $T_{\mathcal{N},i}(G)$ is defined as in the paragraph following Definition 5.7. Hence, the proof is finished if we can show (by induction on $i \in \mathbb{N}$) that

$$\pi_{\mathcal{N},i}(G) = \text{val}_{\pi}(T_{\mathcal{N},i}(G)).$$

For $i = 0$, both sides of the equation are equal to \perp . Now, assume that the assertion holds for some $i \in \mathbb{N}$, i.e., for all $F \in \Sigma_{\mathcal{N}}$,

$$\pi_{\mathcal{N},i}(F) = \begin{cases} \text{val}_{\pi}(T_{\mathcal{N},i}(F)) & \text{if } F \in \mathcal{N} \\ \pi(F) & \text{otherwise.} \end{cases}$$

Using this, as well as Lemma 5.6 and the relevant definitions, we get

$$\begin{aligned}\pi_{\mathcal{N},i+1}(G) &= \text{val}_{\pi_{\mathcal{N},i}}(L(\gamma)) \\ &= \text{val}_{\pi}(L(\gamma)[[F \leftarrow T_{\mathcal{N},i}(F) \mid F \in \mathcal{N}]]) \\ &= \text{val}_{\pi}(T_{\mathcal{N},i}(G)),\end{aligned}$$

as claimed. ■

6 The tree-generating power of delegating generators

Theorem 5.9 shows that the generating power of any class DEL of delegation networks is closely related to its tree-generating power, i.e., to the question which tree languages are of the form $T_{\mathcal{N}}(G)$ for some $\mathcal{N} \in \text{DEL}$ and $G \in \mathcal{N}$. Therefore, it is of particular interest to investigate the tree-generating power of delegation networks. In this section, we study one of the most basic cases, namely delegation networks over regular tree grammars. Here, for a class Γ of tree generators, a *delegation network over Γ* is a delegation network consisting of delegating generators (X, γ) such that $\gamma \in \Gamma$. If $C = \{L(\gamma) \mid \gamma \in \Gamma\}$ is the class of tree languages generated by tree generators in Γ , then we denote by $DEL(C)$ the set of all tree languages of the form $T_{\mathcal{N}}(G)$, where \mathcal{N} is a delegation network over Γ and $G \in \mathcal{N}$. (Clearly, this definition is consistent as $DEL(C)$ depends only on C , rather than on the particular choice of Γ .)

For the results presented in this section, the many-sortedness of signatures is irrelevant. Therefore, we can simplify the technicalities by restricting our attention to unsorted signatures. Technically, let us fix an arbitrary sort U . An *unsorted signature* is a $\{U\}$ -sorted signature. In the unsorted case, the notation $f: U^k \rightarrow U$ is abbreviated as $f^{(k)}$, and k is called the rank of f . As another technical simplification, we consider only the (unsorted) parameter signatures $X_k = \{x_i \mid i \in [k]\}$ for $k \in \mathbb{N}$, where $\text{pos}(x_i) = i$ for all $i \in [k]$. Using the fact that the class of regular tree languages is closed under finite-state relabellings, it is easy to see that the results of this section continue to be valid if arbitrary unsorted parameter signatures are considered instead.

Let us now recall the definition of regular tree grammars.

Definition 6.1 (regular tree grammar) A *regular tree grammar* is a tuple $\gamma = (\Xi, \Sigma, R, \xi_{\text{ini}})$, where

- Ξ is a finite unsorted signature of constant symbols called *nonterminals*,
- Σ is a finite unsorted signature of *terminals* which is disjoint with Ξ ,
- R is a finite set of *rules* $\xi \rightarrow r$, where $\xi \in \Xi$ and $r \in T_{\Sigma \cup \Xi}$, and
- $\xi_{\text{ini}} \in \Xi$ is the *initial nonterminal*.

A *derivation step* $s \rightarrow_{\gamma} t$ (or simply $s \rightarrow t$, if γ is understood) consists of two trees $s, t \in T_{\Sigma \cup \Xi}$ such that t is obtained from s by replacing a single occurrence of a nonterminal ξ with r , for some rule $\xi \rightarrow r$ in R . The tree language generated by γ , called a *regular tree language*, is

$$L(\gamma) = \{t \in T_{\Sigma} \mid \xi_{\text{ini}} \rightarrow^+ t\},$$

where \rightarrow^+ denotes the transitive closure of the relation \rightarrow .

