From Tree-Based Generators to Delegation Networks

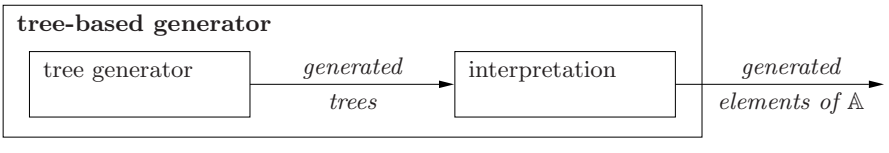
Frank Drewes

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

Abstract. The first part of this paper is a brief survey on tree-based generators, including some typical examples taken from the fields of string, tree, graph, and picture generation. In the second part, an extension of the tree-based generator called delegation network is proposed. Intuitively, a delegation network is a network of tree-based generators that can "delegate" subtasks to each other. In this way, different types of tree-based generators can be combined to generate complex objects.

1 Introduction

The theory of tree languages and tree transformations is an important and lively field of theoretical computer science [GS84, NP92, GS97, FV98, CDG+02]. It is concerned with formal devices that generate, recognize or transform trees. A tree in this sense is a term, i.e., a formal expression composed of abstract operation symbols. The usefulness of devices dealing with such trees is to a large extent based on the fact that trees can be interpreted by choosing a domain $\mathbb{A}$ and associating an operation on $\mathbb{A}$ (of the appropriate arity) with each symbol. Thus, given such an interpretation, also called algebra, every tree denotes an element of $\mathbb{A}$. This means that a device generating trees provides the syntactic basis for a *tree-based generator* – a system consisting of the tree generator and an interpretation, that generates elements of $\mathbb{A}$:



Similarly, a device that transforms trees, together with two interpretations, can be used to compute a function from $\mathbb{A}$ to $\mathbb{B}$. The tree transformation can then be seen as a symbolic algorithm [Eng80].

In this paper, we will focus on generation rather than transformation. We will first recall the formal notions needed for the definition of tree-based generators, and give some examples from the areas of string, tree, graph, and picture generation. Afterwards, a generalization called delegation network is proposed

S. Bozapalidis and G. Rahonis (Eds.): CAI 2007, LNCS 4728, pp. 48–72, 2007.

and illustrated by means of an example. A first attempt to formalize the notion of delegation networks, and to prove some of their basic properties, was made in [Dre07]. However, as pointed out by Engelfriet[1], the evaluation of trees with respect to nondeterministic algebras is not defined correctly in this paper, and not all results (in particular the "Mezei&Wright-like" Theorem 5.9) hold for the nondeterministic case. Therefore, the second part of the present paper tries to formalize the notion of delegation networks in a more appropriate way, thus laying the basis for future work in this area.

The purpose of delegation networks is to be able to combine several tree-based generators (possibly of different types) with each other. Intuitively, a delegation network consists of a finite number of tree-based generators that may "delegate" parts of the generation process to each other. Roughly speaking, every delegating generator in the network is associated with a symbol $\mathtt{g}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \to \mathtt{A}$ from a certain signature. The semantics of the delegation network interprets $\mathtt{g}$ as a function $g\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \to \mathbb{A}$.[2] Delegation means that another generator from the network, associated with a symbol $\mathtt{g}'$, can generate trees that contain occurrences of $\mathtt{g}$. If such a tree is evaluated, $\mathtt{g}$ is interpreted as g. Thus, the interpretation g' of $\mathtt{g}'$ depends on g – and as delegation networks can be cyclic, g may also depend on g'. For this reason, we choose a least fixed-point semantics for delegation networks (Definition 5).

The structure of this paper is thus as follows. In the next section, the basic notions regarding tree-based generators are recalled. In Section 3, the tree-based versions of some well-known grammatical devices are discussed. Section 4 introduces delegation networks, which are illustrated by means of an example in Section 5. Section 6 concludes the paper.

2 Tree-Based Generators

Before recalling the notion of tree-based generators, let us summarize some standard notions and notation. Throughout this paper, $\mathbb{N}$ denotes the set of natural numbers (including zero). For $n \in \mathbb{N}$, the set $\{1, \ldots, n\}$ is denoted by $[n]$. The powerset of a set $\mathtt{A}$ is denoted by $\wp(\mathtt{A})$. A function f of arity 0 is identified with the constant $f()$.

2.1 Signatures and Trees

Let S be a set of sorts. An *S-sorted signature* (or just *signature*) is a finite set Σ of symbols $\mathtt{f}$, each of which has an associated *profile* $\mathtt{A}_1 \times \cdots \times \mathtt{A}_k \to \mathtt{A}$, where $k \in \mathbb{N}$ and $\mathtt{A}_1, \ldots, \mathtt{A}_k, \mathtt{A} \in S$. To indicate that $\mathtt{f}$ has this profile, we also write $\mathtt{f}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \to \mathtt{A}$. The number k is also called the *rank* of $\mathtt{f}$. Symbols of rank 0 are called *constant symbols*. We write the profile of a constant symbol without the arrow, i.e., $\mathtt{a}\colon \mathtt{A}$, thus saying that its profile is $\mathtt{A}$.

[1] Personal communication.

[2] Actually, these are nondeterministic rather than ordinary functions, but this is not important for the moment.

Given a sort $\mathtt{A}$, the set of all *trees of sort* $\mathtt{A}$ *over* Σ is denoted by $\mathrm{T}_\Sigma^{\mathtt{A}}$. By definition, the sets $\mathrm{T}_\Sigma^{\mathtt{A}}$ are the smallest sets of strings simultaneously satisfying the following conditions:

- For every $\mathtt{a}\colon \mathtt{A}$ in Σ, the string $\mathtt{a}$ is in $\mathrm{T}_\Sigma^{\mathtt{A}}$.
- For all $\mathtt{f}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \to \mathtt{A}$ in Σ ($k > 0$) and all $t_1 \in \mathrm{T}_\Sigma^{\mathtt{A}_1}, \ldots, t_k \in \mathrm{T}_\Sigma^{\mathtt{A}_k}$, the string $\mathtt{f}[t_1, \ldots, t_k]$ is in $\mathrm{T}_\Sigma^{\mathtt{A}}$. (Here, the square brackets and the comma are assumed to be special symbols not in Σ.)

The set T_Σ of all trees over Σ is given by $\mathrm{T}_\Sigma = \bigcup_{\mathtt{A} \in S} \mathrm{T}_\Sigma^{\mathtt{A}}$.

2.2 Algebras and Evaluation

Given an S-sorted signature Σ, a *Σ-algebra* (or just *algebra*) is a pair $\mathcal{A} = (dom, \sigma)$. Here, *dom* is a *domain mapping for S* – a function assigning to every sort $\mathtt{A} \in S$ a set $dom(\mathtt{A})$ called its domain, and σ is the *interpretation of symbols* – a function that assigns to every function symbol $\mathtt{f}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \to \mathtt{A}$ in Σ a corresponding function $\sigma(\mathtt{f})\colon dom(\mathtt{A}_1) \times \cdots \times dom(\mathtt{A}_k) \to dom(\mathtt{A})$. The function σ is also called a *(Σ, dom)-interpretation*.

Throughout the rest of the paper, we will use the following typographical conventions in connection with algebras, unless there is a risk of confusion: the domain $dom(\mathtt{A})$ assigned to a sort $\mathtt{A}$ is denoted by $\mathbb{A}$, and the interpretation of a symbol $\mathtt{f}$ is denoted by f. Using these conventions, defining an algebra means to associate a domain $\mathbb{A}$ with every sort $\mathtt{A}$, and a function $f\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \to \mathbb{A}$ with every function symbol $\mathtt{f}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \to \mathtt{A}$ in Σ.

Given a Σ-algebra $\mathcal{A}$, trees over Σ can be viewed as expressions that can be evaluated. This evaluation, denoted by $val_{\mathcal{A}}$, is defined recursively, as one would expect: $val_{\mathcal{A}}(t) = f(val_{\mathcal{A}}(t_1), \ldots, val_{\mathcal{A}}(t_k))$ for every tree $t = \mathtt{f}[t_1, \ldots, t_k] \in \mathrm{T}_\Sigma$. Note that the definition of T_Σ makes sure that $val_{\mathcal{A}}(t)$ is well defined. In the following, we may write $val(t)$ if $\mathcal{A}$ is understood.

Especially in examples, we shall frequently work with algebras over *unsorted signatures*, which is an S-sorted signature such that S is a singleton $\{\mathtt{A}\}$. In this case, the notation $\mathtt{f}\colon \mathtt{A}^k \to \mathtt{A}$ may be abbreviated as $\mathtt{f}^{(k)}$. The (unique) domain of an algebra $\mathcal{A}$ over Σ is denoted by $dom(\mathcal{A})$.

2.3 Tree-Based Generators

A *tree language* is a subset of $\mathrm{T}_\Sigma^{\mathtt{A}}$, for some S-sorted signature Σ and a sort $\mathtt{A} \in S$. A formal device γ defining a tree language $L(\gamma)$ is called a *tree generator*. If $L(\gamma) \subseteq \mathrm{T}_\Sigma$, then Σ is called the *output signature* of γ. (Thus, to be picky, one should rather speak of *an* output signature of γ.)

Now, a *tree-based generator* is a pair $G = (\gamma, \mathcal{A})$ that consists of a tree generator γ and a Σ-algebra $\mathcal{A}$, where Σ is the output signature of γ. The *language generated by* G is given by $L(G) = \{val(t) \mid t \in L(\gamma)\}$.

2.4 Treebag

One of the advantages of the notion of tree-based generators is that it gives rise to a flexible implementation in a rather straightforward manner. The system Treebag [Dre06, Chapter 8] is such a system. It allows its user to interactively load instances of a variety of tree generators (and tree transducers), algebras, and so-called displays, and to establish input-output relations between them. In this way, tree-based generators can be assembled, and displays that show the resulting objects on the screen can be attached to them. The pictures in Sections 3.4 and 3.5 have been created in this way. Moreover, the delegation network discussed in Section 5 has been simulated in Treebag in order to create the pictures shown.

3 Examples of Tree-Based Generators

We shall now discuss a few typical classes of tree-based generators. All of them are based on tree generators with unsorted output signatures; sorted signatures will become important in the next section.

Let us start with one of the simplest meaningful cases: the tree-based version of string grammars.

3.1 String Generation

Let T be a set of symbols. We denote the set of all strings over T by T^*, and the empty string by ε.

