Computing Beyond the Turing Limit Using the H Systems

Cezar Câmpeanu[1,⋆] and Andrei Păun[2,⋆]

[1] Department of Computer Science and Information Technology,
University of Prince Edward Island,
Charlottetown, P.E.I., Canada C1A 4P3
ccampeanu@upei.ca
[2] Department of Computer Science,
College of Engineering and Science,
Louisiana Tech University, Ruston, P.O. Box 10348,
Louisiana, LA-71272 USA
apaun@latech.edu

Abstract. We introduce a new variant of the heavily studied model of H systems. The new variant will use an external factor to determine the set of the active splicing rules. We improve the best known universality result for time-varying H systems with respect to the diameter of such a system and we prove that if the function recording the behavior of the external factor is uncomputable so is the newly defined model, thus exceeding the Turing barrier. We also construct an universal system that is also more powerful than the Turing Machines.

1 Introduction

For more than half a century the Turing machine model of computation was used to define what it means to "compute" or 'to be "computable", notions which are the foundations of the modern theory of computing. In the last few years several researchers have started to look "beyond Turing", i.e., trying to find models of computation that would be able to compute more than a Turing machine (see [1], [3], [4], [15]). This is a very important endeavor, since it means that once such model finds its implementation we would have a computer more powerful than any silicon computer as we know them today. One might argue that this is not possible, and we would like to point out that the speed of our current computers is many times higher than the speed of the biological systems (our brain, as an example), and still we can perform much better/accurate pattern matching than computers.

This observation was the starting point of our work and we will present a model that is capable to compute non-Turing computable languages. The pro-

⋆ The first author is supported by NSERC grant UPEI-600089 and the second author is supported by a LATECH-CenIT grant.

C. Ferretti, G. Mauri and C. Zandron (Eds.): DNA10, LNCS 3384, pp. 24–34, 2005.

posed model is based on the H systems theory, but it will also take in consideration the environment of the system (an idea borrowed from the P systems in which the environment plays a vital role in the computation). None of the models known so far in the area of H systems try to deal with the computing power over Turing computability, therefore, we think it is of real interest to design a system able to compute languages beyond Turing's limit in this framework.

The reason for doing so is that in real life we can see that many phenomena cannot be explained, mostly because real life systems are not isolated, but they interact with the outside world. Temperature, light, radiation, or simply substance contamination can influence the chemical reactions necessary to recombine DNA strands. An uncontrolled chemical reaction may happen in a normal body when cancerous cells may develop. This kind of reactions are mostly dependent on external factors. For example, to keep a temperature constant is a very difficult task for real life systems, since they need very good insulation. Variation of temperature can speed up or slow down chemical reactions and sometimes temperature can behave like an activator/inhibitor for some reactions. Therefore, it is natural to consider systems where an external factor like temperature can influence DNA splicing.

For an H-system this means we have to add a function τ depending on time, and this function will decide what sets of splicing rules can or cannot be applied at some moment. Hence, the appropriate model for simulating this behavior is a time varying H system where the rules are not applied periodically, but depending on the external conditions, i.e., depending on the value of function τ. Without restricting the generality, we can consider τ as a function from $\mathbf{N}$ to $\{1, \ldots, n\}$, where n is the number of sets of rules that can be activated/deactivated. So, at the moment t, we can consider that only the $\tau(t)$ set of rules can be used.

Hence, we have a new model of τ-time varying H systems. Some natural questions will arise:

1. Can this model be universal? In other words, can we reach the computational power of Turing Machines or other equivalent devices?
2. Is the power of this systems limited to Turing machines? Is the function τ able to add more power to these machines?

In this paper we prove that we can construct an universal system influenced by the temperature and moreover, the computational power will exceed the one of Turing machines in the case that τ is an uncomputable function.