The sets of all regular tree grammars and of all regular tree languages are denoted by REG and REG , resp.

The main result of this section characterizes the generative power of delegation networks over REG in terms of REG and the so-called YIELD mapping Y (to be defined soon). In fact, we shall see that delegation networks over REG are equivalent to delegation networks over the much more restricted class FIN . We define FIN to be the class of all regular tree grammars having only one nonterminal (the initial one), and only rules with terminal right-hand sides. Hence, $\text{FIN} = \{L(\gamma) \mid \gamma \in \text{FIN}\}$ is the class of all finite tree languages. Clearly, every regular tree grammar $\gamma \in \text{FIN}$ is uniquely determined by $L(\gamma)$.

YIELD mappings [Mai74] formalize the construction of trees using parameter substitution. Given a finite signature Σ and some $k \in \mathbb{N}$ such that $l \leq k$ for all $f^{(l)} \in \Sigma$, the corresponding YIELD mapping Y maps trees over the signature

$$\Sigma_{Y,k} = \{\text{subst}^{(k+1)}\} \cup \{f_l^{(0)} \mid f^{(l)} \in \Sigma \cup X_k\}$$

to trees over $\Sigma \cup X_k$. In this context, the signature $\Sigma_{Y,k}$ is called a *derived signature*. Thus, in a derived signature, each of the original symbols $f^{(l)}$ in $\Sigma \cup X_k$ is turned into a symbol f_l of rank 0. The only symbol of rank greater than 0 (namely of rank $k + 1$) is subst .

Definition 6.2 (YIELD mapping) Let $\Sigma_{Y,k}$ be a derived signature. The *YIELD mapping* on $\text{T}_{\Sigma_{Y,k}}$ is the mapping $Y: \text{T}_{\Sigma_{Y,k}} \rightarrow \text{T}_{\Sigma \cup X_k}$ which is defined recursively, as follows. For every tree $t \in \text{T}_{\Sigma_{Y,k}}$,

$$Y(t) = \begin{cases} Y(t_0)(Y(t_1), \dots, Y(t_k)) & \text{if } t = \text{subst}[t_0, t_1, \dots, t_k] \\ f[x_1, \dots, x_l] & \text{if } t = f_l \text{ with } f \in \Sigma \cup X_k. \end{cases}$$

Note that, in particular, $Y(x_{i_0}) = x_i$ for all $i \in [k]$. It is easy to see that the definition above is in all relevant aspects equivalent to any of the definitions of Y found in the literature (see, e.g., [ES77, ES78, EV85, FV98]). Given a tree language $T \subseteq \text{T}_{\Sigma_{Y,k}}$, we let $Y(T) = \{Y(t) \mid t \in T\}$. In the following, we shall be interested in the class of all languages which can be obtained by applying the YIELD mapping to regular tree languages. We denote this class by $Y(\text{REG})$, i.e., $Y(\text{REG}) = \{Y(T) \mid T \in \text{REG}\}$. More generally, $Y(C) = \{Y(T) \mid T \in C\}$ for every class C of tree languages.

We can now prove one direction of the main result of this paper.

Lemma 6.3 $Y(\text{REG}) \subseteq \text{DEL}(\text{FIN})$.

Proof Consider a tree language $Y(L(\gamma))$, where $\gamma = (\Xi, \Sigma_{Y,k}, R, \xi_{\text{ini}})$ for a finite unsorted signature Σ and an appropriate $k \in \mathbb{N}$. Using a well-known normal form of regular tree grammars, we can assume that every rule in R is of the form $\xi \rightarrow f[\xi_1, \dots, \xi_l]$, where $f^{(l)} \in \Sigma_{Y,k}$ and $\xi, \xi_1, \dots, \xi_l \in \Xi$.

The idea behind the following construction is to turn every nonterminal $\xi \in \Xi$ into a delegating generator $G_\xi^{(k)}$ that generates all trees of the form $Y(t)$ such that t can be derived from the nonterminal ξ . In particular, this means that $T_{\mathcal{N}}(G_{\xi_{\text{ini}}}) = Y(L(\gamma))$, which proves the lemma.

