Journal of Universal Computer Science, vol. 12, no. 10 (2006), 1455-1463
submitted: 8/3/06, accepted: 24/10/06, appeared: 28/10/06 © J.UCS

Time-varying H Systems Revisited

Remco Loos
Research Group on Mathematical Linguistics
Rovira i Virgili University
Pça Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: remcogerard.loos@urv.net

Abstract: We cast a new look on time-varying distributed H systems. In their original definition, where only new strings are passed to the next component, this language definition in itself is already enough to obtain computational completeness. Here, we consider two types of time-varying H systems with weaker language definitions, based on the usual definition of splicing systems: The next generation of strings consists of the union of all existing strings and the newly created strings. We show that if all strings, both old and new, are passed to the next component these systems are regular in power. If however, the new strings pass to the next component and the existing ones remain accessible to the current one, we prove that systems with 4 components are already computationally complete.
Key Words: DNA computing, Molecular computing, Splicing systems
Category: F.1.1, F.4.2

1 Introduction

Time-varying distributed H systems were introduced in [9]. They have a distributed architecture in which different sets of splicing rules are used periodically. In [9] it was shown that time-varying distributed H systems are computationally complete.

Moreover, in a series of papers, including [10], [7] and [5], the degree (i.e. the number of different sets of splicing rules) of the time-varying H systems needed to obtain computational completeness has been decreased progressively. Finally, in [4], it was shown that time-varying distributed H systems of degree 1 can generate all recursively enumerable languages. Such systems are really no longer distributed nor time-varying, having only a single set of splicing rules.

This result can be explained by the way the language is defined in time-varying H systems: From one splicing step to the next, only the newly created strings are kept. The result in [4] shows that this way of defining the splicing language alone is sufficient to obtain computational completeness. Recently ([2] and [3]), this definition has also been studied in the context of basic finite splicing systems.

The fact that the computational power of the language definition alone makes the distributed architecture superfluous, suggests that it may be interesting to consider time-varying H systems with a weaker language definition. A candidate for such a definition is easy to find, since in basic splicing systems as well as in practically all splicing formalisms another definition is used. In this definition, the strings passed to the next step consist of the union of the strings present at this step and the new strings created by applying the splicing rules. This definition is considerably weaker from a computational point of view. In the case of basic extended splicing systems the definition which only conserves the new strings yields computationally complete systems, whereas defining the new generation by the union of old en new strings gives systems of only regular power.

In this paper, we introduce two new language definitions for time-varying H systems. In the first one, all strings, both new and already present, are passed to the next component. In the second one, the new strings are passed to the next component, whereas the existing strings remain accessible to the current set of rules. We know that both of these definitions are weaker than the original one, since in both cases systems of degree one are equivalent by definition to extended finite H systems, which generate exactly all regular languages.

After reviewing the basic definitions, we formally define the two new variants. We then prove that the first variant generates only regular languages for any degree (any number of components). For the second variant, we show that systems of degree of at least 4 are computationally complete.

2 Basic Definitions and Notation

A splicing rule over V is a string $u_1\#u_2\$u_3\#u_4$, with $u_1, u_2, u_3, u_4 \in V^*$, and $\$, \#$ special symbols not in V.

For a splicing rule $r = u_1\#u_2\$u_3\#u_4$ and $x, y, w, z \in V^*$, we write

$$
\begin{aligned}
(x, y) \vdash_r (w, z) \text{ iff } & x = x_1u_1u_2x_2, \\
& y = y_1u_3u_4y_2, \\
& z = x_1u_1u_4y_2, \\
& w = y_1u_3u_2x_2, \\
& \text{for some } x_1, x_2, y_1, y_2 \in V^*.
\end{aligned}
$$

A *splicing scheme* is a pair (V, R), where V is an alphabet and R a set of splicing rules over V. For a splicing scheme $h = (V, R)$ and a language L over V we define

$$
\begin{aligned}
\sigma(L) = \{w \in V^* \mid & (w_1, w_2) \vdash_r (w, w') \text{ or } (w_1, w_2) \vdash_r (w', w) \\
& \text{for some } w_1, w_2 \in L, \text{ some } w' \in V^*, \text{ and some rule } r \in R\}.
\end{aligned}
$$

Given a splicing scheme h and an initial language L, the splicing language

$\sigma_h^*(L)$ is defined as follows.

$$\sigma_h^0(L) = L,$$
$$\sigma_h^{i+1}(L) = \sigma_h^i(L) \cup \sigma_h(\sigma_h^i(L)), i \geq 0,$$
$$\sigma_h^*(L) = \bigcup_{i\geq 0} \sigma_h^i(L).$$

When the splicing scheme is clear, we omit the subscript.