We also improve the best known result for the "usual" time-varying H systems in terms of their diameter/radius of their splicing rules in an effort to bring these systems more closely to an actual implementation. The diameter of the splicing rules is important since the restriction enzymes (that are modeled by a splicing rule) recognize usually small sites on the DNA strand; having arbitrary alphabets in our systems means actually that groups of nucleotides would have to codify one single letter. So, it is clear that there is a significant difference between splicing rules of diameter (3,2,2,2) and (4,5,4,4). Assuming that the alphabet of the system is 10 letters long, then each letter has to be codified with at least 2

nucleotides, making the site recognized by the restriction enzyme modeling the first half of the splicing rule of size 10 for diameter (3,2,2,2) or 18 for diameter (4,5,4,4).

2 Definitions

Let V be an alphabet and $\#, \$$ two symbols not in V. A splicing rule over V is a string of the form $r = u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$ (V^* is the free monoid generated by V; the empty string is denoted by λ; for formal language details we refer to [14]).

For $x, y, w, z \in V^*$ and r as above, we write

$$(x, y) \vdash_r (w, z) \text{ if and only if } x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,$$
$$w = x_1u_1u_4y_2,\ z = y_1u_3u_2x_2,$$
$$\text{for some } x_1, x_2, y_1, y_2 \in V^*.$$

A pair $\sigma = (V, R)$, where V is an alphabet and R is a set of splicing rules, is called an *H scheme.*

For an H scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$, we define:

$$\sigma(L) = \{w \in V^* \mid (x, y) \vdash_r (w, z) \text{ or } (x, y) \vdash_r (z, w), \text{ for some } x, y \in L,\ r \in R\}.$$

A *periodically time-varying H system* (of degree $n, n \geq 1$) is a construct

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

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

Each set $R_i, 1 \leq i \leq n$, is called a *component* of Γ.

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

$$L_0 = A,\ L_k = \sigma_i(L_{k-1}), \text{ for } i \equiv k(\bmod\ n), k \geq 1, \text{ where } \sigma_i = (V, R_i), 1 \leq i \leq n.$$

The language generated by Γ is defined by $L(\Gamma) = (\bigcup_{k\geq 0} L_k) \cap T^*$.

One of the aims of this paper is to consider a precise estimation of the size of the splicing rules in a time-varying H system, in the sense of [12]. Namely, for a system $\Gamma = (V, T, A, R_1, R_2, \ldots, R_n)$ we define $dia(\Gamma) = (n_1, n_2, n_3, n_4)$, where $n_i = \max\{|u_i| \mid u_1\#u_2\$u_3\#u_4 \in R_j, 1 \leq j \leq n\},\ 1 \leq i \leq 4$. We say that $dia(\Gamma)$ is the *diameter* of Γ.

The family of languages generated by time-varying H systems with at most n components having the diameter less than or equal to $(n_1, n_2, n_3, n_4), n_i \geq 0$, $1 \leq i \leq 4$, is denoted by $TVH_n(n_1, n_2, n_3, n_4)$ (the vector ordering is the natural componentwise one). The family of languages generated by time-varying H systems with at most n components, $n \geq 1$, and of an arbitrary diameter is denoted by TVH_n. By RE we denote the family of recursively enumerable languages (for definitions and properties of the RE sets, see [5]).

As we have already mentioned, we consider an external factor recorded by a function τ that will influence the work of the system. Without loosing the generality, we may assume that this external factor is the temperature. We will consider in the current paper a new variant of the H systems, namely *time-varying H systems with temperature*; we assume that in the system there are n sets of splicing rules, each set being activated/deactivated by the temperature of the environment. To model the temperature activation/deactivation, we will associate with each such system a number from [0,1] written in base n. The number's representation will be $0.n_1n_2n_3...$, where $0 \leq n_1, n_2, n_3, ... \leq n-1$ and will signify that at moment i the active rules are the ones from R_{n_i+1}.

In the following we will study the generating power of this model.