Now, consider an unsorted signature Σ. The Σ-algebra $\mathcal{A}_{\Sigma,T}$, which allows to assemble strings by means of concatenation, is given by

- $dom(\mathcal{A}_{\Sigma,T}) = T^*$,
- $a = \mathtt{a}$ for every constant symbol in Σ which belongs to T, and
- $f(u_1, \ldots, u_k) = u_1 \cdots u_k$ for all other symbols $\mathtt{f}^{(k)} \in \Sigma$ and all strings $u_1, \ldots, u_k \in T^*$. Thus, all symbols not in T are interpreted as concatenation operators of the appropriate arity. In particular, $f = \varepsilon$ for all $\mathtt{f}^{(0)} \in \Sigma$ which are not in T.

By definition, a tree-based generator of the form $G = (\gamma, \mathcal{A}_{\Sigma,T})$ generates a string language $L(G) \subseteq T^*$. Several types of string grammars well known from traditional string language theory can be formulated in this way. In fact, historically, this was one of the motivations for developing a theory of tree languages and tree transformations (see, e.g., [Rou70, Tha73]).

As one of the easiest examples, let us see how the context-free grammar can be turned into a tree-based generator. For this, we use a tree generator known as regular tree grammar, which is defined as follows. (As a minor extension of the usual definition found in the literature, we define regular tree grammars generating trees over sorted signatures.)

Definition 1 (regular tree grammar). *Let S be a set of sorts. A* regular tree grammar *is a tuple $\gamma = (\Xi, \Sigma, R, \xi_{\mathrm{ini}})$, where*

- *Ξ is an S-sorted signature of constant symbols called* nonterminals*,*
- *Σ is an S-sorted signature of* output symbols *which is disjoint with Ξ,*
- *R is a finite set of* rules *$\xi \to r$, where $\xi \in \Xi$ and $r \in \mathrm{T}_{\Sigma\cup\Xi}$ are of the same sort, and*
- *$\xi_{\mathrm{ini}} \in \Xi$ is the* initial nonterminal*.*

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

$$L(\gamma) = \{t \in \mathrm{T}_\Sigma \mid \xi_{\mathrm{ini}} \to^* t\},$$

where $\to^*$ denotes the transitive and reflexive closure of the relation $\to$.

Now, let $G = (\Xi, T, R, \xi_{\mathrm{ini}})$ be a context-free grammar consisting, as usual, of finite sets Ξ and T of nonterminal and terminal symbols, resp., a set R of context-free rules, and an initial nonterminal $\xi_{\mathrm{ini}} \in \Xi$. We turn G into an equivalent tree-based generator, as follows.

First, we need a suitable unsorted signature Σ. For each rule $r = (\xi \to u)$ in R, let Σ contain a unique symbol $\mathtt{r} \notin T$ of rank $|u|$ (where $|u|$ denotes the length of u). In addition, Σ contains all symbols in T as symbols of rank 0.

Second, let $\gamma = (\Xi, \Sigma, R', \xi_{\mathrm{ini}})$ be the regular tree grammar obtained by turning every rule $(r = \xi \to \mathtt{a}_1 \cdots \mathtt{a}_k)$ in R (where $\mathtt{a}_1, \ldots, \mathtt{a}_k \in T$) into a rule $\xi \to \mathtt{r}[\mathtt{a}_1, \ldots, \mathtt{a}_k]$ in R'. It is an easy exercise to show that the tree-based generator $G' = (\gamma, \mathcal{A}_{\Sigma,T})$ satisfies $L(G') = L(G)$. Every tree t generated by γ', where $val(t) = u$, corresponds to an abstract syntax tree of u with respect to G.

Of course, there are several natural ways to choose Σ in the previous construction. For example, instead of including $\mathtt{r}$ for every rule $r = (\xi \to u)$, one could choose a symbol $\xi_{[k]}$ of rank $k = |u|$. Then, the trees generated by γ would correspond to the derivation trees of G. Another possibility is to include only two symbols in addition to the symbols in T, namely $\circ^{(2)}$ and $\epsilon^{(0)}$, and to use rules of the form $\xi \to \circ[\mathtt{a}_1, \circ[\cdots, \circ[\mathtt{a}_k, \epsilon]\cdots]]$ in R'.

It is also easy to show that the construction can be reversed: for every tree-based generator of the form $(\gamma, \mathcal{A}_{\Sigma,T})$, where γ is a regular tree grammar, there is a context-free grammar generating the same language. Thus, a characterization of the class of context-free languages in terms of languages generated by tree-based generators has been obtained.

3.2 Tree Generation

Trivially, trees can be generated by tree-based generators. For this, just use a tree-based generator $G = (\gamma, \mathcal{A})$, where $\mathcal{A}$ is the free term algebra over the output signature of γ. In this algebra, the interpretation of symbols is given by

$f(t_1, \ldots, t_k) = \mathtt{f}[t_1, \ldots, t_k]$, which means that *val* is the identity on T_Σ, and $L(G) = L(\gamma)$.

The situation becomes more interesting if $\mathcal{A}$ is not as simple as the free term algebra. An important example for this is the so-called YIELD algebra (or YIELD mapping, see [Mai74, Eng80, ES77, ES78, FV98, Dre06]), which formalizes the construction of trees using variable substitution. We will only discuss the variant dealing with trees over an unsorted signature. The extension to arbitrary S-sorted signatures is straightforward, but technical.

Let $X = \{x_1, x_2, \ldots\}$ be a countably infinite set of special symbols of rank 0, called variables. For $l \in \mathbb{N}$, we let X_l denote $\{x_1, \ldots, x_l\}$. For $t, t_1, \ldots, t_l \in \mathrm{T}_{\Sigma \cup X}$,[3] we let $t[\![t_1, \ldots, t_l]\!]$ denote the tree obtained from t by simultaneously replacing all occurrences of variables x_i by t_i ($i \in [l]$).

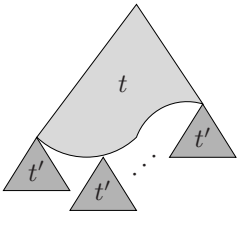
Definition 2 (YIELD algebra). *Let Σ be an unsorted signature and let $l \geq \max\{k \in \mathbb{N} \mid \mathtt{f}^{(k)} \in \Sigma\}$. The (unsorted)* derived signature *$\Sigma_{\mathrm{Y},l}$ is given by*

$$\Sigma_{\mathrm{Y},l} = \{\mathtt{subst}^{(l+1)}\} \cup \{\mathtt{f}^{\langle k \rangle^{(0)}} \mid \mathtt{f}^{(k)} \in \Sigma \cup X_l\}.$$

The YIELD algebra *(with respect to Σ and l) is the $\Sigma_{\mathrm{Y},l}$-algebra $\mathcal{Y}$ such that*

- *$dom(\mathcal{Y}) = \mathrm{T}_{\Sigma \cup X}$,*
- *$subst(t, t_1, \ldots, t_l) = t[\![t_1, \ldots, t_l]\!]$ for all $t, t_1 \ldots, t_l \in \mathrm{T}_{\Sigma \cup X}$, and*
- *$f^{\langle k \rangle} = f[x_1, \ldots, x_k]$ for all $\mathtt{f}^{(k)} \in \Sigma \cup X_l$.*

Tree-based generators of the form $G = (\gamma, \mathcal{Y})$, where γ is a regular tree grammar, generate the so-called IO context-free tree languages. As an example, let $\Sigma = \{\mathtt{f}^{(2)}, \mathtt{g}^{(2)}, \mathtt{a}^{(0)}\}$. We show how to generate the set of all trees of the form $t[\![t']\!]$, where t is an arbitrary tree over $\{\mathtt{f}^{(2)}, x_1^{(0)}\}$, and t' is a totally balanced tree over $\{\mathtt{g}^{(2)}, \mathtt{a}^{(0)}\}$:



For this, use the regular tree grammar $\gamma = (\{\xi_{\mathrm{ini}}, \xi_{\mathrm{arb}}, \xi_{\mathrm{bal}}\}, \Sigma_{\mathrm{Y},2}, R, \xi_{\mathrm{ini}})$, where R is given as in Table 1. In an IO context-free tree grammar, these rules would be written as

$$\begin{aligned} \xi_{\mathrm{ini}} &\to \xi_{\mathrm{arb}}[\xi_{\mathrm{bal}}[\mathtt{a}]], \\ \xi_{\mathrm{arb}}[x_1] &\to \mathtt{f}[\xi_{\mathrm{arb}}[x_1], \xi_{\mathrm{arb}}[x_1]] \mid x_1, \\ \xi_{\mathrm{bal}}[x_1] &\to \xi_{\mathrm{bal}}[\mathtt{f}[x_1, x_1]] \mid x_1. \end{aligned}$$

[3] We use the notation $\mathrm{T}_{\Sigma \cup X}$ to abbreviate $\bigcup_{l \in \mathbb{N}} \mathrm{T}_{\Sigma \cup \{x_1, \ldots, x_l\}}$.

Table 1. Rules of a regular tree grammar that, in connection with the YIELD algebra $\mathcal{Y}$, mimics an IO context-free tree grammar, where '|' is used to separate alternatives. We choose $l = 2$ in Definition 2; hence, `subst` is of rank 3. For simplicity, we write $\mathtt{subst}[t, t_1]$ if the third subtree is uninteresting (i.e., would never be used). The right column shows the rules in a partially evaluated form, obtained by recursively replacing all subterms $t = \mathtt{subst}[\mathtt{h}^{\langle k\rangle}, t_1, \ldots, t_k]$ with $t' = \mathtt{h}[t'_1, \ldots, t'_k]$ (and all $\mathtt{h}^{\langle 0\rangle}$ with their values h).

Actual rules	Simplified form
$\xi_{\mathrm{ini}} \to \mathtt{subst}[\xi_{\mathrm{arb}}, \mathtt{subst}[\xi_{\mathrm{bal}}, \mathtt{a}^{\langle 0\rangle}]]$	$\xi_{\mathrm{ini}} \to \mathtt{subst}[\xi_{\mathrm{arb}}, \mathtt{subst}[\xi_{\mathrm{bal}}, \mathtt{a}]]$
$\xi_{\mathrm{arb}} \to \mathtt{subst}[\mathtt{f}^{\langle 2\rangle}, \xi_{\mathrm{arb}}, \xi_{\mathrm{arb}}] \mid x_1^{\langle 0\rangle}$	$\xi_{\mathrm{arb}} \to \mathtt{f}[\xi_{\mathrm{arb}}, \xi_{\mathrm{arb}}] \mid x_1$
$\xi_{\mathrm{bal}} \to \mathtt{subst}[\xi_{\mathrm{bal}}, \mathtt{subst}[\mathtt{f}^{\langle 2\rangle}, x_1^{\langle 0\rangle}, x_1^{\langle 0\rangle}]] \mid x_1^{\langle 0\rangle}$	$\xi_{\mathrm{bal}} \to \mathtt{subst}[\xi_{\mathrm{bal}}, \mathtt{f}[x_1, x_1]] \mid x_1$

3.3 Graph Generation

The tree-based perspective has turned out to be very fruitful in the area of context-free graph grammars. The first paper investigating the generation of context-free graph languages in a tree-based manner was [BC87]; see also the surveys [Cou90, Eng97].