A *splicing system* or *H system* is a construct

$$H = (V, A, R),$$

where V is an alphabet, $A \subseteq V^*$ is the initial language, and R is a set of splicing rules over V. The generated language is defined as $L(H) = \sigma^*(A)$.

An *extended H system* is a construct

$$\gamma = (V, T, A, R),$$

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is the initial language, and R is a set of splicing rules over V. The language generated by γ is defined as

$$L(\gamma) = \sigma^*(A) \cap T^*.$$

For H systems with a finite set of rules and a finite initial language, i.e. A and R are both finite sets, it is shown in [13] that they generate only regular languages.

A *time-varying distributed H system*(of degree n) is a construct:

$$D = (V, T, A, R_1, R_2, ..., R_n),$$

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and components R_i are finite sets of splicing rules over V, $1 \leq i \leq n$.

At each moment $k = n \cdot j + i$, for $j \geq 0$, $1 \leq i \leq n$, only the component R_i is used for splicing the currently available strings. Specifically, we define

$$L_0 = A,$$
$$L_k = \sigma_{h_i}(L_{k-1}), \textit{ for } i \equiv k (\textit{mod } n), k \geq 1, 1 \leq i \leq n, h_i = (V, R_i).$$

Therefore, from step $k-1$ to the next step, k, one passes only the result of splicing the strings in L_{k-1} according to the rules in R_i. The strings in L_{k-1} that cannot enter a splicing rule are removed when passing to L_k.

The language generated by D is, by definition:

$$L(D) = (\bigcup_{k\geq 0} L_k) \cap T^*.$$

Finally, we use the notations REG and RE to denote the families of regular and recursively enumerable languages, respectively.

3 New Definitions

Now we can formally define the systems presented informally in the introduction. In the first type, which we call time-varying H systems with *full transfer*, all strings, both new and already present, are passed to the next component.

Definition 1. A *time-varying distributed H system (of degree n) with full transfer* is a construct:

$$D = (V, T, A, R_1, R_2, ..., R_n),$$

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and components R_i are finite sets of splicing rules over V, $1 \leq i \leq n$. We define

$$L_0 = A,$$
$$L_k = L_{k-1} \cup \sigma_{h_i}(L_{k-1}), \textit{ for } i \equiv k (\textit{mod } n), k \geq 1, 1 \leq i \leq n, h_i = (V, R_i).$$

The language generated by D is, by definition:

$$L(D) = (\bigcup_{k \geq 0} L_k) \cap T^*.$$

The definition of the second type is slightly more intricate. Here, the newly created strings are passed to the next component, and the strings already present before applying the rules, remain accessible to the current component. Formally, we define

Definition 2. A *time-varying distributed H system (of degree n) with partial transfer* is a construct:

$$D = (V, T, A, R_1, R_2, ..., R_n),$$

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and components R_i are finite sets of splicing rules over V, $1 \leq i \leq n$. We define

$$L_{-k} = \emptyset, \ k \geq 1,$$
$$L_0 = A,$$
$$L_k = \sigma_{h_i}(\bigcup_{j \geq 0} L_{k-j \cdot n-1}), \textit{ for } i \equiv k (\textit{mod } n), k \geq 1, 1 \leq i \leq n, h_i = (V, R_i).$$

The language generated by D is, by definition:

$$L(D) = (\bigcup_{k \geq 0} L_k) \cap T^*.$$

The idea of distribution of strings to different components is reminiscent of communicating distributed H systems (see [12]), also known as test tube systems. But in these systems, components are full systems rather than sets of rules, and the contents of each component are redistributed to all components according to filters. This means our approach does not carry over naturally to these systems. Also the distinction between new and existing strings, which is an intrinsic part of time-varying H systems, has no natural expression there. This is still true for alternative types of communicating distributed H systems incorporating some aspects of time-varying systems ([1],[14].

In what follows, we use the abbreviations $FT\text{-}TVH_n$ and $PT\text{-}TVH_n$, $n \geq 1$ to denote the families of languages generated by time-varying H systems with full and partial transfer respectively, and of degree at most n. $FT\text{-}TVH_*$ and $PT\text{-}TVH_*$ correspond to the families of languages generated by such systems of any degree.

4 Computational Power

We start the investigation of the computational power of these systems with the following observation.