We will consider several cases:

a) the temperature is constant (only one group of rules is continuously activated, e.g., 0.44444444...);

b) after finitely many steps the temperature becomes constant (there will be several steps when the temperature varies, but after some moment the temperature remains the same, e.g., 0.12543621542222222222222...);

c) the temperature is periodic (e.g., there is a smallest "period" through which the temperature varies, and then becomes as before, e.g., 0.12341234123412...);

d) after a while the temperature is periodic,
e.g., 0.437264884729876598765987765...;

e) the temperature has no period, but it is computable,
e.g., Π–3=0.14159265358979323846264338327950288419716939937510582097....

We will prove now that in the cases a) through e) one cannot go beyond Turing limit. It was proved that $TVH_1 = RE$ in [8], thus there is a TM that will simulate the work of each possible H system. One can construct for each of the cases a) through e) a Turing machine that generates each of those temperatures (each of these cases has the temperature codified as a computable number, so there is for each of them a Turing machine to generate it). One can easily construct now a Turing machine that has as input the codification of the H system and also the codification of the Turing machine generating the temperature and simulate the work of the two machines.

Let us introduce now the interesting case which will be studied in the Section 4:

f) the temperature has no period and it is an uncomputable number;
example: Chaitin's Ω constant (see, for example, [2]).

Since temperature is an unknown variable, τ varying in time it can be represented as a function on $\mathbf{N}$ with values in $\mathbf{N} \cap [0, n-1]$. Therefore, τ behaves like an oracle allowing rules $R_{\tau(t)}$ to be applied at moment t.

The family of languages generated by temperature time-varying H systems with at most n components, where n is greater than one, is denoted by $\tau - TTVH_n$. By $\tau - RE$ we denote the family of τ-recursively enumerable languages [5].

3 Universality Results

In [13] (Theorem 10.8) it is proved that $RE = TVH_n$, for all $n \geq 7$. From the proof, one can see that, in fact, we have $RE = TVH_7(2,3,2,3)$. This was improved in both the number of components and the diameter in [11]: $TVH_n(2,1,1,1) = TVH_n(1,2,1,1) = TVH_n(1,1,2,1) = TVH_n(1,1,1,2) = RE$, $n \geq 4$.. This result is the best one considering only the diameter of the system. Great effort went into diminishing the number of components in such a system and recently it was shown that such H systems with only two components are universal [9], and the current best results (in terms of the number of components) is that one component is enough for universality, [10], [8], [7]. The last two proofs mentioned above have constructions with a diameter of (4,3,4,4) and (4,5,4,4), respectively. We improve here these results by decreasing the diameter of an universal time-varying H system to (3,2,2,2), or (2,3,2,2), or (2,2,3,2), or (2,2,2,3). For our proof we use the universality of type-0 grammars (the previous two proofs were simulating Turing Machines and Tag systems); we think that the grammars are a closer model to the H systems than the Turing Machines and the Tag systems, this being one of the reasons why the diameter could be reduced significantly with respect to the previous constructions. As a secondary note, we would like to point out that most of the universality proofs in this area are simulating grammars, rather than directly the Turing Machines so, other proof techniques could be combined to the current proof if needed.

First, we give an auxiliary result, which will simplify the subsequent investigations.

Lemma 1. *$TVH_n(n_1,n_2,n_3,n_4) = TVH_n(n_3,n_4,n_1,n_2)$, for all $n \geq 1$ and all $n_i \geq 0, 1 \leq i \leq 4$.*

Proof. Consider a time-varying H system $\Gamma = (V,T,A,R_1,\ldots,R_n)$ and construct the system $\Gamma' = (V,T,A,R'_1,\ldots,R'_n)$ with

$$R'_i = \{u_3\#u_4\$u_1\#u_2 \mid u_1\#u_2\$u_3\#u_4 \in R_i\},\ 1 \leq i \leq n.$$