There are two major types of context-free graph languages: those generated by *hyperedge replacement* (HR) and those generated by *node replacement* (NR). Both have equivalent formulations in terms of tree-based generators, where the underlying tree generator is the regular tree grammar. Thus, only the algebras differ. Here, we will consider the HR case.

For simplicity, we restrict ourselves to directed unlabelled graphs. For technical reasons, these graphs are equipped with a number of distinguished nodes, so-called ports. More precisely, a *graph* is a quadruple $H = (V, E, att, port)$ consisting of

- finite sets V and E of *nodes* and *edges*, resp.,
- a mapping $att\colon E \to V^2$ assigning to every edge its attached nodes (i.e., $att(e) = (v, v')$ means that e points from v to v'), and
- a partial mapping $port\colon \mathbb{N} \to V$, the *port labelling*, which is defined on a finite subset $df(port)$ of $\mathbb{N}$.

For $i \in df(port)$, the node $port(i)$ is called the i-port of H. Note that graphs whose port labelling is the totally undefined function can be considered as ordinary graphs without ports. If a graph is of this kind, we may omit the fourth component.

We consider the following two types of operations on graphs.[4]

1. Let $H = (V, E, att, port)$ and $H' = (V', E', att', port')$ be graphs and assume, without loss of generality, that $V \cap V' = \emptyset = E \cap E'$ (otherwise, take

[4] Strictly speaking, these operations work on abstract graphs (i.e., isomorphism classes of graphs) rather than concrete ones. Intuitively, this means that isomorphic graphs are considered to be the same.

isomorphic copies). Let $\equiv$ be the equivalence relation on $V \cup V'$ generated by $\{(port(i), port'(i)) \mid i \in df(port) \cap df(port')\}$. Then the *parallel composition* $par(H, H')$ of H and H' is obtained by taking their union and identifying ports with the same label. Formally,

$$par(H, H') = (V'', E \cup E', att'', port''),$$

where

(a) $V'' = \{[v]_\equiv \mid v \in V \cup V'\}$ is the set of equivalence classes of $V \cup V'$ with respect to $\equiv$,
(b) $att''(e) = ([v]_\equiv, [v']_\equiv)$ for $e \in E$ with $att(e) = (v, v')$ or $e \in E'$ with $att'(e) = (v, v')$, and
(c) for all $i \in df(port) \cup df(port')$,

$$port''(i) = \begin{cases} [port(i)]_\equiv & \text{if } i \in df(port) \\ [port'(i)]_\equiv & \text{otherwise.} \end{cases}$$

2. Given a partial function $\rho\colon \mathbb{N} \to \mathbb{N}$ which is defined on a finite subset $df(\rho)$ of $\mathbb{N}$, the *port relabelling* of a graph $H = (V, E, att, port)$ with respect to ρ is given by $rel_\rho(H) = (V, E, att, port \circ \rho)$. Here, $port \circ \rho$ denotes the composition of partial functions, i.e., $(port \circ \rho)(i)$ is defined if $i \in df(\rho)$ and $\rho(i) \in df(port)$, and yields $port(\rho(i))$ in this case.

Now, for an unsorted signature Σ, a Σ-algebra $\mathcal{A}$ is an *HR Σ-algebra* if it has as its domain the set of all graphs, and Σ contains, in addition to a finite number of constant symbols,

– the symbol $\mathtt{par}^{(2)}$, which is interpreted as *par*, and
– finitely many symbols of the form $\mathtt{rel}_\rho^{(1)}$ (where ρ is as above), each of which is interpreted as rel_ρ.

Remark. For readers who wonder about the definition of NR algebras, it can be mentioned that the graphs in NR algebras are allowed to contain any number of distinct nodes with the same port label, i.e., *port* is turned into a binary relation rather than a partial function. Moreover, let $ports_i(H)$ denote the set of all i-ports of such a graph H. Instead of the operation *par*, NR algebras contain binary operations $connect_C$, where C is a binary relation on port labels. For graphs H, H', $connect_C(H, H')$ is obtained by taking their disjoint union and adding, for all $(i, j) \in C$ and $(v, v') \in ports_i(H) \times ports_j(H') \cup ports_i(H') \times ports_j(H)$, an edge from v to v'. Thus, this operation connects the disjoint components H and H' with edges according to C.

As an example, we consider an HR context-free graph grammar (i.e., a tree-based generator consisting of a regular tree grammar and an HR algebra) that generates the set of all *wheels* W_n, for $n \geq 1$. Here, $W_n = (\{v_0, v_1, \ldots, v_n\}, \{e_1, \ldots, e_n, e'_1, \ldots, e'_n\}, att)$, where $att(e_i) = (v_0, v_i)$ (the "spokes" of the wheel) and $att(e'_i) = (v_i, v_{(i \bmod n)+1})$ (the "rim"), for $i \in [n]$. For instance, Figure 1 shows W_4.

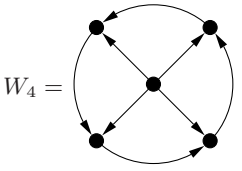


Fig. 1. The wheel with four spokes

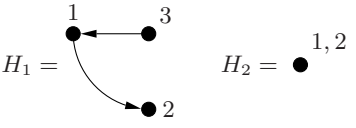


Fig. 2. Graphs to compose wheels of; the numbers indicate the ports, e.g., the bottom node of H_1 is its 2-port

Our grammar assembles such wheels from the graphs shown in Figure 2. It uses two nonterminals, ξ_{ini} and ξ, and the rules

$$\begin{aligned}
\xi_{\text{ini}} &\to \mathtt{rel}_{[]}[\mathtt{par}[\xi, \mathtt{H}_2]] \\
\xi &\to \mathtt{rel}_{\left[\begin{smallmatrix}1\mapsto 1\\ 2\mapsto 4\\ 3\mapsto 3\end{smallmatrix}\right]}[\mathtt{par}[\xi, \mathtt{rel}_{\left[\begin{smallmatrix}2\mapsto 1\\ 3\mapsto 3\\ 4\mapsto 2\end{smallmatrix}\right]}[\xi]]] \\
\xi &\to \mathtt{H}_1.
\end{aligned}$$

Here, a subscript of the form $\left[\begin{smallmatrix} i_1 \mapsto i'_1 \\ \vdots \\ i_k \mapsto i'_k \end{smallmatrix}\right]$ denotes the partial function $\rho\colon \mathbb{N} \to \mathbb{N}$ with $df(\rho) = \{i_1, \ldots, i_k\}$ and $\rho(i_j) = i'_j$ for all $j \in [k]$. In the first rule, parallel composition with H_2 just means that the 1-port of the first argument is identified with its 2-port, thus closing the rim, whereas the application of $rel_{[]}$ removes all port labels. Of course, the second rule could be replaced by the linear rule

$$\xi \to \mathtt{rel}_{\left[\begin{smallmatrix}1\mapsto 1\\ 2\mapsto 4\\ 3\mapsto 3\end{smallmatrix}\right]}[\mathtt{par}[\xi, \mathtt{rel}_{\left[\begin{smallmatrix}2\mapsto 1\\ 3\mapsto 3\\ 4\mapsto 2\end{smallmatrix}\right]}[H_1]]]$$

without affecting the generated language. Figure 3 indicates how to evaluate the right-hand side of the second rule if both occurrences of ξ are replaced with $\mathtt{H}_1$.

Using an ET0L tree grammar (see Section 3.4) instead of a regular one, taking the same rules but putting them into separate tables, the set of all wheels with 2^n spokes would be generated, which is neither HR nor NR context free.

3.4 Generation of Line Drawings

Tree-based generation has also turned out to be very useful in the area of picture generation, because several well-known devices generating pictures can be given a

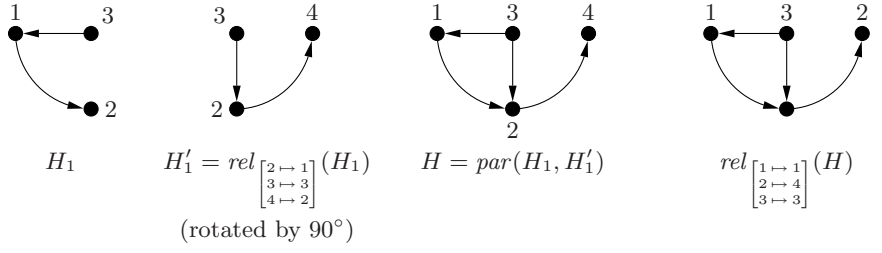


Fig. 3. The evaluation of $\mathtt{rel}_{\begin{bmatrix}1\mapsto 1\\2\mapsto 4\\3\mapsto 3\end{bmatrix}}[\mathtt{par}[\mathtt{H}_1, \mathtt{rel}_{\begin{bmatrix}2\mapsto 1\\3\mapsto 3\\4\mapsto 2\end{bmatrix}}[\mathtt{H}_1]]]$

tree-based definition.[5] Two such devices are chain-code picture grammars and L-systems with turtle interpretation. We will briefly discuss the latter, which can be seen as an extension of the former. In its original definition, the turtle mechanism interprets strings whose symbols are regarded as instructions to a plotter-like device. It originated from the "turtle" of the programming language LOGO [Ad80] and was made popular in the area of grammatical picture generation using L-systems through the book by Prusinkiewicz and Lindenmayer [PL90]; see also the later survey [PHHM97].

Let us say that a *line drawing* (in $\mathbb{R}^2$) is a pair $\Delta = (D, e)$ consisting of a finite set D of straight line segments and an end point $e \in \mathbb{R}^2$. Furthermore, choose two angles α_0 and α. Given an unsorted signature Σ containing

$$\Sigma_{\text{turtle}} = \{\mathtt{F}^{(0)}, +^{(1)}, -^{(1)}, \mathtt{enc}^{(1)}\}$$

as a subset, the *turtle Σ-algebra*[6] $\mathcal{A}$ has as its domain the set of all line drawings. The interpretation of symbols in Σ by $\mathcal{A}$ depends on α_0 and α, and is given as follows.

