Companion Notes to "Tetrasystems: A Framework for String Generation Devices"

Eero Lassila
Aalto University, Helsinki, Finland
`eero.lassila@aalto.fi`
`http://users.ics.aalto.fi/ela/dsrfs/nwpt2012/`

October 28, 2012

Contents

1 Preliminaries

The set of *letters* divides into *nonterminals* and *terminals*. A *word* is a finite letter sequence.

A *letter-refiner* is a function that takes three arguments: a word, a letter, and a word. The former word represents the left-hand context, and the latter the right-hand context. The letter-refiner returns a non-empty set of non-empty words, which represent the possible (mutually alternative) refinement results of the argument letter in the specified two-sided context. Each terminal refines only to itself.

We consider *trees* that are finite, rooted, and ordered. Each tree node holds a letter.

A *belt* of a tree is such a subset of the tree nodes that contains exactly one ancestor of each leaf of the tree. So the belts of a given tree are simply the cross sections of the tree.

2 Belt-selectors

2.1 Span between two tree nodes

Let n_1 and n_2 be two given nodes in a given tree, and let n_0 denote the lowest (i.e. closest) common ancestor of n_1 and n_2. We define the *span* between n_1 and n_2 as the following integer triple $\langle i, d, j\rangle$:

$$\begin{aligned} i &= \text{the difference of the depths of } n_1 \text{ and } n_0 \\ d &= \begin{cases} -1 & \text{if } n_2 \text{ is on the left of } n_1 \\ 0 & \text{if (at least) one of } n_1 \text{ and } n_2 \text{ is an ancestor of the other} \\ 1 & \text{if } n_2 \text{ is on the right of } n_1 \end{cases} \\ j &= \text{the difference of the depths of } n_2 \text{ and } n_0 \end{aligned}$$

(Of course, the depth of the root is 0, and the depth of a child is always one more than the depth of the parent.)

We denote the span between n_1 and n_2 as $\angle(n_1, n_2)$. Obviously, $\angle(n_1, n_2) = \langle i, d, j\rangle$ implies $\angle(n_2, n_1) = \langle j, -d, i\rangle$; and trivially, $\angle(n_1, n_1) = \angle(n_2, n_2) = \langle 0, 0, 0\rangle$. If n_1 is a child of n_2, then $\angle(n_1, n_2) = \langle 1, 0, 0\rangle$; and if n_1 and n_2 siblings (and distinct), then either $\angle(n_1, n_2) = \langle 1, -1, 1\rangle$ or $\angle(n_1, n_2) = \langle 1, 1, 1\rangle$.

2.2 Combs and belt-selectors

A *comb* is any such function $f : \{\ldots, -2, -1, 0, 1, 2, \ldots\} \to \{0, 1, 2, \ldots\} \cup \{\infty\}$ that $\forall i \neq 0 : f(i) > 0$.

For each comb, there is a corresponding *belt-selector*; and every belt-selector corresponds to some comb. Each belt-selector is such a function that takes a tree and one of its nodes as its arguments and returns such a belt of the argument tree that contains no proper ancestor of the argument node. For a given comb f, for a given tree X, and for a given node n of X, the belt selected by the belt-selector corresponding to f is the node set constructed by the following two successive steps:

1. Each such node $n' \in X$ that is not a proper ancestor of n is added to the set if $\angle(n, n') = \langle i, d, f(i \times d)\rangle$ for some i and d.

2. Each such leaf of X that has no ancestor already in the set is added to the set.

A belt-selector is said to be *settled* at a tree node if every leaf added by step 2 above holds a terminal.

It is easy to see that no belt-selector can correspond to two distinct combs, which means that the correspondence between combs and belt-selectors is a bijection. So for a given belt-selector s, the comb which s corresponds to can be denoted as f_s.

2.3 Belt-selector examples

First, we define belt-selectors σ_{E}, σ_{C}, and σ_{I} by specifying the corresponding combs (here i ranges over all integers, of course):

$$\begin{array}{rcl} f_{\sigma_{\mathsf{E}}}(i) & = & \infty \\ f_{\sigma_{\mathsf{C}}}(i) & = & |i| \\ f_{\sigma_{\mathsf{I}}}(i) & = & \left\{ \begin{array}{ll} 0 & \text{when } i = 0 \\ 1 & \text{otherwise} \end{array} \right. \end{array}$$

Clearly, σ_{E} is settled at a tree node if and only if all the leaves of the tree hold terminals. On the other hand, σ_{I} is settled at every node in every tree.