Because $(x,y) \vdash_r (w,z)$ by $r = u_1\#u_2\$u_3\#u_4$ if and only if $(y,x) \vdash_{r'} (z,w)$ by $r' = u_3\#u_4\$u_1\#u_2$, we obtain $L(\Gamma) = L(\Gamma')$. Clearly, if $dia(\Gamma) = (n_1,n_2,n_3,n_4)$, then $dia(\Gamma') = (n_3,n_4,n_1,n_2)$. □

We pass now the main result of this section:

Theorem 1. *$RE = TVH_1(3,2,2,2) = TVH_1(2,2,3,2)$.*
And also $RE = TVH_1(2,3,2,2) = TVH_1(2,2,2,3)$.

Proof. Consider a type-0 grammar $G = (N,T,S,P)$ in Kuroda normal form, that is, with the rules in P of the forms $B \to x, B \to DE, BC \to DE$, for $B,C,D,E \in N$, $x \in T \cup \{\lambda\}$.

Let P_1 be the set of context-free rules in P and P_2 be the set of non-context-free rules in P. We denote the rules in P_1 by $j : u_j \to v_j$, for $1 \leq j \leq m$, and the

rules in P_2 by $j : u_j \to v_j$, for $m+1 \le j \le l$. Note that $|u_j| = 1$ for $1 \le j \le m$, and $|u_j| = 2$ for $m+1 \le j \le l$.

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

$$
\begin{aligned}
V &= N \cup T \cup \{X, Y, F, Z\} \cup \{X_i, Y_i \mid 1 \le i \le l\},\\
A &= \{XSY, ZF\} \cup \{X_i x Y_i \mid 1 \le i \le m,\ i : B \to x \in P_1, x \in T\}\\
&\quad \cup \{X_i x Y_i \mid 1 \le i \le m,\ i : B \to DE \in P_1, D, E \in N\}\\
&\quad \cup \{X_i D E Y_i \mid m+1 \le i \le l,\ i : BC \to DE \in P_2, D, E \in N\},\\
R_1 &= \{\alpha_1 \# B \alpha_2 \$ X_i \# x Y_i,\ \alpha_1 x \# Y_i \$ X_i B \# \alpha_2 \mid i : B \to x \in P_1,\ \alpha_1 \in N \cup T \cup \{X\},\\
&\qquad \alpha_2 \in N \cup T \cup \{Y\},\ B \in N,\ x \in T \cup \{\lambda\}\}\\
&\quad \cup \{\alpha_1 \# B \alpha_2 \$ X_i \# DE,\ \alpha_1 DE \# Y_i \$ X_i B \# \alpha_2 \mid i : B \to DE \in P_1,\\
&\qquad \alpha_1 \in N \cup T \cup \{X\}, \alpha_2 \in N \cup T \cup \{Y\},\ B, D, E \in N\}\\
&\quad \cup \{\alpha_1 \# BC \$ X_i \# DE,\ \alpha_1 DE \# Y_i \$ BC \# \alpha_2 \mid i : BC \to DE \in P_2,\\
&\qquad \alpha_1 \in N \cup T \cup \{X\}, \alpha_2 \in N \cup T \cup \{Y\},\ B, C, D, E \in N\}\\
&\quad \cup \{Z \# F \$ \alpha \# Y,\ \#ZY \$ X \# \beta,\ XZY \# \$ \alpha \# F \mid \alpha, \beta \in T\} \cup \{Z \# F \$ Z \# F\}\\
&\quad \cup \{X_i \# \alpha_1 \$ X_i \# \alpha_1 \mid 1 \le i \le m, i : B \to \alpha_1 \alpha_2, \alpha_1 \in N \cup T, \alpha_2 \in N \cup \{\lambda\},\ B \in N\}\\
&\quad \cup \{X_i \# Y_i \$ X_i \# Y_i \mid 1 \le i \le m,\ i : B \to \lambda,\ B \in N\}\\
&\quad \cup \{X_i \# DE \$ X_i \# DE \mid m+1 \le i \le l,\ i : BC \to DE \in P_2,\ B, C, D, E \in N\},
\end{aligned}
$$

One can easily see that Γ has the diameter $(3, 2, 2, 2)$. We will prove in the following that the constructed time varying H system Γ has the same language as the grammar G; i.e., $L(\Gamma) = L(G)$.