- F is the line drawing consisting of a single line segment extending one unit from the origin into the direction given by α_0. The end point of this line segment is the end point of F.
- The symbols $+$ and $-$ are interpreted as rotation around the origin by α and $-\alpha$ degrees, resp. Here, both the line segments and the end point are rotated.
- The symbol `enc` is interpreted as *encapsulation*, replacing the end point of the argument by the origin: $enc(D, e) = (D, (0, 0))$.
- Every symbol $\mathtt{f}^{(k)} \notin \Sigma_{\text{turtle}}$ is interpreted as the k-ary concatenation of line drawings: $f(\Delta_1, \ldots, \Delta_k) = (\cdots(\Delta_1 \circ \Delta_2) \circ \cdots) \circ \Delta_k$, where $\Delta \circ \Delta'$ is obtained by translating Δ' by the vector given by the end point of Δ and taking the union of the sets of line segments of both. The translated end point of Δ' becomes the end point of the resulting line drawing.

[5] See [Dre06] for an extensive treatment of tree-based picture generation.
[6] Slightly simplified, compared to [Dre06].

As the turtle mechanism is usually studied in connection with L-systems, let us recall the definition of ET0L tree grammars, which is the tree grammar version of ET0L systems.

Definition 3 (ET0L tree grammar). *Let S be a set of sorts. An* ET0L tree grammar *is a tuple $\gamma = (\Xi, \Sigma, R, t_0)$ consisting of*

- *(not necessarily disjoint) S-sorted signatures Ξ and Σ of* nonterminals *(each of rank 0) and* output symbols*, resp.,*
- *a finite set R of* tables *$R_1, \dots, R_n$, each table being a finite set of rules as in the case of regular tree grammars, and*
- *an* axiom *$t_0 \in \mathrm{T}_{\Sigma \cup \Xi}$.*

To guarantee that $\Sigma \cup \Xi$ is a well-defined signature, it is required that each nonterminal occurring in Σ has the same profile in both signatures. Moreover, in each table R_i, every nonterminal is required to occur among the left-hand sides of rules in R_i.

For trees $s, t \in \mathrm{T}_{\Sigma \cup \Xi}$, there is a derivation step *$s \Rightarrow_\gamma t$ (or just $s \Rightarrow t$) if there is a table R_i such that t can be obtained from s by simultaneously replacing every occurrence of a nonterminal ξ by r, where $\Xi \to r$ is a rule in R_i. The* ET0L tree language generated by γ *is given by*

$$L(\gamma) = \{t \in \mathrm{T}_\Sigma \mid t_0 \Rightarrow^* t\}.$$

A tree-based generator consisting of an ET0L tree grammar and a turtle algebra is called an ET0L turtle grammar. Such grammars have been used quite extensively to capture plant architecture by means of grammatical rules. To discuss an example, let $\gamma = (\Xi, \Sigma, \{R_1, R_2\}, \xi_{\mathrm{ini}})$, where $\Xi = \{\xi_{\mathrm{ini}}, \xi\}$, $\Sigma = \Xi \cup \Sigma_{\mathrm{turtle}} \cup \{\mathtt{c}_2^{(2)}, \mathtt{c}_3^{(3)}\}$, and

$$R_1 = \{\xi_{\mathrm{ini}} \to \mathtt{c}_3[\mathtt{F}, \mathtt{enc}[+[\mathtt{c}_3[\mathtt{F}, \xi_{\mathrm{ini}}, -[\mathtt{F}]]]], \mathtt{enc}[-[\xi_{\mathrm{ini}}]]],\ \mathtt{F} \to \mathtt{c}_2[\mathtt{F}, \xi_{\mathrm{ini}}]\},$$
$$R_2 = \{\xi_{\mathrm{ini}} \to \mathtt{c}_3[\mathtt{F}, \mathtt{enc}[-[\mathtt{c}_3[\mathtt{F}, \xi_{\mathrm{ini}}, +[\mathtt{F}]]]], \mathtt{enc}[+[\xi_{\mathrm{ini}}]]],\ \mathtt{F} \to \mathtt{c}_2[\mathtt{F}, \xi_{\mathrm{ini}}]\}.$$

Thus, γ is a so-called DT0L tree grammar: its tables are deterministic, and all nonterminals are output symbols.

Note that derivations in γ never terminate. However, as $\Xi \subseteq \Sigma$, all trees that are derivable from ξ_{ini} are in $L(\gamma)$. Now, let $\mathcal{A}$ be the turtle Σ-algebra with $\alpha_0 = 90°$ and $\alpha = 22.5°$. An initial part of a derivation in $G = (\gamma, \mathcal{A})$ (i.e., a derivation in γ whose individual trees are interpreted using $\mathcal{A}$) is shown in Figure 4, while Figure 5 shows some randomly chosen pictures in $L(G)$. Note that the figures do not show correct relative sizes. As the pictures grow beyond any bound as derivations get longer and longer, they must be scaled in an appropriate manner.

3.5 Generation of Collages

Another well-known type of picture generator is the collage grammar, which was originally introduced in [HK91] (see also [DK99] and, for the tree-based version,

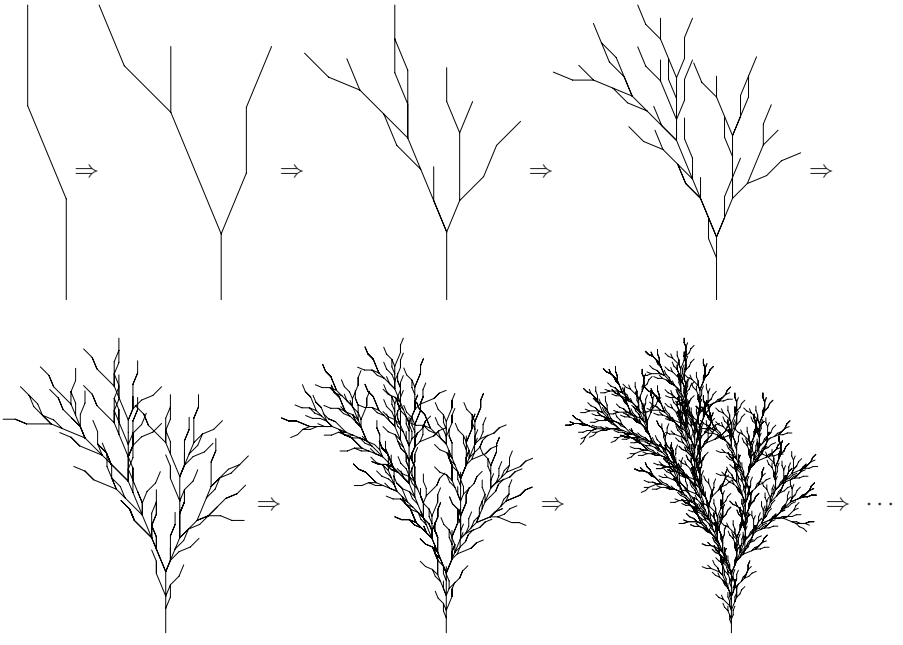


Fig. 4. An initial part of a derivation in an ET0L turtle grammar

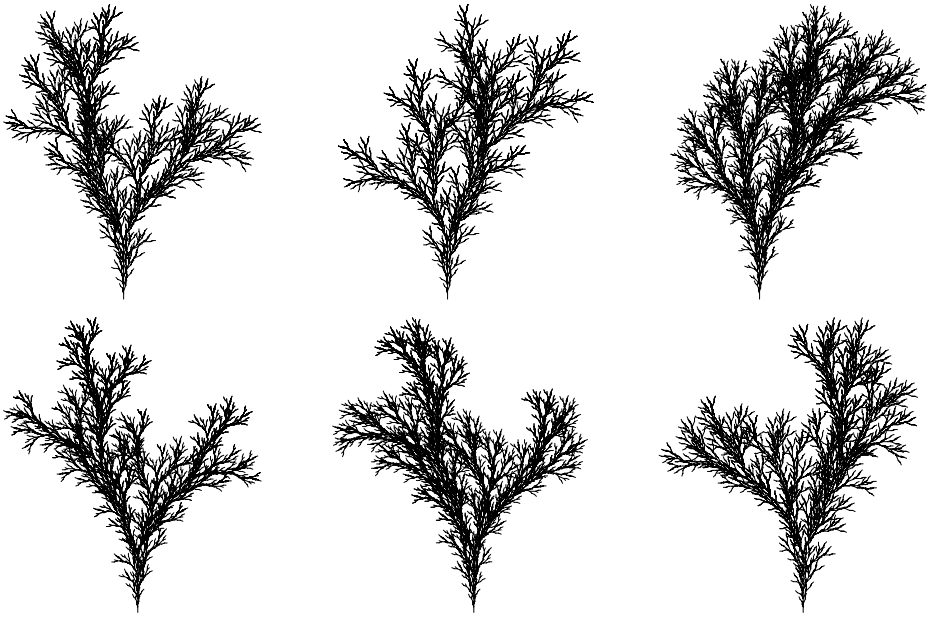


Fig. 5. Pictures generated by an ET0L turtle grammar

[Dre06]). Choose some dimension $d \geq 1$. A *collage* in the Euclidean space $\mathbb{R}^d$ is a finite set of *parts*, each part being a nonempty and bounded subset of $\mathbb{R}^d$. Recall that an affine transformation of $\mathbb{R}^d$ is a mapping of the form $\alpha(x) = Mx + b$, where M is a $d \times d$ matrix and $b \in \mathbb{R}^d$. Such an affine transformation is applied to a part in a pointwise manner. Applying it to a collage means to apply it to each of its parts.

Now, given injective affine transformations $\alpha_1, \ldots, \alpha_k$ and a collage C, let $\langle \alpha_1 \cdots \alpha_k, C \rangle$ denote the k-ary operation f on collages given by

$$f(C_1, \ldots, C_k) = C \cup \bigcup_{i \in [k]} \alpha_i(C_i).$$

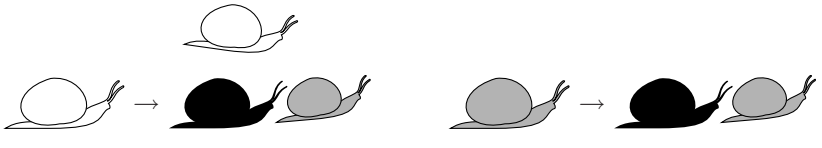
As a side remark, it may be interesting to note that, in particular, f is equal to the constant C if $k = 0$. Moreover, if $C = \emptyset$, then f is equal to the affine transformation α_1 (viewed as a unary operation on collages) if $k = 1$, and equal to the union of collages if $k = 2$ and $\alpha_1 = \alpha_2 = id$ (where id denotes the identity). For many types of tree-based collage generators, these three types of operations suffice to obtain the full generative power.