Theorem 3. $FT\text{-}TVH_1 = PT\text{-}TVH_1 = REG.$

Proof: It is easily verified that with only one component, these systems reduce to systems equal by definition to extended H systems with a finite set of rules and a finite initial language. The theorem follows from the the characterization of these systems in [11]. □

Theorem 4. $FT\text{-}TVH_* = REG.$

Proof: To show the theorem we will prove that any time-varying H system with full transfer $\Gamma = (V, T, A, R_1, ..., R_n)$ generates the same language as the extended finite H system $H = (V, T, A, R)$, where

$$R = \bigcup_{i \geq 1}^{n} R_i.$$

As before, the theorem then follows from the characterization of extended finite H systems in [11].

From the definition it is obvious that $L(\Gamma) \subseteq L(H)$. For the other direction, we show by induction that for all $i \geq 0$,

$$\sigma_h^i(A) \subseteq \bigcup_{k \geq 0} L_k,$$

where $h = (V, R)$ and L_k is defined for Γ as in Definition 1. For $i = 0$, $\sigma_h^0(A) = L_0 = A$. Now, assuming the assertion is true for i, we show it is true for $i + 1$.

Suppose that $x, y \in \sigma_h^i(A)$ and that $x, y \vdash_r w$ for some $r \in R$. By the induction hypothesis, $x, y \in \bigcup_{k \geq 0} L_k$ and, by the definition of H, $r \in R_j$ for some $1 \leq j \leq n$. This means that there exists an $s \geq i$ such that $s \equiv j (mod\ n)$. Then w will be in L_{s+1} and $w \in \bigcup_{k \geq 0} L_k$. □

Theorem 5. *PT-$TVH_4 = RE$.*

Proof: Consider a type-0 grammar $G = (N, T, S, P)$. We denote by $\alpha_1, ..., \alpha_{n-1}$ the symbols in $N \cup T$. Let $\alpha_n = F$ be a new symbol. Let $u_j \rightarrow v_j$ for $n+1 \leq j \leq m$ denote the rules in P and assume we have $u_i = v_i = \alpha_i$ for $1 \leq i \leq n$.

We construct the time-varying H system with partial transfer
$\Gamma = (V, T, A, R_1, R_2, R_3, R_4)$, with

$$
\begin{aligned}
&V = N \cup T \cup \{X, Y, Z, Z', Z_0, Z_0', F\} \cup \{X_i, Y_i \mid 0 \leq i \leq m\}\\
&\cup \{Z_j \mid 1 \leq j \leq m\},\\
&A = \{XSFY, XZ', ZY, Z_0, Z_0'\} \cup \{X_j Z', ZY_j \mid 0 \leq j \leq m\}\\
&\cup \{X_j v_j Z_j' \mid 1 \leq j \leq m\},\\
&R_1 = Q \cup \{\#Y\$\#Y, X_0\#\$X_0\#\} \cup \{\#Y_j\$\#Y_j \mid 1 \leq j \leq m\},\\
&R_2 = Q \cup \{X\#\$X\#, \#Y_0\$\#Y_0\} \cup \{X_j\#\$X_j\# \mid 1 \leq j \leq m\},\\
&R_3 = Q \cup \{\#Y_0\$Z\#Y, \#FY_0\$Z_0\#\} \cup \{\#u_j Y\$Z\#Y_j \mid 1 \leq j \leq m\}\\
&\cup \{\#Y_j\$Z\#Y_{j-1} \mid 1 \leq j \leq m\},\\
&R_4 = \{X_0\#\$X\#Z', X_0\#\$\#Z_0' \mid 1 \leq j \leq m\}\\
&\cup \{X\#\$\#Z_j, X_j\#\$X_{j-1}\#Z' \mid 1 \leq j \leq m\},
\end{aligned}
$$

where

$$Q = \{Z\#\$Z\#, \#Z'\$\#Z', Z_0\#\$Z_0\#, \#Z_0'\$\#Z_0'\} \cup \{\#Z_j'\$\#Z_j' \mid 1 \leq j \leq m\}.$$

This system simulates G using the rotate-and-simulate technique first used in [8]. We simulate a rule application by a splicing at the right end of the string. To ensure that all symbols in the string can be rewritten we circularly permutate the current sentential form. Here, as in [7], simulation and rotation are done in the same way: A suffix of the current string is removed and the corresponding string is added to the left. This is done by the rules in R_3 and R_4. The rules in R_1 and R_2 ensure these operations are applied correctly. We will prove the two inclusions $L(G) \subseteq L(\Gamma)$ and $L(G) \supseteq L(\Gamma)$.