Second, we introduce two operations $\triangle s$ and $s \| s'$ (i.e. a unary one and a binary one) on belt-selectors:

$$\begin{array}{rcl} f_{\triangle s}(i) & = & f_s(i) + 1 \\ f_{s\|s'}(i) & = & \left\{ \begin{array}{cl} f_s(i) & \text{when } i < 0 \\ 0 & \text{when } i = 0 \\ f_{s'}(i) & \text{when } i > 0 \end{array} \right. \end{array}$$

So for example:

$$\begin{array}{rcl} f_{\triangle \sigma_{\mathsf{C}}}(i) & = & |i| + 1 \\ f_{\sigma_{\mathsf{E}}\|\sigma_{\mathsf{I}}}(i) & = & \left\{ \begin{array}{cl} \infty & \text{when } i < 0 \\ 0 & \text{when } i = 0 \\ 1 & \text{when } i > 0 \end{array} \right. \end{array}$$

3 Tetrasystems

A *tetrasystem* implements rewriting as a tree generation process. The operation of a tetrasystem is governed by a refinement rule base and a separate control mechanism. These two are intended to be as orthogonal to each other as possible.

The alphabet of a tetrasystem may be countably infinite, and the effective rewriting context may be unbounded in both directions.

3.1 Components of a tetrasystem

The two main components of a tetrasystem are a letter-refiner (i.e. the rule base) and a *frame* (i.e. the control mechanism) consisting of four belt-selectors.

All the components of a given tetrasystem $\langle V_N, V_T, c_S, r, \langle s_1, s_2, s_3, s_4 \rangle \rangle$ are as follows:

- set V_N of nonterminals must be non-empty but may be finite or countably infinite,
- set V_T of terminals may be empty, finite, or countably infinite,
- the *seed-letter* c_S must belong to V_N,

- the letter-refiner r must have the property that every possible refinement result of each letter in V_N may contain only letters in $V_N \cup V_T$,
- the frame $\langle s_1, s_2, s_3, s_4 \rangle$ is a quadruple of belt-selectors.

3.2 Tetrasystem operation

The tree generation process proceeds as follows:

1. The tree is booted up by introducing a single root node holding the seed-letter.
2. The tree grows by repeated expansion of leaves holding nonterminals:
   (a) (Use of s_1.) A leaf is *fertile* if it is nonterminal-lettered and s_1 is settled at it.
   (b) (Use of s_2.) At a time, exactly one of the fertile leaves (if any) is expanded by applying the letter-refiner: the two sides of the refinement context are given by the belt returned by s_2; and the node changes from a leaf into a non-leaf as it is now provided with a child node sequence collectively holding one of the possible refinement results.
3. Output words (if any) of the process can be found at any time (and so even if the process has not terminated):
   (a) (Use of s_3.) A node is *mature* if it is nonterminal-lettered and s_3 is settled at it.
   (b) (Use of s_4.) For any mature node, the belt returned by s_4 gives an output word.

So the process does not terminate as long as there are fertile nonterminal-lettered leaves. Fertileness implies that the leaf is ready to be expanded as the generation process has already proceeded sufficiently far in the other parts of the tree.

3.3 Some specific frames

Let us specify four frames M, C, P, and L as follows:

		s_1	s_2	s_3	s_4
macro processors	M	$\sigma_{\mathsf{E}} \| \sigma_{\mathsf{I}}$	σ_{E}	σ_{E}	σ_{E}
Chomsky grammars	C	σ_{I}	σ_{E}	σ_{E}	σ_{E}
pure grammars	P	σ_{I}	σ_{E}	$\triangle\sigma_{\mathsf{I}}$	σ_{E}
L systems	L	σ_{C}	σ_{C}	$\triangle\sigma_{\mathsf{C}}$	$\triangle\sigma_{\mathsf{C}}$

The comments on the left anticipate the use of each of these frames. (Actually, a perhaps more elegant s_3 for P would be the belt-selector s with $f_s(i) = 1$ for every integer i.)

4 Selective substitution grammars

Here we adopt a modified version of the definition in [1]. (Of course, '*' is the Kleene star.)

If V is a letter set, we first define $\overline{V} = \{\overline{c} : c \in V\}$, and then the homomorphism $h_V : (V \cup \overline{V})^* \to V^*$ as follows:

$$\begin{array}{rcl} h_V(c) = c & \text{for} & c \in V \\ h_V(\overline{c}) = c & \text{for} & \overline{c} \in \overline{V} \end{array}$$

A *selective substitution grammar* is a construct $G = \langle V, V', r, c_S, K\rangle$ where

- V and V' are such letter sets that $V' \subseteq V$,
- r is such a letter-refiner that every possible refinement result of each letter in V may contain only letters in V,
- the *seed-letter* c_S belongs to $V \setminus V'$,
- the *selector language* K is a subset of $(V \cup \overline{V})^*$.

We say that a word x *directly derives* a word y in G if there exists such a word $z \in K$ that $h_V(z) = x$ and when we denote $z = b_1 b_2 \dots b_n$ with $b_i \in V \cup \overline{V}$ for $1 \le i \le n$, then $y = \beta_1 \beta_2 \dots \beta_n$ with each β_i meeting the appropriate one of the following conditions:

$$\begin{array}{rcl} \beta_i = b_i & for & b_i \in V \\ \beta_i \in r(h_V(b_1 \dots b_{i-1}), a_i, h_V(b_{i+1} \dots b_n)) & for & b_i = \overline{a}_i \in \overline{V} \end{array}$$

Finally, we say that the *language generated* by G is the set of all such words $w \in V'^*$ that $\langle c_S, w\rangle$ belongs to the transitive closure of 'directly derives'. (Note the technical detail that the closure is transitive rather than reflexive-transitive.)