A *collage algebra* is a Σ-algebra whose domain is the set of all collages (in $\mathbb{R}^d$) and which interprets every symbol in Σ as a collage operation. Here, Σ is any unsorted signature.

Let us briefly look at an example for $d = 2$ (taken from [Dre06]). We use an EDT0L tree grammar[7] γ whose nonterminals are ξ_{ini} and ξ, and whose output signature is $\Sigma = \{\mathtt{f}^{(2)}, \mathtt{g}^{(1)}, \mathtt{snail}^{(0)}\}$. The axiom consists of the nonterminal ξ_{ini}, and the tables are

$$\{\xi_{\text{ini}} \to \mathtt{f}[\xi, \xi_{\text{ini}}],\ \xi \to \mathtt{g}[\xi]\},\ \{\xi_{\text{ini}} \to \mathtt{snail},\ \xi_{\text{ini}} \to \mathtt{snail}\}.$$

Instead of giving a formal definition of the collage Σ-algebra $\mathcal{A}$ used to interpret the trees in $L(\gamma)$, let us show how the rules in the first table look if they are interpreted in $\mathcal{A}$. For this, we extend $\mathcal{A}$ to $\Sigma \cup \Xi$, and interpret every constant symbol as a part whose outline resembles a snail. To be able to distinguish between the symbols, `snail` is filled with black, ξ_{ini} with white, and ξ with grey[8]. Using this interpretation of constant symbols, the two rules in the first table look like this:



Clearly, each of the rules in the second table replaces the corresponding "nonterminal snail" with the black one.

[7] i.e., with deterministic tables.

[8] Formally, the grey fill colour may be interpreted as a sparse area of the part, e.g., a region where the part contains only rational points.

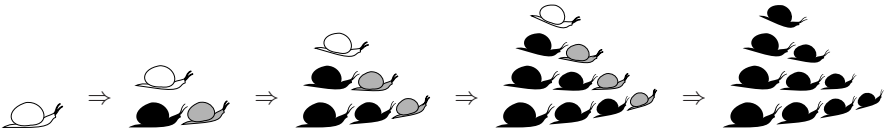


Fig. 6. Deriving a picture made of snails

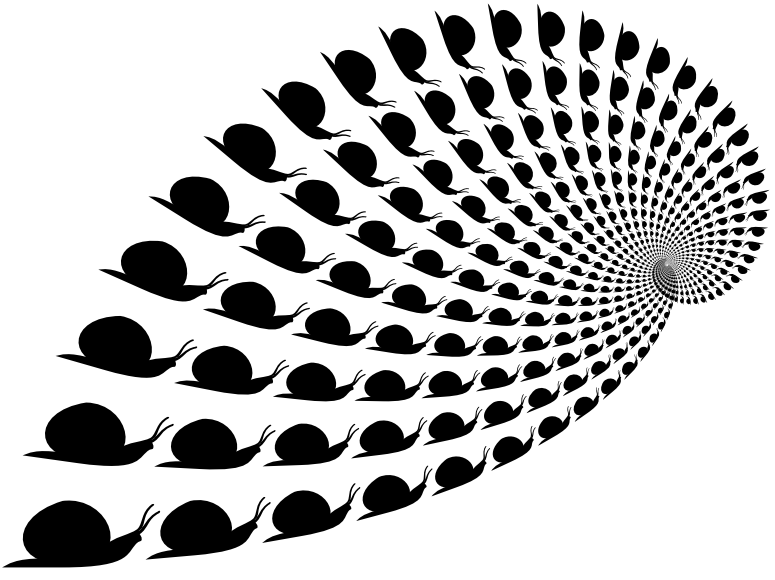


Fig. 7. Many snails

In this example, all relevant derivations consist of a number of applications of the first table, followed by one application of the second table. A short derivation of this kind is shown in Figure 6, while Figure 7 shows a picture generated by a rather long derivation.

3.6 Music Generation

Let us briefly mention that tree-based generators are even suitable for the generation of "music" – sound structures that adhere to certain basic rules of composition. A suitable algebra whose domain is the set of all musical pieces (in a certain sense) is proposed in [DH07]. It contains operations that, e.g., invert a piece, play it backwards, concatenate two pieces or create their overlay. In [DH07], a tree generator consisting of a regular tree grammar and a sequence of so-called tolerant top-down and macro tree transducers is used in connection with the music algebra in order to generate simple musical pieces. On `http://www.cs.umu.se/~johanna/algebra`, the implementation in Treebag and some generated pieces can be found.

4 Delegation Networks

A potential application area of tree-based generators concerns the generation of complex scenes involving structural, spatial, and pictorial elements. As an example, one may think of a virtual reality consisting of streets, buildings, plants, and other objects. For such an application, grammatical approaches to picture generation could be particularly useful, as they allow to generate very detailed models using simple and well-understood rules. As the previous section showed, one may indeed speak of models being generated since most of these systems actually do not generate pictures in a strict sense. Instead, they generate objects having a natural pictorial representation. For example, a collage grammar in $\mathbb{R}^3$ generates collections of three-dimensional objects that become pictures only if they are passed through a ray tracer or similar software. Thus, in principle, a collage grammar could be used to generate a virtual reality. For example, the street shown in Figure 8 has been generated in Treebag using a collage grammar designed by C. von Totth. Unfortunately, each of the grammatical picture generators studied in the theoretical literature (such as ET0L turtle grammars and collage grammars) is only suitable for generating a rather specific type of structures. Moreover, the devices themselves provide no support for assembling large systems from smaller components – which would certainly be needed for

Fig. 8. A street generated by a collage grammar (designed by C. von Totth)

systems comprising thousands of rules or in a context where generators are designed by a group of developers. A third disadvantage is that they provide no means for integrating nongrammatical methods in an elegant way.

In this section, we propose the delegation network, a generalization of the tree-based generator which tries to overcome these limitations by allowing to combine several generating devices. A delegation network $\mathcal{N}$ contains a finite number of so-called delegating generators, which are basically tree-based generators. However, each of them may "delegate" the generation of certain parts of the object to other delegating generators in the network. Moreover, delegation networks are based on many-sorted algebras whose predefined operations can be nondeterministic. This makes it possible to

- modularize the generation of complex objects in a meaningful way,
- combine different classes of generators, each working on a part of the problem and the domains it is appropriate for, and
- let parts of the generation task be performed by devices that are not tree based, and possibly not even grammatical at all, by viewing them as nondeterministic predefined operations.

To understand the last item, imagine that we are given some implementation of a (nongrammatical) method for generating random textures. We could then use this device in order to define a nondeterministic operation that takes a (grammatically generated) collage as input and applies nondeterministically one of the randomly generated texture patterns to it.

Conceptually, delegation is easily achieved. The tree generator component γ of a delegating generator $(\gamma, \mathcal{A})$ generates trees each of whose symbols is either interpreted by $\mathcal{A}$ or refers to another delegating generator of the network. The second case is what gives rise to delegation.

To give the formal definition of delegation networks, some preliminaries are needed. We start with the definition of nondeterministic functions.

4.1 Nondeterministic Functions and Algebras

A nondeterministic function from $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k$ to $\mathbb{A}$ is a function of the form $f\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \to \wp(\mathbb{A})$. We identify f with the function $f'\colon \wp(\mathbb{A}_1) \times \cdots \times \wp(\mathbb{A}_k) \to \wp(\mathbb{A})$ such that, for all $A_1 \subseteq \mathbb{A}_1, \ldots, A_k \subseteq \mathbb{A}_k$,

$$f'(A_1, \ldots, A_k) = \bigcup \{ f(a_1, \ldots, a_k) \mid a_1 \in A_1, \ldots, a_k \in A_k \}.$$

Further, we write $f\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ in order to indicate that f is a nondeterministic function, and call $\mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ its (nondeterministic) profile. As in the deterministic case, we write $f\colon \mathbb{A}$ instead of $f\colon \twoheadrightarrow \mathbb{A}$ if $k = 0$, and identify f with the set $f() \subseteq \mathbb{A}$.

Note that a partial function $f\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \to \mathbb{A}$ can be regarded as the special case where $|f(a_1, \ldots, a_k)| \leq 1$ for all $(a_1, \ldots, a_k) \in \mathbb{A}_1 \times \cdots \times \mathbb{A}_k$.

The basic definitions regarding signatures, trees, and algebras carry over from the deterministic case. In an S-sorted signature Σ, profiles may from now on be

nondeterministic (and a deterministic profile is regarded as a special case of a nondeterministic one). The definition of trees over Σ and the related notation are literally the same as in the deterministic case, except that $\to$ must be replaced with $\twoheadrightarrow$. Similarly, the definition of nondeterministic Σ-algebras and the related notations carry over from the deterministic case by replacing $\to$ with $\twoheadrightarrow$.

4.2 Evaluating Trees with Parameters in Nondeterministic Algebras

In the following, we want to represent complex operations by trees. For this purpose, let us reserve a set of special symbols which represent the parameters of such an operation. For every every sort $\mathtt{A}$, let $\{x_1^{\mathtt{A}}, x_2^{\mathtt{A}}, \dots\}$ be a countably infinite set of pairwise distinct *parameter symbols* of sort $\mathtt{A}$. The *parameter signature of type* $\mathtt{A}_1 \times \cdots \times \mathtt{A}_k$ is the signature $\{x_1^{\mathtt{A}_1}, \dots, x_k^{\mathtt{A}_k}\}$, containing the i-th parameter symbol $x_i^{\mathtt{A}_i}$ of profile $\mathtt{A}_i$ for every $i \in [k]$. In the following, we will simply denote $x_i^{\mathtt{A}_i}$ by x_i. To avoid confusion, we assume that parameter symbols occur only in parameter signatures, i.e., are not used as elements of ordinary signatures.