Let \mathcal{N} be the delegation network with primitives in Σ given as follows. For every nonterminal $\xi \in \Xi$, \mathcal{N} contains the delegating generator $G_\xi^{(k)} = (X_k, \gamma_\xi)$. Here, $L(\gamma_\xi) \in \text{FIN}$ contains one tree for every rule $\xi \rightarrow r$ in R , namely

- (a) $f[x_1, \dots, x_l]$ if $r = f_l$ with $f \in \Sigma \cup X_k$, and
- (b) $G_{\xi_0}[\tilde{G}_{\xi_1}, \dots, \tilde{G}_{\xi_k}]$ if $r = \text{subst}[\xi_0, \xi_1, \dots, \xi_k]$, where $\tilde{G}_{\xi'} = G_{\xi'}[x_1, \dots, x_k]$ for all $\xi' \in \Xi$. (Note that $\tilde{G}_{\xi'}\alpha = \alpha(G_{\xi'})$ for every \mathcal{N} -assignment α .)

Choosing $G_0 = G_{\xi_{\text{ini}}}$, the proof is completed by showing that $T_{\mathcal{N}}(G_\xi) = T_\xi$, for all $\xi \in \Xi$, where $T_\xi = \{Y(t) \mid t \in \mathbb{T}_{\Sigma_{Y,k}} \text{ and } \xi \rightarrow^+ t\}$.

‘ \subseteq ’ We proceed by induction, using Lemma 5.8. Thus, formally, Φ_{G_ξ} states that the tree language in question is a subset of T_ξ . Clearly, Φ_{G_ξ} is positively continuous and is satisfied by the empty tree language.

Now, consider some $\xi \in \Xi$, and let α be some \mathcal{N} -assignment such that $\alpha(G_{\xi'}) \subseteq T_{\xi'}$ for all $\xi' \in \Xi$. We have to show that $t\alpha \subseteq T_\xi$ for all $t \in L(\gamma_\xi)$.

Case 1 $t = f[x_1, \dots, x_l]$ for some $f^{(l)} \in \Sigma \cup X_k$, i.e., the rule $\xi \rightarrow f_l$ is in R .

In this case, we obviously have

$$t\alpha = \{t\} = \{Y(f_l)\} \subseteq T_\xi.$$

Case 2 $t = G_{\xi_0}[\tilde{G}_{\xi_1}, \dots, \tilde{G}_{\xi_k}]$ for nonterminals $\xi_0, \xi_1, \dots, \xi_k \in \Xi$, i.e., the rule $\xi \rightarrow \text{subst}[\xi_0, \xi_1, \dots, \xi_k]$ is in R .

Here, we get

$$\begin{aligned} t\alpha &\subseteq \{t_0(t_1, \dots, t_k) \mid t_i \in \alpha(G_{\xi_i}) \text{ for } 0 \leq i \leq k\} \\ &\subseteq \{t_0(t_1, \dots, t_k) \mid t_i \in T_{\xi_i} \text{ for } 0 \leq i \leq k\} \\ &= \{Y(s_0)(Y(s_1), \dots, Y(s_k)) \mid s_i \in L(\gamma_{\xi_i}) \text{ for } 0 \leq i \leq k\} \\ &= \{Y(\text{subst}[s_0, s_1, \dots, s_k]) \mid s_i \in L(\gamma_{\xi_i}) \text{ for } 0 \leq i \leq k\} \\ &\subseteq T_\xi. \end{aligned}$$

‘ \supseteq ’ This time, the proof is by induction on the length of derivations. We show that $\xi \rightarrow^+ t$ implies $Y(t) \in T_{\mathcal{N}}(G_\xi)$ for all $\xi \in \Xi$ and $\mathbb{T}_{\Sigma_{Y,k}}$.

Derivations of length 1 in γ are of the form $\xi \rightarrow f_l = t$ for some $f^{(l)} \in \Sigma \cup X_k$. This yields

$$Y(t) = f[x_1, \dots, x_l] \in T_{\mathcal{N}}(G_\xi)$$

by case (a) in the construction of γ_ξ .

