A Straightforward Translation of D0L Systems in the Declarative Data-Parallel Language $8_{1/2}$ ⋆

Olivier Michel

LRI u.r.a. 410 du CNRS
Btiment 490, Universit de Paris-Sud, F-91405 Orsay Cedex, France.
Tel: +33 (1) 69 41 76 01 email: `michel@lri.fr`

Abstract. L Systems are a mathematical formalism originally designed to model the biological growth of plants. They are based upon a parallel rewriting mechanism. We show in this paper that there exists an equivalence between the class of D0L Systems and a particular construction of programs in the $8_{1/2}$ declarative data-parallel language.

1 Introduction

$8_{1/2}$ is a declarative data-parallel language aimed at the simulation of dynamical systems. Its main structure is the *fabric* which is a *stream* [1, 2, 3] of *collections* [4]. The stream aspect of the fabric is used to describe the time axis of the simulation whereas the collection aspect of the fabric describe the space axis of the simulation. Because of space restrictions, we will not detail the $8_{1/2}$ language here. The reader should refer to the paper *"Introducing dynamicity in the data-parallel language 81/2"* in these proceedings or to [5].

An $8_{1/2}$ program is a set of equations defining the relations between the variables of the dynamical system. $8_{1/2}$ has operators to handle the collection and streams aspect of the fabric but we will only detail the concatenation, the packing and the delay operator. The *concatenation* ($a \# b$) and the *packing* ($\{\dots\}$) are two spatial operators. The relationship between the concatenation and the packing is the same as between the `cons` and `append` operations for lists: $\{\mathtt{A}, \mathtt{B}, \dots\} = \{\mathtt{A}\} \# \{\mathtt{B}\} \# \dots$ The *delay* operator, which is a temporal operator, shifts an entire stream to give at the current instant of the simulation, the previous stream value. For example, if we have $S = > 0, 1, 2, 3, \cdots >$ (that is, at time instant 0, the value of S is 0, at time instant 1, the value is 1, etc.), then the value of $\$S$ is $> \perp, 0, 1, 2, 3 >$. The definition of an $8_{1/2}$ fabric is usually made of two parts: 1) the definition at some specific instants expressed by quantified equations; 2) the definition for all other instants. For example, $T@0 = 1$ means that the value of the fabric at time instant 0 is 1 and $T = \$T + 1$ when Clock means that for all other instants, the current value of T is the previous one increased by one. The time instants where this increase occurs are specified by the "when Clock" expression.

⋆ This research is partially supported by the operation "Programmation parallle et distribue" of the french "GDR de programmation".

2 L Systems and 81/2: From Transformation Rules to Equations Between Fabrics

2.1 Lindenmayer Systems

An *L System* is a *parallel* rewriting system (every production rule that might be used at each derivation state is used simultaneously) developed by A. Lindenmayer in 1968 [6].Since their initial design for the specification of growing processes [6], L Systems have proved to be an effective paradigm (for example, in the field of image processing, symbolic processing [7], cellular development modelisation [8], parallel computation model [9], ...).

We will restrict ourselves to the simplest form of L Systems: *D0L Systems.* The *D* letter stands for deterministic, that is there exists at most a single production rule for each element of Σ. Therefore the derivation sequence is unique while in non-deterministic L Systems, since there can be more than one production rule applied at each derivation state, there exists more than one derivation sequence. The numerical argument of the L System gives the number of interactions in the rewriting process; therefore a 0L System is a context free L System (whereas an *nL System* is context sensitive with n interactions). A *D0L System* is a triple $G = (\Sigma, h, \omega)$ where Σ is an alphabet, h is a finite substitution on Σ (into the set of subsets of Σ^*) and ω, referred to as the axiom, is an element of Σ^+. The derivation $H(w)$ of a word $w \in \Sigma^*$ is defined in the following manner:

$$\begin{array}{lll} H(a) & = h(a) & \text{for } a \in \Sigma \\ H(u \centerdot v) & = H(u) \centerdot H(v) & \text{for } u, v \in \Sigma^+ \end{array}$$

that is, H is the homomorphism extending h.

Our goal is to show that D0L Systems can be mimicked in the 81/2 language *without adding any new features* to the current definition of the language. We sketch in the next section the relation between L Systems and the 81/2 language.

2.2 A Translation of L Systems into 81/2

We are interested in describing the derivation mechanism involved in the translation of the L Systems into the 81/2 language. Fabrics with a static structure cannot describe a phenomenon that grows in space (like plants). To describe those structures, we need dynamically structured fabrics. We do not need to introduce new operators, current fabrics definition already enables the construction of dynamically shaped fabrics. The translation of a D0L System in 81/2 relies on the following: 1) a word is a *vector of elements* of Σ; 2) a fabric W is associated to the axiom ω, and the value $W(t)$ of W at time t is $H^t(\omega)$.

To implement this in 81/2, each rule defining h is translated into the following fabric definition:

$$a \rightarrow a_1 \, \ldots \, a_n \text{ gives } \begin{cases} A@0 = {'a'} \\ A = \$A_1 \,\#\, \ldots \,\#\, \$A_n \text{ when } Clock \end{cases} \tag{1}$$

and a word $w = b_1 \ldots b_n$ gives the fabric definition $W = B_1 \# \ldots \# B_n$.