5 Emulating selective substitution grammars

We want to be able to show that particular tetrasystem frames are able to emulate particular classes of selective substitution grammars. In each case this means that for a given member G of the particular class, we aim at constructing a tetrasystem T_G with the particular frame meeting the following conditions:

- The language generated by G is constituted by the output words of T_G.
- There is a natural correspondence between the derivations of G and the generation processes of T_G. That is, there is a one-to-one correspondence in which every two mutual counterparts (i.e. a sequence of direct derivation steps and and a sequence of leaf expansions) are step-by-step isomorphic to each other.

5.1 Macro processors

Consider such selective substitution grammars $G = \langle V, V', r, c_S, K \rangle$ where

- V' consists of the terminals in V.
- K consists of every such $z \in (V \cup \overline{V})^*$ that meets all the following conditions:
  - z contains exactly one occurrence of the members of $\overline{V \setminus V'}$ in total.
  - z contains no occurrence of the members of $\overline{V'}$.
  - In z, no occurrence of any member of $V \setminus V'$ precedes the single occurrence of a member of $\overline{V \setminus V'}$.
- For every letter c and for every two words w_1 and w_2, the set $r(w_1, c, w_2)$ is a singleton and moreover equals $r(w_1, c, \Lambda)$ for the empty word Λ. (This is insignificant from the emulation viewpoint.)

We claim that frame M emulates this class of selective substitution grammars.

5.2 Context-sensitive Chomsky grammars

Consider such selective substitution grammars $G = \langle V, V', r, c_S, K \rangle$ where

- V' consists of the terminals in V.
- K consists of every such $z \in (V \cup \overline{V})^*$ that meets both the following conditions:
  - z contains exactly one occurrence of the members of $\overline{V \setminus V'}$ in total.
  - z contains no occurrence of the members of $\overline{V'}$.
- For every letter c and for every four words w_1, w_2, $w_1{}'$, and $w_2{}'$, we have $r(w_1, c, w_2) \subseteq r(w_1{}'w_1, c, w_2w_2{}')$. (This is insignificant from the emulation viewpoint.)

Note that we omit the requirement that both the alphabet and the production set must be finite (which is insignificant from the emulation viewpoint). (The same omission will be made below below with both pure grammars and L systems.)

We claim that frame C emulates this class of selective substitution grammars. However, the above does restrict the context-sensitive Chomsky grammars we can cope with:

- For every nonterminal c, there must in practice exist at least one context-free production, such as the dummy production $c \to c$. The addition of such dummies does not affect the language generated but introduces infinite derivations.
- In general, a context-sensitive Chomsky grammar may contain a single length-reducing rule of the form $w_1 c w_2 \to w_1 w_2$, that is, $c_S \to \Lambda$ for the seed-letter c_S. (But in this case the seed-letter is not allowed to occur in the right side in any production.) Here we effectively exclude even this possibility. This simply means that for a given context-sensitive language L, only $L \setminus \{\Lambda\}$ can be generated. (See pages 15 and 83 of [3].)

5.3 Pure grammars

Consider such selective substitution grammars $G = \langle V, V', r, c_S, K \rangle$ where

- $V = V'$ contains no terminals.
- K consists of every such $z \in (V \cup \overline{V})^*$ that contains exactly one occurrence of the members of $\overline{V}$ in total.
- For every letter c and for every four words w_1, w_2, $w_1{}'$, and $w_2{}'$, we have $r(w_1, c, w_2) \subseteq r(w_1{}'w_1, c, w_2w_2{}')$. (This is again insignificant from the emulation viewpoint.)

Note that here the seed-letter is actually a dummy placeholder introduced only when the pure grammar is converted into a selective substitution grammar. (The same holds in the L system case below. See Example 10.5 of [1] for a closely related example.)

We claim that frame P emulates this class of selective substitution grammars. However, the above restricts the context-sensitive pure grammars we can cope with, similarly as in the Chomsky grammar case.

5.4 L systems

Consider such selective substitution grammars $G = \langle V, V', r, c_S, K \rangle$ where

- $V = V'$ contains no terminals.
- K consists of every $z \in \overline{V}^*$.

We claim that frame L in an obvious sense emulates this class of selective substitution grammars, which can be interpreted as the family FPIL of Lindenmayer systems [2]. However, there is an important difference from all the earlier cases: now each direct derivation step on the selective substitution grammar side breaks into a finite sequence of leaf expansions on the tetrasystem side. The order of these leaf expansions is fully arbitrary though.

References

[1] Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*, chapter 10, pages 279–289. Springer, 1989.

[2] Lila Kari, Grzegorz Rozenberg, and Arto Salomaa. L systems. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 253–328. Springer, 1997.

[3] Arto Salomaa. *Formal Languages.* Academic Press, 1973.