Derivations of length strictly greater than 1 are of the form

$$\xi \rightarrow \text{subst}[\xi_0, \xi_1, \dots, \xi_k] \rightarrow^+ \text{subst}[t_0, t_1, \dots, t_k] = t$$

for nonterminals $\xi_0, \xi_1, \dots, \xi_k \in \Xi$, and trees $t_0, t_1, \dots, t_k \in \mathbb{T}_{\Sigma_{Y,k}}$. Thus, using the induction hypothesis for t_0, t_1, \dots, t_k , we get³

$$\begin{aligned} Y(t) &= Y(t_0)(Y(t_1), \dots, Y(t_k)) \\ &\subseteq \left(G_{\xi_0}[\tilde{G}_{\xi_1}, \dots, \tilde{G}_{\xi_k}] \right) \alpha_{\mathcal{N}} \\ &\subseteq T_{\mathcal{N}}(G_\xi), \end{aligned}$$

which completes the proof. ■

³Recall that $\alpha_{\mathcal{N}}$ denotes the \mathcal{N} -assignment $\llbracket G_\xi \leftarrow T_{\mathcal{N}}(G_\xi) \mid \xi \in \Xi \rrbracket$.

For the other direction, we show that $Y(REG)$ contains all of $DEL(REG)$.

Lemma 6.4 $DEL(REG) \subseteq Y(REG)$.

Proof Let \mathcal{N} be a delegation network over REG , with primitives in some unsorted finite signature Σ , and let k be the maximum rank of all $G \in \mathcal{N}$. In fact, we may assume without loss of generality that all $G \in \mathcal{N}$ are of rank precisely k . Thus, every $G \in \mathcal{N}$ is of the form $G = (X_k, \gamma_G)$. We denote the set of nonterminals, the set of rules, and the initial nonterminal of γ_G by Ξ_G , R_G , and ξ_{ini}^G , resp. The signature of terminals of each γ_G is assumed to be $\Sigma_{\mathcal{N}} \cup X_k = \Sigma \cup X_k$, and the sets Ξ_G are assumed to be pairwise disjoint. Furthermore, we may again assume that R_G is in normal form, i.e., all rules in R_G are of the form $\xi \rightarrow f[\xi_1, \dots, \xi_l]$, where $f^{(l)} \in \Sigma_{\mathcal{N}} \cup X_k$ and $\xi, \xi_1, \dots, \xi_l \in \Xi_G$. Now, consider some $G_0 \in \mathcal{N}$. To show that $T_{\mathcal{N}}(G_0) \in Y(REG)$, let $\gamma = (\Xi, \Sigma_{Y,k}, R, \xi_{\text{ini}}^{G_0})$, where $\Xi = \bigcup_{G \in \mathcal{N}} \Xi_G$, and R is defined as follows.

- For every rule $\xi \rightarrow f[\xi_1, \dots, \xi_l]$ in R_G , where $G \in \mathcal{N}$ and $f \in \Sigma \cup X_k$, the rule

$$\xi \rightarrow \text{subst}[f_l, \xi_1, \dots, \xi_l, \underbrace{\perp, \dots, \perp}_{k-l}]$$

is in R . Here, \perp is an arbitrary symbol of rank 0 in $\Sigma_{Y,k}$ (used only for padding; note that $Y(\text{subst}[f_l, t_1, \dots, t_l, \perp, \dots, \perp]) = f[Y(t_1), \dots, Y(t_l)]$).

- For every rule $\xi \rightarrow G'[\xi_1, \dots, \xi_k]$ in R_G , where $G' \in \mathcal{N}$, the rule

$$\xi \rightarrow \text{subst}[\xi_{\text{ini}}^{G'}, \xi_1, \dots, \xi_k]$$

is in R .

To continue with the proof, let

$$T_{\xi} = \{t \in T_{\Sigma_{Y,k}} \mid \xi \rightarrow_{\gamma}^+ t\} \quad \text{and} \quad T_{\xi}^G = \{t \in T_{\Sigma_{\mathcal{N}} \cup X_k} \mid \xi \rightarrow_{\gamma_G}^+ t\}$$

for every $G \in \mathcal{N}$ and $\xi \in \Xi_G$. In particular, $T_{\xi_{\text{ini}}^{G_0}} = L(\gamma)$. To finish the proof, let $G \in \mathcal{N}$. We show that $T_{\mathcal{N}}(G) = Y(T_{\xi_{\text{ini}}^G})$.