Given a nondeterministic Σ-algebra $\mathcal{A}$ and the parameter signature X of type $\mathtt{A}_1 \times \cdots \times \mathtt{A}_k$, we can evaluate trees $t \in \mathrm{T}_{\Sigma \cup X}^{\mathtt{A}}$. The result of this evaluation is denoted by $val_{\mathcal{A}}^X(t)$. It is the function $\varphi\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ such that $\varphi(a_1, \dots, a_k)$ is given as follows, for all $a_1 \in \mathbb{A}_1, \dots, a_k \in \mathbb{A}_k$:

- If $t = x_i$, then $\varphi(a_1, \dots, a_k) = \{a_i\}$.
- Otherwise, if $t = \mathtt{f}[t_1, \dots, t_n]$ with $\varphi_i = val_{\mathcal{A}}^X(t_i)$ for all $i \in [n]$, then $\varphi(a_1, \dots, a_k) = f(\varphi_1(a_1, \dots, a_k), \dots, \varphi_n(a_1, \dots, a_k))$.

Given a set $T \subseteq \mathrm{T}_{\Sigma \cup X}^{\mathtt{A}}$ of trees rather than a single tree, we let $val_{\mathcal{A}}^X(T) = \Phi$, where $\Phi(a_1, \dots, a_k) = \bigcup_{t \in T} val_{\mathcal{A}}^X(t)(a_1, \dots, a_k)$.

During the rest of this paper, we will drop the qualifier *nondeterministic* when talking about nondeterministic Σ-algebras.

4.3 The Definition and Semantics of Delegation Networks

For the formal definition of delegation networks, one additional notion is needed. Consider an S-sorted signature Σ and a Σ-algebra $\mathcal{A} = (dom, \sigma)$. Given a domain mapping dom' for another set S' of sorts, $\mathcal{A}$ is said to be *dom'-compatible* if $dom(\mathtt{A}) = dom'(\mathtt{A})$ for all $\mathtt{A} \in S \cap S'$.

Now, the formal definition of delegation networks reads as follows.

Definition 4 (delegation network). *A* delegation network *is a system* $\mathcal{N} = (\Sigma, dom, G, \mathtt{g}_0)$*, where*

- Σ *is an* S*-sorted signature of* generator symbols*, for some set* S *of sorts,*
- dom *is a domain mapping for* S*,*
- $\mathtt{g}_0$ *is a constant symbol in* Σ*, and*
- $G = (G_{\mathtt{g}})_{\mathtt{g} \in \Sigma}$ *is a* Σ*-indexed family of* delegating generators $G_{\mathtt{g}} = (\gamma_{\mathtt{g}}, \mathcal{A}_{\mathtt{g}})$*, where, for every generator symbol* $\mathtt{g}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \twoheadrightarrow \mathtt{A}$*,*

- $\mathcal{A}_{\mathtt{g}}$ *is a dom-compatible* $\Sigma_{\mathtt{g}}$*-algebra, for some signature* $\Sigma_{\mathtt{g}}$ *disjoint with* Σ*, and*
- $\gamma_{\mathtt{g}}$ *is a tree generator such that* $L(\gamma_{\mathtt{g}}) \subseteq \mathrm{T}^{\mathtt{A}}_{\Sigma_{\mathtt{g}} \cup \Sigma \cup X}$*, where* X *is the parameter signature of type* $\mathtt{A}_1 \times \cdots \times \mathtt{A}_k$*.*

The semantics of $\mathcal{N}$ is obtained by constructing a Σ-algebra $\mathcal{A}_{\mathcal{N}}$. Intuitively, a symbol $\mathtt{g} \in \Sigma$ is interpreted by evaluating the trees in $L(\gamma_{\mathtt{g}})$. To see what this means, let $t \in L(\gamma_{\mathtt{g}})$. According to Definition 4, every non-parameter symbol $\mathtt{f}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \twoheadrightarrow \mathtt{A}$ occurring in t is either interpreted by $\mathcal{A}_{\mathtt{g}}$ or a generator symbol. Thus, both cases yield an appropriate interpretation of $\mathtt{f}$ (using recursion if $\mathtt{f} \in \Sigma$), 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 choose a least fixed-point semantics for delegation networks.

For this purpose, we turn the set of all functions of $f, g\colon \mathbb{A}_1 \times \cdots \times \mathbb{A}_k \twoheadrightarrow \mathbb{A}$ into a complete lattice by defining $f \leq g$ if and only if $f(a_1, \ldots, a_k) \subseteq g(a_1, \ldots, a_k)$, for all $a_1 \in \mathbb{A}_1, \ldots, a_k \in \mathbb{A}_k$. This extends to (Σ, dom)-interpretations σ, σ', where Σ is S-sorted, in the obvious way: $\sigma \leq \sigma'$ if and only if $\sigma(f) \leq \sigma'(f)$ for all $f \in \Sigma$.

For the following definition, if $\mathcal{A} = (dom, \sigma)$ and $\mathcal{A}' = (dom', \sigma')$ are Σ- and Σ'-algebras, resp., where Σ and Σ' are disjoint and $\mathcal{A}'$ is dom-compatible, we let $\mathcal{A} \cup \mathcal{A}'$ denote the $\Sigma \cup \Sigma'$-algebra (dom'', σ'') such that $dom''(\mathtt{A}) = dom(\mathtt{A})$ for all $\mathtt{A} \in S$ and

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

We are now ready to define the semantics of delegation networks.

Definition 5 (semantics of delegation networks). *Let* $\mathcal{N} = (\Sigma, dom, G, \mathtt{g}_0)$ *be a delegation network.*

1. *The operator* $iterate_{\mathcal{N}}$ *on* (Σ, dom)*-interpretations* σ *is defined as follows:* $iterate_{\mathcal{N}}(\sigma)$ *is the* (Σ, dom)*-interpretation* σ' *such that, for every symbol* $\mathtt{g}\colon \mathtt{A}_1 \times \cdots \times \mathtt{A}_k \twoheadrightarrow \mathtt{A}$ *in* Σ*,*

$$\sigma'(\mathtt{g}) = val^X_{\mathcal{A}_{\mathtt{g}} \cup (dom, \sigma)}(L(\gamma_{\mathtt{g}})),$$

   where X *is the parameter signature of type* $\mathtt{A}_1 \times \cdots \times \mathtt{A}_k$*.*
2. *The least fixed point of* $iterate_{\mathcal{N}}$ *is denoted by* $\sigma_{\mathcal{N}}$*, and* $\mathcal{A}_{\mathcal{N}} = (dom, \sigma_{\mathcal{N}})$*. (Note that, by construction,* $iterate_{\mathcal{N}}$ *is monotonically increasing. Thus, by Tarski's fixed-point theorem, it has a least fixed point.)*
3. *The* language generated by $\mathcal{N}$ *is* $L(\mathcal{N}) = \sigma_{\mathcal{N}}(\mathtt{g}_0)$*.*

It should be noticed that, for a delegation network as above and $\mathtt{g} \in \Sigma$, $\sigma_{\mathcal{N}}(\mathtt{g})$ is just $val^X_{\mathcal{A}_{\mathtt{g}}}(L(\gamma_{\mathtt{g}}))$ if no symbols from Σ appear in the trees generated by $\gamma_{\mathtt{g}}$. Thus, a tree-based generator can be identified with a delegation network in which $\Sigma = \{\mathtt{g}_0\}$, $\gamma(\mathtt{g}_0)$ has the output signature $\Sigma_{\mathtt{g}_0}$ (i.e., does not delegate to itself), and $\mathcal{A}_{\mathtt{g}_0}$ is a deterministic $\Sigma_{\mathtt{g}_0}$-algebra.

From an implementation point of view, one may think of each delegating generator $G_{\mathtt{g}}$ in $\mathcal{N}$ as a nondeterministic device working as follows. Let the profile of $\mathtt{g}$ be $\mathtt{A}_1 \times \cdots \times \mathtt{A}_k \twoheadrightarrow \mathtt{A}$, and let $a_1 \in \mathbb{A}_1, \ldots, a_k \in \mathbb{A}_k$ be arguments. We may then nondeterministically "execute" $G_{\mathtt{g}}$ in order to produce an element of the set $g(a_1, \ldots, a_k)$. For this purpose, we first run $\gamma_{\mathtt{g}}$ as a tree generator, which produces a tree $t \in \mathrm{T}^{\mathtt{A}}_{\Sigma_{\mathtt{g}} \cup \Sigma \cup X}$. The tree t is evaluated in a bottom-up manner by nondeterministically assigning a value to each of its nodes. To each leaf carrying a variable $x_i \in X$, the value a_i is assigned. Now, consider the root node of a subtree $s \notin X$ with direct subtrees $s_1, \ldots, s_l$, and suppose that we have already assigned values $b_1, \ldots, b_l$ to its direct descendants, i.e., to the roots of $s_1, \ldots, s_l$. There are two cases.

- If $s = \mathtt{f}[s_1, \ldots, s_l]$ with $\mathtt{f} \in \Sigma_{\mathtt{g}}$, we choose nondeterministically any element of $f(b_1, \ldots, b_l)$ (where f is the interpretation of $\mathtt{f}$ in $\mathcal{A}_{\mathtt{g}}$) as the value of s.
- If $s = \mathtt{g}'[s_1, \ldots, s_l]$ with $\mathtt{g}' \in \Sigma$, we create (recursively) an instance of $G_{\mathtt{g}'}$, apply it to the arguments $b_1, \ldots, b_l$, and consider the result to be the value assigned to the root node of s.

Without the last case, this is just the evaluation of trees with respect to $\mathcal{A}_{\mathtt{g}}$, in the sense that the set of all possible results that can be obtained equals $val^X_{\mathcal{A}_{\mathtt{g}}}(t)(a_1, \ldots, a_k)$. Thus, the base case is the one where $t \in \mathrm{T}_{\Sigma_{\mathtt{g}} \cup X}$, as such trees can be evaluated directly. Of course, a naive implementation may lead to an infinitely descending recursion because of the second case, if the delegation structure is cyclic. Thus, some care must be taken in an implementation.

5 An Example

Let us now consider an example. We make use of two domains, namely the set $\mathbb{C}$ of collages in $\mathbb{R}^2$ and the set $\mathbb{N}$ of natural numbers. The corresponding sorts are $\mathtt{C}$ and $\mathtt{N}$, resp. This defines the domain mapping *dom* to be used throughout this section, i.e., $dom(\mathtt{N}) = \mathbb{N}$ and $dom(\mathtt{C}) = \mathbb{C}$. As operations on $\mathbb{N}$, we use the constant 0 and the successor function s. The operations on $\mathbb{C}$ used are of two different types. On the one hand, we use the collage operations explained in Section 3.5. One the other hand, we turn cellular automata into operations on collages. For this, let us first recall what a cellular automaton is.