We prove that for any word w, $W(t) = H^t(w)$. First, we prove by recurrence on t that $A(t) = H^t(a)$. This is obviously true for $t = 0$. Suppose the property true for $t' < t$. Then,

$$\begin{aligned}
A(t) &= (\$A_1)(t) \# \ldots \# (\$A_n)(t) \\
&= A_1(t-1) \# \ldots \# A_n(t-1) && \text{(definition of \$)} \\
&= H^{t-1}(a_1) \,.\, \ldots \,.\, H^{t-1}(a_n) && \text{(by hypothesis)} \\
&= H^{t-1}(a_1 \ldots a_n) && \text{(H homomorphism)} \\
&= H^{t-1}(H(a)) \\
&= H^t(a)
\end{aligned}$$

Secondly, $W(t) = B_1(t) \# \ldots \# B_n(t) = H^t(b_1) \,.\, \ldots \,.\, H^t(b_n) = H^t(b_1 \ldots b_n) = H^t(w)$. □

It is therefore possible, using only the # and delay operators to program in 81/2 the whole class of D0L Systems. Note that it is possible to describe the whole class of D0L Systems in 81/2 (even non propagating L Systems). We consider each transition rule as a recursive 81/2 equation. Parallelism is present in this "rewriting" mechanism, in the form of:

- *data-parallelism:* actually no data-parallelism is directly involved in the recursive operations except that the data structure involved (collections) are distributed (so that multiple concatenations can be implemented in parallel). However, the mechanism presented here is very general. L Systems are a *model* used to describe phenomena. At each step, data-parallel (reductions, scans, etc.) operations can be performed locally (on each collection) to reflect a particular behaviour taking place after the rewriting.

- *control-parallelism:* control parallelism, depends on the data dependencies between the equations involved in the axiom and in each rewriting rule. There are no dependencies because each fabric refers to past values of other fabrics. Therefore, the computation of the current value of each fabric can be fully processed in parallel.

An implementation based solely on the exploitation of control-parallelism will map the computations of a fabric to some processors. Assume $A = \$B \# \$C$; the "owner computes rule" (as in HPF) assigns the computation of the concatenations to the processor owning A and implies the communication of the past values of B and C from the processors owning B and C to the one of A. This can be avoided using locality, that is assigning B and/or C to the processor owning A. This is in contradiction with the maximal parallelism exploitation but minimizes the communications. A mapping and scheduling heuristic, taking into account data-communications, is presented in [10].

3 Conclusions and Future Work

We have described in this paper the translation of L Systems into the declarative data-parallel language 81/2. Each transition rule of a D0L System can easily be rewritten into equations between streams and collections. Following this basic scheme, it is very simple to implement even more complex programs in $8_{1/2}$ (like the graph generation scheme using G0L Systems found in [11]). We are currently working on the implementations of more general L Systems and especially the class of parametric L Systems.

Acknowledgments. The author wishes to thank the members of the $8_{1/2}$ team: Jean-Louis Giavitto, Jean-Paul Sansonnet, Dominique De Vito and Abderrahmane Mahiout for many stimulating discussions. He also thanks Dan Truong for careful reading the paper.

References

1. Edward A. Ashcroft, A. Faustini, R. Jagannathan, and W. Wadge. *Multidimensional Programming*. Oxford University Press, February 1995. ISBN 0-19-507597-8.
2. P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: A declarative language for programming synchronous systems. In *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 178–188, Munich, West Germany, January 21–23, 1987. ACM SIGACT-SIGPLAN, ACM Press.
3. P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier. Signal, a dataflow oriented language for signal processing. *IEEE-ASSSP*, 34(2):362–374, 1986.
4. G. E. Blelloch and G. W. Sabot. Compiling collection-oriented languages onto massively parallel computers. *Journal of Parallel and Distributed Computing*, 8:119–134, 1990.
5. Olivier Michel. Design and implementation of $8_{1/2}$, a declarative data-parallel language. *special issue on Parallel Logic Programming in Computer Languages*, 1996. (to appear).
6. A. Lindenmayer. Mathematical models for cellular interactions in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.
7. N. S. Goel and M. D. Goodwin. Symbolic computation using L systems. *Applied mathematics and computation*, 42:223–253, 1991.
8. M. J. M. de Boer, F. D. Fracchia, and P Prusinkiewicz. A model for cellular development in morphogenetic fields. In G. Ronzenberg and A. Salomaa, editors, *Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, pages 351–370. Springer Verlag, February 1992.
9. P. Prusinkiewicz and J. Hanan. L systems: from formalism to programming languages. In G. Ronzenberg and A. Salomaa, editors, *Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, pages 193–211. Springer Verlag, February 1992.
10. A. Mahiout, J.-L. Giavitto, and J.-P. Sansonnet. Distribution and scheduling data-parallel dataflow programs on massively parallel architectures. In *SMS-TPE'94: Software for Multiprocessors and Supercomputers*, Moscow, September, 1994. Office of Naval Research USA & Russian Basic Research Foundation.

11. E. J. W. Boers. Using L systems as graph grammars: G2L systems. Technical Report TR 95-30, Leiden University, 1995.

This article was processed using the LaTeX macro package with LLNCS style