We first prove that the time-varying H system is capable of generating all the words that are generated by the grammar G; i.e., $L(G) \subseteq L(\Gamma)$. The work of the system is done in two phases: the first phase is simulating the productions from the grammar and the second phase is actually producing the word in the language of $L(\Gamma)$ by removing the special markers from the current word.

We start with a "main" axiom, XSY, in fact we have only the start symbol from the grammar G between two special markers X, *and* Y which mark the start and the end of the word. We will replace S with other nonterminal and/or terminal symbols according to the productions in the grammar G. At some point we choose (nondeterministically) that the current word that appears between X and Y is terminal, which means that by removing from that word the special markers X, Y we generate a word in the language of the H system. This is done by the rules $Z\#F\$\alpha\#Y$, $\#ZY\$X\#\beta$, $XZY\#\$\alpha\#F$ which compose actually the phase two of our simulation, but let us focus on the first phase, the simulation of the grammar productions.

The rules from P_1 are simulated in the following way: let us assume that the current sentential form is XwY, where w is a word over $N \cup T$ and it contains the symbol $B \in N$ for which we have the rule in the grammar G: $k : B \to x$, $x \in T \cup \{\lambda\}$. We have the axiom $X_k x Y_k$ present initially in the system, and because of the rule $X_k \# x \$ X_k \# x$ present in the set of rules it is clear

that the aforementioned axiom "survives" through all the steps of the computation, so it is available for splicing with the main word using the following rule: $r_k : \alpha_1 \# B\alpha_2 \$ X_k \# x Y_k$. This will produce in one step $(Xw_1|Bw_2Y, X_k|xY_k) \vdash_{r_k} (Xw_1xY_k, X_kBw_2Y)$, where $w = w_1Bw_2$. At the next step in the computation we can apply to these two strings the rule $r'_k : \alpha_1 x \# Y_k \$ X_k B \# \alpha_2$, which will produce $(Xw_1x|Y_k, X_kB|w_2Y) \vdash_{r'_k} (Xw_1xw_2Y, X_kBY_k)$. At this moment we have correctly simulated the rule $k : B \to x$ from G producing the word Xw_1xw_2Y and the "by-product" X_kBY_k. The simulation of a rule $k : B \to DE$ follows the same path, the only difference is that in the splicing rules the right marker from the axiom (Y_k) is not appearing, in this way the diameter of the system could be kept to a low value.

We are now looking at the way the rules from P_2 $(k : BC \to DE)$ are simulated. Also this process is done in two steps as before, the first splicing is using the rule $r_k : \alpha_1 \# BC \$ X_k \# DE$ to splice together the main string and the axiom X_kDEY_k: $(Xw_1|BCw_2Y, X_k|DEY_k) \vdash_{r_k} (Xw_1DEY_k, X_kBCw_2Y)$. The second splicing is done according to the rule $r'_k : \alpha_1 DE \# Y_k \$ BC \# \alpha_2$ and produces the strings Xw_1DEw_2Y and X_kBCY_k. It is easy to see now that all the rules from the grammar G are simulated by the H system in this manner.

The last step of the simulation is to remove the special markers X and Y from the "main DNA strand" in the system. This is done by the rules: $Z\#F\$\alpha\#Y,\ \#ZY\$X\#\beta,\ XZY\#\$\alpha\#F$ which first replace Y by an F. After this, we produce the word ZY, which is able to remove X from the main word and, then, XZY (that is just produced by the last splicing) is able to remove also F from the word. In this way we generate a word in the language of the $L(\Gamma)$ if all the symbols in the main word are terminal at this moment.