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 $cell_{ij}$ $(i, j \in \mathbb{Z})$. Geometrically, we identify the cell $cell_{ij}$ 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 $Q = \{0, \ldots, k\}$ 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 $\Delta\colon Q^{3\times 3} \to Q$, where $\Delta\left(\begin{smallmatrix}0&0&0\\0&0&0\\0&0&0\end{smallmatrix}\right) = 0$. Initially, the so-called inactive state 0 is assigned to all cells $cell_{ij}$ except $cell_{00}$, 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 $cell_{ij}$ changes its state to $\Delta(N)$, where N is the 3×3 array of states of its neighbouring cells (including $cell_{ij}$ itself), i.e., the states of the cells $cell_{pq}$ 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 $cell_{00}$ will be $\Delta\binom{000}{010}{000}$, and the state of $cell_{11}$ will be $\Delta\binom{000}{000}{100}$ since $cell_{11}$ has $cell_{00}$ as its lower left neighbour. Note that the number of active cells will always remain finite, because $\Delta\binom{000}{000}{000} = 0$.

Now, to turn a cellular automaton CA as above into an operation acting on collages, we view it as a function $CA\colon \mathbb{N} \times \mathbb{C}^k \to \mathbb{C}$, where $CA(n, C_1, \ldots, C_k)$ is obtained as follows. First, CA is executed n steps. Then, for every cell whose current state is $q \in [k]$, a copy of C_q is horizontally and vertically scaled and translated in such a way that the cell $cell_{ij}$ becomes its bounding box[9]. The resulting collage is the union of all these transformed copies of $C_1, \ldots, C_k$.

Note that one could alternatively view CA as a nondeterministic function $CA'\colon \mathbb{C}^k \twoheadrightarrow \mathbb{C}$, where $CA'(C_1, \ldots, C_k) = \{ca(n, C_1, \ldots, C_k) \mid n \in \mathbb{N}\}$. Obviously, this would not allow for as much control as the variant used here.

In the following, we use only one nontrivial cellular automaton CA, where $k = 3$. Rather than defining CA formally, let us use a delegation network $\mathcal{N}_0 = (\{\mathtt{g}_0\colon \mathbb{C}\}, dom, (\gamma_{\mathtt{g}_0}, \mathcal{A}_{\mathtt{g}_0}), \mathtt{g}_0)$ containing only one delegating generator, to show how CA behaves. The algebra $\mathcal{A}_{\mathtt{g}_0}$ contains the operations CA, 0, s, and constant collages C_i, $1 \leq i \leq 3$. The C_i are hollow squares with different figures placed inside: a square with indented edges, a triangle, and a hollow circle, resp. The tree generator $\gamma_{\mathtt{g}_0}$ is a regular tree grammar generating the tree language $\{\mathtt{CA}[\mathtt{s}^n[\mathtt{0}], \mathtt{C}_1, \mathtt{C}_2, \mathtt{C}_3] \mid n \in \mathbb{N}\}$.[10] The obvious definition of $\gamma_{\mathtt{g}_0}$ is omitted here. The resulting collages for $0 \leq i \leq 6$ are shown in Figure 9.

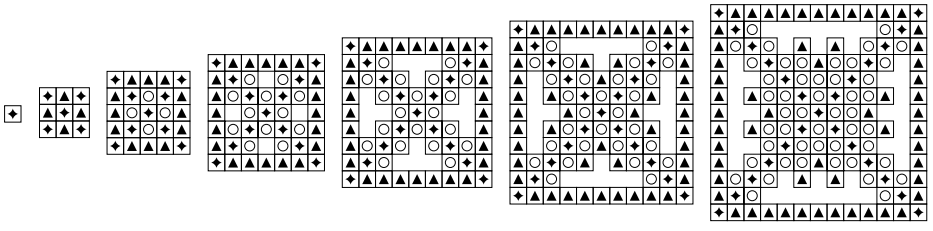


Fig. 9. Initial steps of the cellular automaton CA

Now, let us discuss a delegation network $\mathcal{N} = (\Sigma, dom, G, \mathtt{g})$ that makes use of CA in a slightly more interesting way. The signature Σ consists of the symbols $\mathtt{g}\colon \mathtt{C}$, $\mathtt{ca}\colon \mathtt{N} \times \mathtt{C}^3 \twoheadrightarrow \mathtt{C}$, and $\mathtt{ifs}\colon \mathtt{C} \twoheadrightarrow \mathtt{C}$. The delegation structure in this example is such that $\mathtt{g}$ uses the other two, and $\mathtt{ifs}$ delegates to itself. All tree generators employed are regular tree grammars.

[9] For the purpose of this example, we may disregard collages C_q for which such a scaling is impossible.

[10] As usual, $\mathtt{s}^0[\mathtt{0}] = \mathtt{0}$ and $\mathtt{s}^{i+1}[\mathtt{0}] = \mathtt{s}[\mathtt{s}^i[\mathtt{0}]]$ for all $i \in \mathbb{N}$.

Let us first discuss $(\gamma_{\texttt{ifs}}, \mathcal{A}_{\texttt{ifs}})$, which basically implements an iterated function system. To achieve this, the tree generator $\gamma_{\texttt{ifs}}$ generates the finite tree language $\{x_1, \texttt{ifs}[\texttt{f}[x_1, x_1, x_1]]\}$. In $\mathcal{A}_{\texttt{ifs}}$, $\texttt{f}$ is interpreted as a collage operation f of the form $\langle \alpha_1\alpha_2\alpha_3, \mathit{line}\rangle$, where the transformations $\alpha_1, \alpha_2, \alpha_3$ and the collage *line* are chosen in such a way that, e.g.,

$f($  $) =$.

As a consequence of the fact that $L(\gamma_{\texttt{ifs}}) = \{x_1, \texttt{ifs}[\texttt{f}[x_1, x_1, x_1]]\}$, the application of $\mathit{ifs} = \sigma_{\mathcal{N}}(\texttt{ifs})$ to a collage C yields C and all collages obtained by applying *ifs* recursively to $f(C, C, C)$. Thus, for example, $\mathit{ifs}($ $)$ yields the set of collages indicated in Figure 10.

Fig. 10. Collages generated by *ifs*, if applied to (up to scaling)

Now, let us discuss $G_{\texttt{ca}} = (\gamma_{\texttt{ca}}, \mathcal{A}_{\texttt{ca}})$. The tree generator $\gamma_{\texttt{ca}}$ is the regular tree grammar having only one nonterminal $\xi_{\text{ini}}\colon \texttt{C}$ and the rules

$$\xi_{\text{ini}} \to \texttt{CA}[x_1, \xi_{\text{ini}}, x_3, x_4],$$
$$\xi_{\text{ini}} \to \texttt{CA}[x_1, x_2, x_3, x_4].$$

The cellular automaton *CA* (i.e., the interpretation of the symbol $\texttt{CA}$) is as before.

Obviously, $\gamma_{\texttt{ca}}$ generates all trees

$$\texttt{CA}[x_1, \texttt{CA}[x_1, \ldots \texttt{CA}[x_1, x_2, x_3, x_4] \ldots, x3, x4], x_3, x_4].$$

Note that the recursion takes place in the argument corresponding to state 1 of *CA* (i.e., where Figure 9 contains a copy of), and all instances of *CA* perform the same number of steps, as determined by the parameter x_1.

Finally, the first component of $G_{\texttt{g}} = (\gamma_{\texttt{g}}, \mathcal{A}_{\texttt{g}})$ is the regular tree grammar with nonterminals $\xi_{\text{ini}}\colon \texttt{C}$ and $\xi\colon \texttt{N}$, and the rules

$$\xi_{\text{ini}} \to \texttt{ca}[\xi, \text{}, \texttt{ifs}[\text{}], \text{}],$$
$$\xi \to \texttt{s}[\xi],$$
$$\xi \to \texttt{0}.$$

In $\mathcal{A}_{\texttt{g}}$, , , and are interpreted as the collages consisting of the corresponding parts (and $\texttt{s}$ and $\texttt{0}$ are interpreted as successor, s, and zero, 0). Thus, the purpose of $G_{\texttt{g}}$ in this example is to provide $G_{\texttt{ca}}$ with sample arguments. From the point of view of $\gamma_{\texttt{g}}$, the only argument that is not fixed, but generated in a nondeterministic manner, is the first one, determining how many steps *CA*

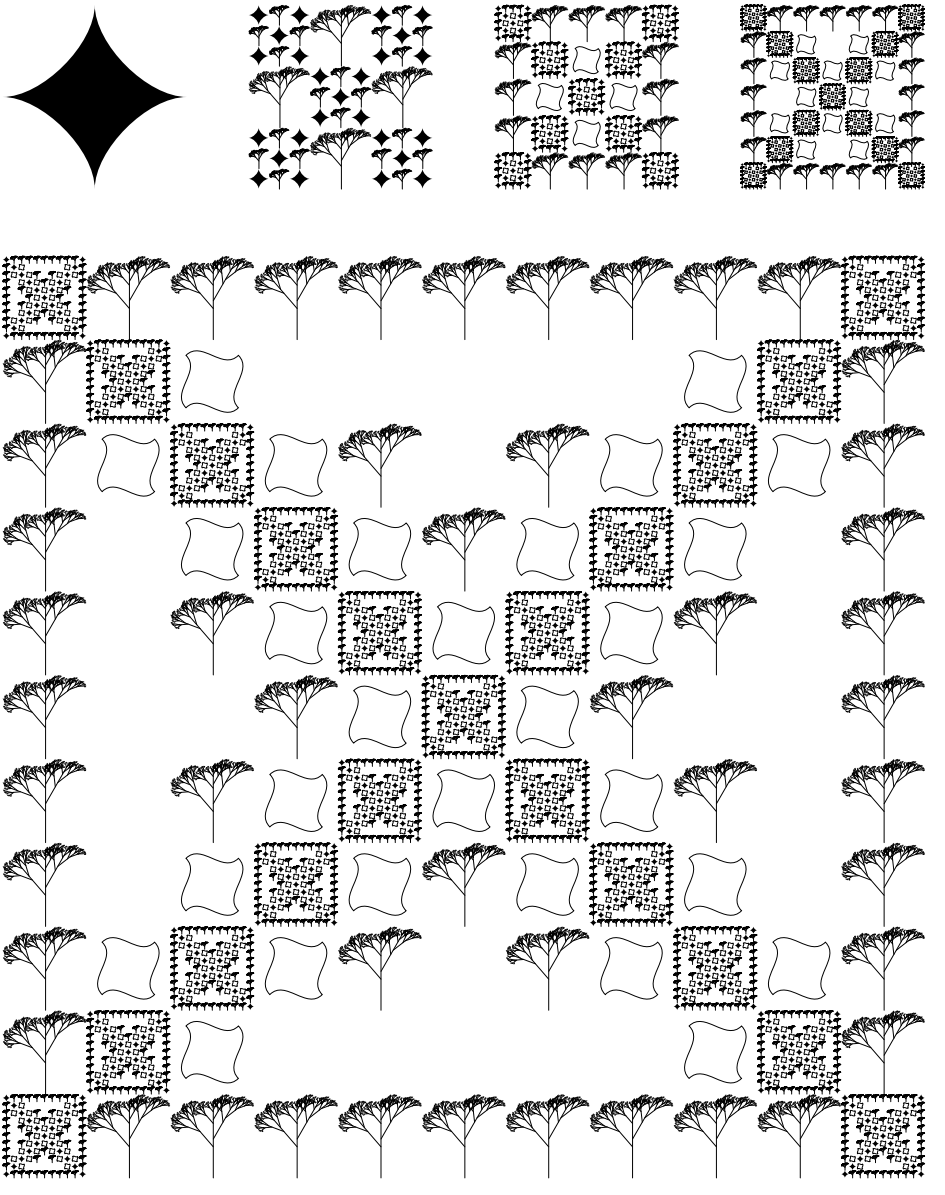


Fig. 11. Collages generated by the delegation network $\mathcal{N}$

is executed. However, actually, even the third argument is nondeterministic, as *ifs*(⧫) is not a single collage, but the set displayed in Figure 10.