1. $L(G) \subseteq L(\Gamma)$. Consider a string of the form XwY in component 1. This string encodes the current sentential form of G. Initially $w = SF$. This string is passed unchanged by the rule $\#Y\$\#Y$ to R_2 where it is again passed unchanged

to R_3 using the rule $X\#\$X\#$. Note that all other axioms are passed to components 2, 3 and 4 by the rules in Q. In R_3, if $w = w'u_i$ for some $1 \le i \le m$ we can perform

$$(Xw'u_iY, ZY_i) \vdash (Xw'Y_i, Zu_iY).$$

The string $Xw'Y_i$ is passed to R_4 where we can apply

$$(X_iv_iZ'_i, Xw'Y_i) \vdash (X_iv_iw'Y_i, XZ'_i).$$

From R_1 this string is passed unchanged to R_2 and R_3 (by rules $\#Y_j\$\#Y_j$ and $X_j\#\$X_j\#$ respectively). In R_3 the subscript of Y is decreased by 1 using a rule $\#Y_j\$Z\#Y_{j-1}$. The resulting string $X_iv_iw'Y_{i-1}$ is passed to R_4 where by applying a rule $X_j\#\$X_{j-1}\#Z'$ the subscript of X is decreased by 1. Iterating this process, we get to a string of the form $X_0v_iw'Y_0$. This string passes through R_1 and R_2 unchanged. In R_3 Y_0 is replaced by Y and in R_4 we substitute X_0 by X. Thus we have passed from $Xw'u_iY$ to $Xv_iw'Y$. If $1 \le i \le n$ we have rotated one symbol, since $u_i = v_i = \alpha_i \in N \cup T \cup \{F\}$. If $n+1 \le i \le m$ we have simulated the application of the rule $u_i \to v_i$. Iterating this procedure, we can simulate any rule of G at any position. So, if in G $S \Rightarrow^* x_1x_2$, we can produce the string Xx_2Fx_1Y in Γ, and by circular permutation also $X_0x_1x_2FY_0$. In R_3 we can apply the rule $\#FY_0\$Z_0\#$ to obtain $X_0x_1x_2$: This string gets to R_4 where we remove the X_0 with the rule $X_0\#\$\#Z'_0$. If the resulting string x_1x_2 is in T^*, then $x_1x_2 \in L(\Gamma)$. So, $L(G) \subseteq L(\Gamma)$.

2. $L(G) \supseteq L(\Gamma)$. To see that Γ does not produce any strings not in $L(G)$, note that to continue the simulation of G, the strings should be passed to the next component. The rules of Γ are such that strings that remain in the current component do not interfere. If these strings encode valid simulations of G they remain unchanged and can resume the simulation at a later moment. All invalid simulations will be 'trapped' in one component and will not lead to strings in T^*.

Specifically, suppose we have performed in R_3

$$(Xw'u_iY, ZY_i) \vdash (Xw'Y_i, Zu_iY)$$

and in R_4

$$(X_jv_jZ'_j, Xw'Y_i) \vdash (X_jv_jw'Y_i, XZ'_j)$$

for some $1 \le i, j \le m$. As mentioned before, the axioms except $XSFY$ are available in all components thanks to the rules in Q.

A string $X_jv_jw'Y_i$ with $i \ge 1$ can pass to R_2 by the rule $\#Y_i\$\#Y_i$. Then it passes to R_3 using $X_j\#\$X_j\#$ provided that $j \ge 1$. In R_3 we decrement the subscript of Y and in R_4 the subscript of X. These are the only operations possible in these components. This process is repeated until some subscript reaches zero.

Suppose that after R_4 we have a string of the form X_0wY_k for $k \geq 1$. This string is passed to R_2 by the rules $X_0\#\$X_0\#$ and $\#Y_j\$\#Y_j$. Now, there is no rule in R_2 that can be applied to the string. So, it will remain in this component and never yield a terminal string. If after R_4 we have a string of the form X_kwY_0 for $k \geq 1$, no rule in R_1 can be applied to it. Thus, this string will not derive a word in T^*. With subscript equal to zero, only strings of the form X_0wY_0 can pass through R_1 and R_2. Then in R_3 and R_4, X_0 and Y_0 are replaced by X and Y and a new simulation or rotation step can start.

So, the only strings that can continue the simulation are those where $i = j$ at the moment that we start to decrement the subscript. This means we have correctly simulated a rule in P or correctly rotated a symbol.

All strings that are produced as a by-product do not lead to terminal strings. All these strings contain the symbol Z or Z', so they are passed to all components by the rules in Q. But they do not interfere with the simulation process. As an example, consider a string Zu_iY produced in R_3. When it returns to R_3, we can replace u_iY by Y_i and then, also in R_3, decrease the subscript. When reaching Y_0, it can be removed or rewritten by Y. The strings ZY and ZY_i are axioms, and other strings of the form Zw or ZwY cannot enter in any splicing rule with strings of the form $Xw'Y$.