We have shown so far the relation $L(G) \subseteq L(\Gamma)$, let us now prove the converse inclusion. We will show that the H system produces no other words than those generated by G. First we look at the rules used to keep the axioms in the system and let them "survive" through all the steps of the computation:

$$\begin{aligned}
&\{Z\#F\$Z\#F\} \\
\cup\ &\{X_i\#\alpha_1\$X_i\#\alpha_1 \mid 1 \le i \le m,\ i : B \to \alpha_1\alpha_2, \alpha_1 \in N \cup T, \alpha_2 \in N \cup \{\lambda\}, B \in N\} \\
\cup\ &\{X_i\#Y_i\$X_i\#Y_i \mid 1 \le i \le m,\ i : B \to \lambda,\ B \in N\} \\
\cup\ &\{X_i\#DE\$X_i\#DE \mid m+1 \le i \le l,\ i : BC \to DE \in P_2,\ B, C, D, E \in N\}.
\end{aligned}$$

It is easy to notice that from their specific form all these rules can only be applied to the axioms of the system. This is due to the fact that X_i is always the first letter of a word in the system, and in axioms it precedes the symbol(s) that will replace the nonterminal(s) (according to the rules in G). In all other instances words that X_i appears, it will be followed by the symbol(s) replaced by the rule i from the grammar. This is due the fact that the simulation of such a rewriting rule first cuts after X_k and before the symbol(s) to be rewritten. Following this discussion it is clear now that these rules mentioned above will not produce anything "bad". Another group of rules can be shown that is only leading to terminal configurations only if the rules are applied in the preestablished order: $Z\#F\$\alpha\#Y,\ \#ZY\$X\#\beta,\ XZY\#\$\alpha\#F$. If we replace the Y with a F in the

main word, then at the next step we have to use the rule $\#ZY\$X\#\beta$, otherwise the word ZY will not survive to the next configuration of the system, since no other splicing rule can be applied to it, and then X and F will never be removed (they need the words ZY and XZY respectively), which means that, in this case, we will not reach a terminal configuration. One can notice that the removal of X and Y can happen "early" in the simulation, and if we reach a terminal string, then that terminal string will also be reached in the grammar, on the other hand, this could lead to the blocking of the simulation, not leading to any "output", so in this case nothing new can be produced.

We will discuss now the case when the simulation of a rewriting rule from G (that should take two steps for all types of rules from G) is interrupted after the first step and the simulation of yet another rule continues after that. In this case we would have after the first step two words of the form: Xw_1Y_k and X_kw_2Y, k being the rule simulated. At this moment, two more rules could start to be simulated; k' in the first word and k'' in the second word, leading to four words now: $Xw_1'Y_{k'}$, $Y_{k'}w_1''Y_k$, $X_kw_2'Y_{k''}$ and $X_{k''}w_2''Y$. Now,we can continue "breaking-apart" the main word or just finish the simulation of the rules k', k''. If we chose to finish the simulation, then at the next step we would have two words that could finish the simulation of the original rule k and nothing new is produced. One might notice that if only one "half" is simulating a rule and the other part cannot simulate any rules, then that particular string cannot use any other splicing rule, thus disappearing from the system. This will lead to the fact that the simulation of the production cannot complete, thus no terminal configuration will be reached due to the special markers X_k, Y_k present in the string and which cannot be removed from now on. This concludes our justification since we showed that no splicing rule can lead to a terminal configuration that would produce a word not in $L(G)$.

The equality $RE = TVH_1(2, 2, 3, 2)$ follows directly from the Lemma 1.