‘ \subseteq ’ Again, we proceed by induction, using Lemma 5.8. Thus, let α be an \mathcal{N} -assignment such that $\alpha(G') \subseteq Y(T_{\xi_{\text{ini}}^{G'}})$ for all $G' \in \mathcal{N}$. We must show that $t\alpha \subseteq Y(T_{\xi_{\text{ini}}^G})$ for all $t \in L(\gamma_G)$. In fact, we shall prove a more general statement:

$$\text{For all } \xi \in \Xi_G \text{ and } t \in T_{\xi}^G, t\alpha \subseteq Y(T_{\xi}).$$

We proceed by induction on the length of derivations $\xi \rightarrow_{\gamma_G}^+ t$. Thus, let $\xi \rightarrow F[\xi_1, \dots, \xi_l] \rightarrow^+ F[t_1, \dots, t_l] = t$, and assume that the statement holds for the subderivations $\xi_i \rightarrow^+ t_i$. There are two cases.

Case 1 $F \in \Sigma \cup X_k$.

In this case,

$$\begin{aligned} t\alpha &= \{F[t'_1, \dots, t'_l] \mid t'_i \in t_i\alpha \text{ for } i \in [l]\} \\ &\subseteq \{F[t'_1, \dots, t'_l] \mid t'_i \in Y(T_{\xi_i}) \text{ for } i \in [l]\} \\ &= Y(\{\text{subst}[F_l, t''_1, \dots, t''_l, \perp, \dots, \perp] \mid t''_i \in T_{\xi_i} \text{ for } i \in [l]\}) \\ &\subseteq Y(T_{\xi}), \end{aligned}$$

since $\xi \rightarrow \text{subst}[F_l, \xi_1, \dots, \xi_l, \perp, \dots, \perp]$ is a rule in R .

Case 2 $F \in \mathcal{N}$ (and, thus, $l = k$).

In this situation,

$$\begin{aligned} t\alpha &= \{s(t'_1, \dots, t'_k) \mid s \in \alpha(F) \text{ and } t'_i \in t_i\alpha \text{ for } i \in [k]\} \\ &\subseteq \{s(t'_1, \dots, t'_k) \mid s \in Y(T_{\xi_{\text{ini}}^F}) \text{ and } t'_i \in Y(T_{\xi_i}) \text{ for } i \in [k]\} \\ &= Y(\{subst[s', t''_1, \dots, t''_k] \mid s' \in T_{\xi_{\text{ini}}^F} \text{ and } t''_i \in T_{\xi_i} \text{ for } i \in [k]\}) \\ &\subseteq Y(T_\xi), \end{aligned}$$

where we, in the last line, use the fact that the rule $\xi \rightarrow subst[\xi_{\text{ini}}^F, \xi_1, \dots, \xi_k]$ is in R .

‘ \supseteq ’ Again, we prove a statement slightly more general than the desired inclusion $Y(T_{\xi_{\text{ini}}^G}) \subseteq T_{\mathcal{N}}(G)$:

$$\text{For all } \xi \in \Xi_G \text{ and } t \in T_\xi, Y(t) \in T_\xi^G \alpha_{\mathcal{N}}.$$

Thus, in particular, $Y(T_{\xi_{\text{ini}}^G}) \subseteq T_{\xi_{\text{ini}}^G}^G \alpha_{\mathcal{N}} = T_{\mathcal{N}}(G)$.

Similar to what was done above, we proceed by induction on the length of derivations $\xi \rightarrow_\gamma^+ t$, assuming that the statement holds for its subderivations. Once more, we distinguish the two possible cases.

Case 1 The derivation has the form

$$\xi \rightarrow subst[F_l, \xi_1, \dots, \xi_l, \perp, \dots, \perp] \rightarrow^+ subst[F_l, t_1, \dots, t_l, \perp, \dots, \perp] = t,$$

where $F \in \Sigma \cup X_k$.

