Tetrasystems: A Framework for String Generation Devices

Eero Lassila

Aalto University, Helsinki, Finland
eero.lassila@aalto.fi

Abstract

Tetrasystems are proposed as a common framework for practical string generation devices such as macro processors and parametric Lindenmayer systems. They are also capable of emulating many devices well-known from formal language theory.

1 Introduction

We are especially interested in rewriting systems neither as language recognition devices nor as language generation devices but as string generation devices, like macro processors [1] or parametric Lindenmayer systems [11, 10, 9], which are proven programming tools. Therefore, we want to allow infinite alphabets (for coping with structured symbols) and unbounded context-sensitivity. (For our earlier attempts in this direction, see [5, 4].)

We propose tetrasystems as a common framework for such string generation devices. However, we also claim that tetrasystems are able to emulate large families of devices well-known from formal language theory. This notion of emulation will be explicated in the actual NWPT 2012 presentation.

Section 2 contains preliminary definitions, and section 3 then introduces tetrasystems. Finally, section 4 takes a closer look at tetrasystem-based emulation of other devices.

2 Preliminary Definitions

The trees considered here are finite, rooted, and ordered. Each tree node moreover holds a *letter*. The letters divide into *nonterminals* and *terminals*. *Words* are finite letter sequences.

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.)

Let the span between n_1 and n_2 be denoted 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, $\angle(n_1, n_1) = \angle(n_2, n_2) = \langle 0, 0, 0\rangle$.

2.2 Tree Belts and Belt-Selectors

A *belt* of a tree is simply a cross section of the tree, that is, such a subset of the tree nodes that contains exactly one ancestor of each leaf of the tree.

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

A *belt-selector* is 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. Moreover, there is a bijection between combs and belt-selectors, and so for any belt-selector s, there is a unique comb f_s.

But we still have to define how a belt-selector selects its belt, and to characterize the claimed bijection. For a given belt-selector s, for a given tree X, and for a given node n of X, the belt selected 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 $\angle(n, n') = \langle i, d, f_s(i \times d) \rangle$ 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. For example, consider the belt-selector s with $\forall i : f_s(i) = \infty$. This belt-selector is settled at a tree node if and only if all the leaves of the tree hold terminals. On the other hand, the belt-selector s^* with $\forall i : f_{s^*}(i) = \min(|i|, 1)$ is settled at every node in every tree.

2.3 Letter-Refiners

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.)

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, which must be non-empty but may be finite or countably infinite

- set V_T of terminals, which 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 $\langle s_1, s_2, s_3, s_4 \rangle$

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 given nonterminal-lettered leaf node is *fertile*, i.e. ready to be expanded, if the generation process has already proceeded sufficiently far in the other parts of the tree. That is, s_1 must be settled at the leaf.
   (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.

4 Emulation of Other Devices

We claim that tetrasystems are intrinsically capable of emulating macro processors and parametric Lindenmayer systems: one just has to choose the appropriate frame.

We also claim that choosing a suitable frame even, in a sense, enables the emulation of

1. FPIL-type Lindenmayer systems [13, 3],
2. context-sensitive Chomsky grammars [7], and
3. context-sensitive pure grammars [8, 6].

However, there are further restrictions concerning the empty word: in cases 2 and 3, the native production rules must never refine any letter into an empty word; and in cases 1 and 3, the native start words must not include the empty word.

We demonstrate the above claims by using *selective substitution grammars* [12, 2] simply for their original purpose: as a unifying rewriting framework, now for all the devices mentioned above (other than tetrasystems). This common intermediate representation facilitates the explication of our notion of emulation in the actual NWPT 2012 presentation.

The presentation will also precisely specify the frames needed in each one of the emulation cases mentioned above.

References

[1] Alfred J. Cole. *Macro Processors.* Cambridge University Press, 2nd edition, 1981.

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

[3] 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.

[4] Eero Lassila. On tree belts and belt-selectors. In Nisse Husberg, Tomi Janhunen, and Ilkka Niemelä, editors, Leksa notes in computer science, pages 47–58. Technical Report HUT-TCS-A63, Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland, October 2000. ISBN 951-22-5211-2. `http://users.ics.aalto.fi/ela/publications/tr-tbb2000.pdf`.

[5] Eero Lassila. A tree expansion formalism for generative string rewriting. Technical Report HUT-TCS-B20, Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland, June 2001. ISBN 951-22-5554-5. `http://users.ics.aalto.fi/ela/publications/tr-tefgsr2001.pdf`.

[6] Alexandru Mateescu and Arto Salomaa. Aspects of classical language theory (section 7.1). In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 175–251. Springer, 1997.

[7] Alexandru Mateescu and Arto Salomaa. Formal languages: an introduction and a synopsis (section 3.1). In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 1–39. Springer, 1997.

[8] Hermann A. Maurer, Arto Salomaa, and Derick Wood. Pure grammars. *Information and Control*, 44(1):47–72, 1980.

[9] Przemyslaw Prusinkiewicz. Simulation modeling of plants and plant ecosystems. *Communications of the ACM*, 43(7):84–93, 2000.

[10] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomír Měch. Visual models of plant development. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 535–597. Springer, 1997.

[11] Przemyslaw Prusinkiewicz and Aristides Lindenmayer. *The Algorithmic Beauty of Plants.* Springer, 1990.

[12] Grzegorz Rozenberg. Selective substitution grammars (Towards a framework for rewriting systems). Part 1: Definitions and examples. *Elektrische Informationsverarbeitung und Kybernetik*, 13(9):455–463, 1977.

[13] Grzegorz Rozenberg and Arto Salomaa. *The Mathematical Theory of L Systems.* Academic Press, 1980.