The second equality mentioned in the theorem: $RE = TVH_1(2, 3, 2, 2)$ requires its own construction. We will give only the basic idea of the construction and leave the details to the reader:

We have to cut after the symbol(s) to be simulated: for a rule $i : BC \to DE$ we would have the splicing rules $BC\#\alpha_1\$DE\#Y_i$ and $\alpha_1\#BCY_i\$X_i\#DE$. The other rules are simulated in a similar way; the other modification to the previous construction is the removing of X, Y which should be performed in a reversed order: first replace X with T and then remove Y, finishing by removing T. The equality $RE = TVH_1(2, 2, 2, 3)$ also follows from Lemma 1. □

4 Computation Beyond the Turing Limit with H Systems

We now proceed to study the power of the new model introduced in this paper: the temperature controlled time-varying H systems. We will start by first noticing that these systems are universal: consider that all the components in a system contain the same set of rules, then the temperature makes no difference in the

computation of the system, so they are equivalent in power to the time-varying H systems. Moreover, since we showed in the previous section that the time-varying H systems are universal, it follows that the temperature controlled time-varying H systems are able to generate all RE languages. In the following we show that if the temperature is an uncountable number (τ), then the time-varying H systems controlled by the temperature τ are able to generate non-RE languages.

Theorem 2. *For τ an uncountable number, $\exists \Gamma \in \tau - TTVH_2$ such that $L(\Gamma) \notin RE$.*

Proof. The theorem states that a time-varying H systems with temperature τ is more powerful than Turing Machines if the temperature is an uncountable number. We can consider a time-varying H systems with temperature that has only two components and the temperature is measured with respect to a threshold (if the temperature is below the threshold, the set 1 is activated, if not, the set two is activated). If this number written in base 2 is not calculable, then we will show that there is a τ-H system to compute a language not in RE.

We construct the time-varying τ-H system with temperature $\Gamma = (V, T, A, R_1, R_2)$, with

$$V = \{X, Y, Z, B\} \cup T, \text{ where } T = \{0, 1\}$$
$$A = \{BX, Z0X, Z1X, YY\},$$
$$R_1 = \{B\#X\$Z\#0X,\ \alpha_1\alpha_2\#X\$Z\#0X \mid \alpha_1 \in \{0, 1, B\},\ \alpha_2 \in T\} \cup Q,$$
$$R_2 = \{B\#X\$Z\#1X,\ \alpha_1\alpha_2\#X\$Z\#1X \mid \alpha_1 \in \{0, 1, B\},\ \alpha_2 \in T\} \cup Q,$$

where

$$Q = \{Z0\#X\$Z0\#X,\ Z1\#X\$Z1\#X, Y\#Y\$Y\#Y\} \cup \{B\#\alpha_1\$\#YY \mid \alpha_1 \in T\}.$$

The work of this system is basically to copy the temperature into the generated word; if the temperature activates the set 1, then at that step a "0" is appended to the current string generated by the system if the set 2 was activated, then a "1" is appended. We can also choose nondeterministically to stop the work of the machine by deleting the nonterminal symbol B and, thus, produce a word in $L(\Gamma)$. It is clear now that the language generated is a sequence of words that approximate the temperature by 1, 2, 3... letters, but since the temperature was assumed uncomputable, then also this language is uncomputable, which means that this simple time varying H system generated a language that is not in RE with the help of the temperature. □

In the following we will give a sketch of the construction of a universal time-varying H system that also goes beyond Turing:

Theorem 3. $\tau - TTVH = \tau - RE$

Proof. We will show that we can simulate the work of the oracle Turing machines[1] with such a temperature controlled time-varying H system. The oracle

[1] for more information on oracle Turing Machines and their states $q_?$, q_Y, q_N, and/or $\tau - RE$ we refer the reader to any of the many good Theory of Computation handbooks; a good starting point is also [5].