In this case, R_G contains the rule $\xi \rightarrow F[\xi_1, \dots, \xi_l]$, and the induction hypothesis yields $Y(t_i) \in T_{\xi_i}^G \alpha_{\mathcal{N}}$ for all $i \in [l]$. Thus, we obtain

$$\begin{aligned} Y(t) &= F[Y(t_1), \dots, Y(t_l)] \\ &\in \{F[t'_1, \dots, t'_l] \mid t'_i \in T_{\xi_i}^G \alpha_{\mathcal{N}} \text{ for } i \in [l]\} \\ &= \{F[t''_1, \dots, t''_l] \alpha_{\mathcal{N}} \mid t''_i \in T_{\xi_i}^G \text{ for } i \in [l]\} \\ &= \{F[t''_1, \dots, t''_l] \mid t''_i \in T_{\xi_i}^G \text{ for } i \in [l]\} \alpha_{\mathcal{N}} \\ &\subseteq T_\xi^G \alpha_{\mathcal{N}}. \end{aligned}$$

Case 2 The derivation has the form

$$\xi \rightarrow subst[\xi_{\text{ini}}^F, \xi_1, \dots, \xi_k] \rightarrow^+ subst[t_0, t_1, \dots, t_k] = t$$

for some $F \in \mathcal{N}$.

By construction, this implies that R_G contains the rule $\xi \rightarrow F[\xi_1, \dots, \xi_k]$. Consequently, we obtain

$$\begin{aligned} Y(t) &= Y(t_0)(Y(t_1), \dots, Y(t_k)) \\ &\in \{s(t'_1, \dots, t'_k) \mid s \in T_{\xi_{\text{ini}}^F} \alpha_{\mathcal{N}} \text{ and } t'_i \in T_{\xi_i}^G \alpha_{\mathcal{N}} \text{ for } i \in [k]\} \\ &= \{s(t'_1, \dots, t'_k) \mid s \in T_{\mathcal{N}}(F) \text{ and } t'_i \in T_{\xi_i}^G \alpha_{\mathcal{N}} \text{ for } i \in [k]\} \\ &= \{F[t''_1, \dots, t''_k] \alpha_{\mathcal{N}} \mid t''_i \in T_{\xi_i}^G\} \\ &= \{F[t''_1, \dots, t''_k] \mid t''_i \in T_{\xi_i}^G\} \alpha_{\mathcal{N}} \\ &\subseteq T_\xi^G \alpha_{\mathcal{N}}. \end{aligned}$$

This completes the proof. \blacksquare

Lemmas 6.3 and 6.3 immediately yield the main result of this paper.

Theorem 6.5 $DEL(REG) = Y(REG) = DEL(FIN)$.

7 Conclusions

As mentioned in the introduction and proved in [ES77], $Y(REG)$ is the set of IO-context-free tree languages. Thus, by Theorem 6.5, the IO-context-free tree languages coincide with $DEL(REG) = DEL(FIN)$. Another well-known result is that the regular tree languages are the unique solutions of finite systems of tree language equations. Here, an equation has the form $x = t_1 + \dots + t_n$ with $x \in X$ and $t_1, \dots, t_n \in T_{\Sigma \cup X}$, where X is a set of variables ranging over tree languages, and $+$ is interpreted as union of tree languages. Obviously, this result is closely related to the equality $DEL(REG) = DEL(FIN)$. Intuitively, a delegating generator G whose tree generator component is a regular tree grammar γ can be seen as an infinite equation $x = t_1 + t_2 + \dots$, where $\{t_1, t_2, \dots\} = L(\gamma)$. By the mentioned result, this infinite equation can be replaced with a finite number of finite ones. This makes it possible to replace G with a finite number of delegating generators which, together, fulfil the same purpose as G .

Using Theorem 5.9, we obtain the following corollary as an immediate consequence.

Corollary 7.1 For every function $\varphi: \mathbb{A}_1 \times \dots \times \mathbb{A}_k \rightarrow \mathbb{A}$, the following statements are equivalent:

1. There is a delegation network \mathcal{N} over REG such that $\varphi = \pi_{\mathcal{N}}(G)$ for some $G \in \mathcal{N}$.
2. There is a delegation network \mathcal{N} over FIN such that $\varphi = \pi_{\mathcal{N}}(G)$ for some $G \in \mathcal{N}$.
3. There is an IO-context-free tree language L such that $\varphi = \text{val}_{\pi}(L)$.