Finally, the symbol Y_0 can only be removed when it follows the symbol F, which guarantees that only the correct permutation yields a terminal string. Thus, the only terminal strings that are generated by Γ correspond to strings in $L(G)$. This concludes the proof that $L(G) \supseteq L(\Gamma)$. □

5 Conclusions and Further Research

We studied two variants of time-varying H systems, both with a weaker language definition than the original one. In the first type, which we called *with full transfer*, all strings, both new and already present, are passed to the next component. We showed that for any degree, these systems generate exactly all regular languages. The second variant, time-varying H systems *with partial transfer* was shown to be universal for systems of degree at least 4. In these systems, the new strings created by applying the splicing rules are passed to the next component and all strings present before rule application remain accessible to the current set of rules. This last type gives rise to an interesting open question. If time-varying H systems with partial transfer of degree 4 generate all RE languages and those of degree 1 generate only regular languages, what is the power of systems of degree 2 and 3? We conjecture that systems of degree 2 can be shown to be universal, by using the technique of forcing the correct derivation to go through all components, as we did in Theorem 5. As an example, we give a very simple time-varying H system with partial transfer of degree 2 that generates a non-regular language.

$\Gamma = (V, T, A, R_1, R_2)$, with

$V = \{a, b, X, Y, Z\}$,

$T = \{a, b\}$,

$A = \{XabY, ZbY, XaZ\}$,

$R_1 = \{X\#a\$Xa\#Z, X\#a\$\#ZbY, \#bY\$XaZ\#\} \cup \{ZbY\#\$ZbY\#\}$,

$R_2 = \{b\#Y\$Z\#bY\}$.

The reader can verify that $L(\Gamma) = \{a^n b^n \mid n \geq 1\}$.

Acknowledgements

Thanks to the anonymous reviewers for useful comments which helped to improve this paper. This work was supported by Research Grant BES-2004-6316 of the Spanish Ministry of Education and Science.

References

1. P. Frisco, C. Zandron, *On variants of communicating distributed H systems*, Fundamenta Informaticae 21, 1001-1012 (2001)
2. T. Harju, M. Margenstern, *Splicing systems for universal Turing machines*, Proc. 10th Internat. Meeting on DNA Based Computers (DNA10, Milan; C. Ferretti et al., eds), Lecture Notes in Computer Science 3384, Springer-Verlag, 151-160 (2005)
3. R. Loos, V. Mitrana, *Non-preserving splicing with delay*, submitted (2005)
4. M. Margenstern, Y. Rogozhin, *Time-varying distributed H systems of degree 1 generate all recursively enumerable languages*, Words, Semigroups, and Transductions (M. Ito, Gh. Paun, S. Yu eds.), World Scientific Publishing, Singapore, 329-340 (2001)
5. M. Margenstern, Y. Rogozhin, S. Verlan: *Time-varying distributed H systems of degree 2 can carry out parallel computations*, DNA8, Lecture Notes in Computer Science, v. 2568, Springer-Verlag, 326-336 (2003)
6. M. Margenstern, Y. Rogozhin, S. Verlan, *Time-varying distributed H systems with parallel computations: the problem is solved*, DNA9, Lecture Notes in Computer Science 2943, Springer-Verlag, 48-53 (2004)
7. A. Paun, *On time-varying H systems*, Bulletin of the EATCS, 67, 157-164 (1999)
8. Gh. Paun, *Regular extended H systems are computationally universal*, J. Automata, languages, Combinatorics, 1, 1, 27-36 (1996)
9. Gh. Paun, *DNA Computing; Distributed splicing systems*, Structures in Logic and Computer Science (J. Mycielski, G, Rozenberg, A. Salomaa eds.), Lecture Notes in Computer Science 1261, Springer-Verlag, 309-327 (1997)
10. Gh. Paun, *DNA computing based on splicing: universality results*, Proc. of Second Intern. Colloq. Universal Machines and Computations, Metz, Vol I, 67-91 (1998)
11. Gh. Paun, G. Rozenberg, A. Salomaa, *Computing by splicing*, Theoretical Computer Sci., 168, 321-336 (1996)
12. Gh. Paun, G. Rozenberg, A. Salomaa, *DNA computing - New computing paradigms*, Springer-Verlag, Berlin (1998)
13. D. Pixton, *Regularity of splicing languages*, Discrete Appl. Math., 69, 101-124 (1996)
14. S. Verlan *Communicating distributed H systems with alternating filters*, Aspects of Molecular Computing (N. Jonoska, Gh. Paun, G. Rozenberg eds.), Lecture Notes in Computer Science 2950, Springer-Verlag, 367-384 (2004)