in the Turing machine is assumed the same as the temperature τ. One can simulate the work of the Turing machine with the time-varying H system, see for example the construction from [8]. The only part of the Turing machine that needs to be considered is the actual oracle and the states that "deal" with the oracle: $q_?$, q_Y, q_N. To do this, one needs to "copy" the temperature into a word in the system (Theorem 2 did exactly this), then read (and erase) the first symbol in the word that "remembers" the temperature and move in the corresponding state q_Y or q_N in the simulated Turing Machine. The temperature could be recorded using the symbols q_Y and q_N and to start the word recording the temperature with an arbitrary number of $q_?$. A sequence of two splicing rules would splice the current configuration word in the Turing machine with the temperature string and replace the $q_?$ with the first symbol in the temperature string, making thus the choice of the oracle. A more delicate matter would require to have the temperature copied in two half strings; the first half being used to simulate the oracle and the second half to just copy the temperature into the string. When the first half is exhausted, one could create a new second half and transform the old second half into a new first half. Due to space restrictions, we will leave the remaining details of the construction to the reader. □

5 Final Remarks

In this paper we prove that real life systems, such as H systems can be more powerful than classical computers in terms of computational complexity, our model being able to simulate Turing machines with oracles. Moreover, the descriptional complexity of our model is the best among all types of time varying H systems.

As future research we would like to continue our investigation for universal H systems with even smaller diameter, and try to apply the idea of the diameter to the time-controlled H systems. It could also be interesting to look at these temperature time-varying H systems as acceptors: we are given an infinite "word" codified as temperature and such a machinery is accepting/rejecting the (temperature)word according to some final state conditions (a state i could be considered final if the number of different DNA strands present in the system is increased (or did not decrease) from the last time that particular set of rules i was applied, etc.). And then, after having such a definition for the final states, one could use different types of accepting methods for infinite words, such as Büchi, Muller, etc.

References

1. C.S. Calude, Gh. Păun, Bio-Steps Beyond Turing, CDMTCS Tech. Rep No 226, 2003, 1–28.
2. G. J. Chaitin, Information, Randomness and Incompleteness, Papers on Algorithmic Information Theory, World Scientific, Singapore, 1987 (2nd ed., 1990).
3. B.J. Copeland, Hypercomputation, *Minds and Machines*, 12, 4 (2002), 461–502.
4. J.-P. Delahaye, La barrière de Turing, *Pour la Science*, 312 October (2003), 90–95.

5. J.E. Hopcroft and J.D. Ullman,*Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.
6. T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology*, 49 (1987), 737 – 759.
7. M. Margenstern, Y. Rogozhin, Time-Varying Distributed H Systems of Degree 1 Generate All Recursively Enumerable Languages, *Proc. of Workshop on Membrane Computing* (WMC-CdeA 2001), Curtea-de-Argeş, România, 2001, 199–207
8. M. Margenstern, Y. Rogozhin, A Universal Time-Varying Distributed H System of Degree 1, *Proc. of 7th Intern. Meeting on DNA Based Computers* (DNA7) (N. Jonoska, N.C. Seeman, eds.), Tampa, Florida, USA, 2001 and *Lecture Notes in Computer Science* 2340, (N. Jonoska, N.C. Seeman, eds.), Berlin, (2002), 371–380
9. M. Margenstern, Y. Rogozhin, An universal time-varying distributed H system of degree 2, *Preliminary Proc. of Fourth Intern. Meeting on DNA Based Computers*, Pennsylvania Univ., June 1998, 83 – 84.
10. M. Margenstern, Y. Rogozhin, S. Verlan, Time-Varying Distributed H Systems with Parallel Computations: The Problem Is Solved, *Lecture Notes in Computer Science* 2943, (G. Goss, J. Hartmanis, and J. van Leeuwen eds.), Berlin, (2004), 48–53
11. A. Păun, On Time-varying H Systems, *Bulletin of the EATCS*, 67 (1999), 157–164
12. A. Păun, On controlled extended H systems of small radius, *Fundamenta Informaticae*, 31, 2 (1997), 185 – 193.
13. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.
14. G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, Springer-Verlag, Heidelberg, 1997.
15. C. Teuscher, M. Sipper. Hypercomputation: Hyper or computation?, *Communications ACM*, 45, 8 (2002), 23–24.