It is a rather obvious question to ask what happens if DEL is iterated. More precisely, for every class C of tree languages, let $DEL^n(C)$ be defined in the straightforward way, i.e., $DEL^0(C) = C$ and $DEL^{n+1}(C) = DEL(DEL^n(C))$. Looking at Theorem 6.5, one may be tempted to conjecture that the resulting hierarchy $(DEL^n(REG))_{n \in \mathbb{N}}$, which we may call the delegation hierarchy, is the same as the so-called IO-hierarchy $(Y^n(REG))_{n \in \mathbb{N}}$.⁴ As shown by Damm [Dam82], the latter is strict at each level. However, the mentioned conjecture is wrong. In fact, $DEL^2(REG)$ is not even contained in $MTT^*(REG)$, where MTT^* denotes the composition closure of total deterministic macro tree transducers [EV85].

Theorem 7.2 $DEL^2(REG) \not\subseteq MTT^*(REG)$

Proof Recall that the *yield* of a tree t , denoted by $\text{yield}(t)$, is the string obtained by reading its leaves from left to right. Let $T_0 \in REG$ be the set

⁴where, of course, $Y^0(REG) = REG$ and $Y^{n+1}(REG) = Y(Y^n(REG))$

of all trees t over $\{f^{(2)}, a^{(0)}, b^{(0)}, 0^{(0)}, 1^{(0)}\}$ such that $yield(t) = wbw'$ for some $w, w' \in \{a, 0, 1\}^*$. It is an easy exercise to show that $DEL^2(REG)$ contains the tree language T consisting of all trees of the form $s(t_1, \dots, t_{2^n})$ such that s is a fully balanced binary tree of height n over $\{f^{(2)}\} \cup X_{2^n}$, and $t_1, \dots, t_{2^n} \in T_0$. However, by the results of [EM02], T is not an element of $MTT^*(REG)$. The proof of this fact is similar to (the second half of) the proof of Theorem 25 of [EM02]. It reads as follows.

For a set A of symbols and a string language L , let $rub_A(L)$ denote the set of all strings of the form $w_0a_1w_1 \cdots a_nw_n$ such that $w_0, \dots, w_n \in A^*$ and $a_1 \cdots a_n \in L$. Intuitively, rub_A inserts “rubbish”, arbitrary strings in A^* , between the symbols of strings in L . Clearly, $yield(T) = rub_{\{0,1,a\}}(L_{2^n}) = rub_{\{0,1\}}(rub_{\{a\}}(L_{2^n}))$, where $L_{2^n} = \{b^{2^n} \mid n \in \mathbb{N}\}$. Therefore, by Theorem 18 of [EM02], $yield(T) \in yield(MTT^*(REG))$ would imply that $rub_{\{a\}}(L_{2^n}) \in yield(dTD(REG))$, where dTD denotes the set of all deterministic top-down tree transformations. Now, by Theorem 3.2.14 of [ERS80], the latter would imply that $L_{2^n} \in yield(TD_{fc}(REG))$, denoting by TD_{fc} the set of all finite-copying top-down tree transformations. However, by Corollary 3.2.7 of [ERS80], this could only be the case if the Parikh image of L_{2^n} was semilinear, which it is not. Hence, $yield(T) \notin yield(MTT^*(REG))$, and thus $T \notin MTT^*(REG)$. ■

Characterizing the delegation hierarchy could be an interesting subject for future work.

Acknowledgment I thank M. Berglund for valuable discussions on the subject of this paper, and J. Engelfriet for refreshing my memories regarding [EM02].

References

- [Dam82] Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–208, 1982.
- [Dre06] Frank Drewes. *Grammatical Picture Generation – A Tree-Based Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- [EM02] Joost Engelfriet and Sebastian Maneth. Output string languages of compositions of macro tree transducers. *Journal of Computer and System Sciences*, 64:350–395, 2002.
- [ERS80] Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20:150–202, 1980.
- [ES77] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. I. *Journal of Computer and System Sciences*, 15:328–353, 1977.
- [ES78] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. II. *Journal of Computer and System Sciences*, 16:67–99, 1978.

- [EV85] Joost Engelfriet and Heiko Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.
- [FV98] Zoltán Fülöp and Heiko Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer, Berlin, Heidelberg, 1998.
- [Mai74] Tom S. E. Maibaum. A generalized approach to formal languages. *Journal of Computer and System Sciences*, 8:409–502, 1974.
- [MW67] Jorge Mezei and Jesse B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11:3–29, 1967.
- [PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, New York, 1990.