Figure 11 shows some of the collages in $L(\mathcal{N})$. They result from the trees $\mathtt{ca}[\mathtt{s}^i[\mathtt{0}], ✦, \mathtt{ifs}[⧫], ⌑]$ in $L(\gamma_{\mathtt{g}})$, for $i = 0, \ldots, 3$ (top) and $i = 5$ (bottom). Among the trees in $L(\gamma_{\mathtt{ca}})$, the tree $\mathtt{CA}[x_1, \mathtt{CA}[x_1, x_2, x_3, x_4], x_3, x_4]$ has been used. The

reader may find it instructive to compare this figure with Figure 9, in order to discover where the structures in Figure 9 reappear in Figure 11.

6 Conclusion and Outlook

In this paper, we have given a brief survey of tree-based generators, and have introduced the delegation network as a generalization. Future work should study both theoretical and practical issues regarding delegation networks, and provide an implementation.

Some initial results regarding delegation networks are given in [Dre07]. However, as mentioned in the introduction, evaluation in nondeterministic algebras is not correctly defined in this paper. Thus, the results of [Dre07] should be taken with care. However, under the assumption that the union $\Sigma_\cup$ of the signatures $\Sigma_\mathtt{g}$, $\mathtt{g} \in \Sigma$, is well defined, it is clear that a delegation network $\mathcal{N} = (\Sigma, dom, G, \mathtt{g}_0)$ can be used to generate a tree language $T(\mathcal{N})$ by defining $T(\mathcal{N}) = L(\mathcal{N}')$, where $\mathcal{N}'$ is obtained from $\mathcal{N}$ by

1. replacing dom with dom', where $dom'(\mathtt{A}) = \mathrm{T}^{\mathtt{A}}_{\Sigma_\cup}$ for every sort $\mathtt{A}$, and
2. replacing each $\mathcal{A}_\mathtt{g}$ ($\mathtt{g} \in \Sigma$) with the free term algebra over $\Sigma_\cup$.

Now, if the algebras $\mathcal{A}_\mathtt{g}$ are deterministic and satisfy some reasonable compatibility requirements (i.e., do not interpret common sorts or function symbols differently), the "Mezei&Wright-like" result $L(\mathcal{N}) = val_\mathcal{A}(T(\mathcal{N}))$ holds, where $\mathcal{A}$ is the union of the algebras $\mathcal{A}_\mathtt{g}$. For the case of nondeterministic algebras, a counterexample was given in [ES78, p. 72]. In fact, looking at the notions and results in [ES77, ES78], it seems that $T(\mathcal{N})$ can be characterized by a generalized version of IO context-free tree grammars. Together with the Mezei&Wright-like result, this would yield an equivalent operational semantics for delegation networks whose algebras are deterministic (or, in other words, a characterization in terms of tree-based generators).

As delegation networks can generate tree languages, they can take the role of tree generators in delegation networks. A characterization of the tree languages $T(\mathcal{N})$ in terms of extended IO context-free tree grammars may also help to understand the resulting *delegation hierarchy* $(DEL^n(REG))_{n\in\mathbb{N}}$. Here, $DEL^n(REG)$ is the class of tree languages generated by $\mathtt{DEL}^n(\mathtt{REG})$, which is defined as follows: $\mathtt{REG}$ is the class of all regular tree grammars, and $\mathtt{DEL}^{n+1}(\mathtt{REG})$ is the set of all delegation networks in which each $\gamma_\mathtt{g}$ is in $\mathtt{DEL}^n(\mathtt{REG})$.

Let us now discuss some more practical issues. Future plans include the implementation of a system that allows to define and execute delegation networks in a flexible and, to the extent possible, efficient manner. The flexibility of this system should preferably be similar to that of the system Treebag (see Section 2.4). However, in contrast to Treebag, whose implementation does not pay much attention to practical issues such as efficiency and usability for large examples, the development of the new system should address these points in particular.

In connection with this, it would be interesting to study the parallel and distributed execution of delegating generators. As indicated at the end of Section 4,

one may view a delegating generator $(\gamma_{\mathtt{g}}, \mathcal{A}_{\mathtt{g}})$ belonging to a delegation network $\mathcal{N} = (\Sigma, dom, G, \mathtt{g}_0)$ as a device that, internally, generates a tree (in a nondeterministic fashion) and evaluates it to a (nondeterministic) function. To be able to evaluate the tree, it creates an instance I of $(\gamma_{\mathtt{g}'}, \mathcal{A}_{\mathtt{g}'})$ for every occurrence of a symbol $\mathtt{g}' \in \Sigma$ it generates (which, at least for regular and ET0L tree grammars, can be done as soon as the occurrence is generated). The function eventually returned by I is then used as the interpretation of the given occurrence of $\mathtt{g}'$. The instances of delegating generators resulting from this process are independent of each other (except for the fact that parent instances have to wait for their children during the evaluation process), it should be possible to execute them in parallel or even distribute their execution over a cluster of processors or machines.

Another question that future investigations may address is dynamic execution. Especially in the generation of graphical scenes, a derivation can often be seen as a development in time (cf. Figure 4). In terms of the discussion above, this would mean that an instance of a delegating generator does not terminate when it has reported the evaluation of its generated tree to its parent. Instead, it may perform further derivation steps, each time reporting the accordingly updated evaluation to its parent. To do this in an efficient manner, one needs to develop incremental techniques that avoid full re-evaluation in each step. This seems to be an interesting research question – as is the theoretical question of defining a suitable "dynamic semantics" for delegation networks.

Acknowledgment. I thank Joost Engelfriet for pointing out problems with the basic definitions (and, thus, results) of [Dre07] and suggesting possible remedies to me. Furthermore, I thank Carolina von Totth for allowing me to include the picture shown in Figure 8.

References

[Ad80] Abelson, H., diSessa, A.: Turtle Geometry: The Computer as a Medium for Exploring Mathematics. MIT Press, Cambridge, MA (1980)

[BC87] Bauderon, M., Courcelle, B.: Graph expressions and graph rewriting. Mathematical Systems Theory 20, 83–127 (1987)

[CDG+02] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2002) available at, `http://www.grappa.univ-lille3.fr/tata`

[Cou90] Courcelle, B.: Graph rewriting: an algebraic and logic approach. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 193–242. Elsevier, Amsterdam (1990)

[DH07] Drewes, F., Högberg, J.: An algebra for tree-based music generation. In: Bozapalidis, S., Rahonis, G. (eds.) Cryptography and Coding 2007. LNCS, vol. 4728, pp. 161–173. Springer, Heidelberg (2007)

[DK99] Drewes, F., Kreowski, H.-J.: Picture generation by collage grammars. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation. Applications, Languages, and Tools, ch. 11, vol. 2, pp. 397–457. World Scientific, Singapore (1999)

[Dre06] Drewes, F.: Grammatical Picture Generation – A Tree-Based Approach. In: Texts in Theoretical Computer Science. An EATCS, Springer, Heidelberg (2006)

[Dre07] Drewes, F.: Delegation networks. Report UMINF 07.04, Umeå University (2007)

[Eng80] Engelfriet, J.: Some open questions and recent results on tree transducers and tree languages. In: Book, R.V. (ed.) Formal Language Theory: Perspectives and Open Problems, pp. 241–286. Academic Press, London (1980)

[Eng97] Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, ch. 3, vol. 3, pp. 125–213. Springer, Heidelberg (1997)

[ES77] Engelfriet, J., Schmidt, E.M.: IO and OI. I. Journal of Computer and System Sciences 15, 328–353 (1977)

[ES78] Engelfriet, J., Schmidt, E.M.: IO and OI. II. Journal of Computer and System Sciences 16, 67–99 (1978)

[FV98] Fülöp, Z., Vogler, H.: Syntax-Directed Semantics: Formal Models Based on Tree Transducers. Springer, Heidelberg (1998)

[GS84] Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)

[GS97] Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, ch.1, vol. 3, pp. 1–68. Springer, Heidelberg (1997)

[HK91] Habel, A., Kreowski, H.-J.: Collage grammars. In: Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Graph Grammars and Their Application to Computer Science. LNCS, vol. 532, pp. 411–429. Springer, Heidelberg (1991)

[Mai74] Maibaum, T.S.E.: A generalized approach to formal languages. Journal of Computer and System Sciences 8, 409–502 (1974)

[NP92] Nivat, M., Podelski, A. (eds.): Tree Automata and Languages. Elsevier, Amsterdam (1992)

[PHHM97] Prusinkiewicz, P., Hammel, M., Hanan, J., Měch, R.: Visual models of plant development. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, ch. 9, vol. 3, pp. 535–597. Springer, Heidelberg (1997)

[PL90] Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer, Heidelberg (1990)

[Rou70] Rounds, W.C.: Mappings and grammars on trees. Mathematical Systems Theory 4, 257–287 (1970)

[Tha73] Thatcher, J.W.: Tree automata: an informal survey. In: Aho, A.V. (ed.) Currents in the Theory of Computing, pp. 143–172. Prentice-Hall, Englewood Cliffs (1973)