Karagiannis J. On the computational ability of the RNA polymerase II carboxy terminal domain. Communicative & Integrative Biology 2014; 7:e28303; http://dx.doi.org/10.4161/cib.28303

On the Computational Ability of the RNA Polymerase II Carboxy Terminal Domain

Jim Karagiannis[§]

Department of Biology, University of Western Ontario, London, Ontario, CANADA

[§]Corresponding Author

Tel: 519-661-2111 ext. 80975

Fax: 519-661-3935

Email: jkaragia@uwo.ca

Keywords:

Semi-Thue string rewriting system, RNA polymerase II, Carboxy terminal domain, Transcription, Turing completeness, Computation, Universal Turing machine

Abstract

The RNA polymerase II carboxy terminal domain has long been known to play an important role in the control of eukaryotic transcription. This role is mediated, at least in part, through complex post-translational modifications that take place on specific residues within the heptad repeats of the domain. In this addendum, a speculative, but formal mathematical conceptualization of this biological phenomenon (in the form of a semi-Thue string rewriting system) is presented. Since the semi-Thue formalism is known to be Turing complete, this raises the possibility that the CTD – in association with the regulatory pathways controlling its post-translational modification – functions as a biological incarnation of a universal computing machine.

The computational power of simple, combinatorial, symbolic systems

Past research has clearly demonstrated the ease with which simple, combinatorial, symbolic systems can function as universal computing machines (i.e. systems capable of calculating any computable function). In addition to Alan Turing's original construction[1], there are many other notable examples; one of the earliest of these being Marvin Minsky's 7 state, 4 symbol machine based upon an Emil Post type "tag system"[2]. Other examples include Stephen Wolfram's 2 state, 5 colour cellular automaton[3], as well as Wolfram's 2 state, 3 colour cellular automaton (the simplest system defined to date)[4]. In each case the simple manipulation of symbols according to explicit rules imparts upon the system the capacity to perform complex computations – in fact, given access to infinite memory and sufficient time, any computable function is calculable (i.e., any function that can be computed, is computable by the given system). Excellent reviews on this subject can be found in both the popular and scientific literature[3, 5-8].

Similar to the realm of abstract computing, the field of molecular and cellular biology is also filled with a multitude of combinatorial, symbolic systems; one of the most striking of which is comprised of the RNA polymerase II carboxy terminal domain and its associated effector molecules[9]. A speculative examination of the computational ability of this system forms the basis of this addendum and is discussed in detail below.

The RNA polymerase II carboxy terminal domain

The RNA polymerase II holoenzyme is a large, eukaryotic enzyme complex that functions to transcribe protein coding genes (as well as microRNAs)[10, 11]. Interestingly, the largest subunit of the complex, Rpb1, possesses at its carboxy terminus an unusual, repetitive consensus sequence referred to simply as the carboxy terminal domain (or CTD)[12-21]. The CTD

is comprised of multiple repeats of the heptapeptide sequence, YSPTSPS, and is highly conserved in all fungi, plants, and metazoans[20, 22]. In addition, it has long been known that the Rpb1 CTD exists in both hyper- and hypo-phosphorylated states and that regulated changes in phosphorylation (on Tyr-1, Ser-2, Thr-4, Ser-5, and Ser-7 residues) influence both the initiation of transcription and transcript elongation[18, 19, 23-30]. Current models suggest that these modifications also affect the physical recruitment of accessory proteins that function in various aspects of pre-mRNA processing[23-25, 27, 29-41].

The importance of the CTD is also supported by the fact that it is essential for viability in all organisms tested to date. While partial truncations of the CTD sequence can be tolerated, the deletion of the entire CTD is invariably lethal [15-17, 21, 25, 42]. Curiously, while the CTD is indeed essential for viability, it is not required for basal transcriptional activity *in vitro* [17, 26, 43]. This strongly suggests that, while the CTD is not catalytically essential, it must perform other crucial functions within eukaryotes. What these functions are, and the mechanism(s) by which the CTD carries out these functions, has been the subject of much interest [9, 12-16, 21, 25, 42, 44].

In the remainder of this addendum, a speculative, but testable mathematical hypothesis regarding the underlying nature of the CTD is proposed (a hypothesis based upon a careful consideration of some of the lab's previous results)[9, 42, 45, 46]. Within this paradigm, the CTD (and its associated effectors) are viewed as a simple semi-Thue string re-writing system[47, 48]. Since the semi-Thue computational formalism is known to be Turing complete, this raises the possibility that the CTD functions as a biological incarnation of a universal computing machine.

To advance these ideas, it is first necessary to present some simple, mathematical preliminaries regarding semi-Thue systems. These preliminaries are described below.

Semi-Thue String Rewriting Systems

Semi-Thue string rewriting systems define an abstract model of computation first described by the Norwegian mathematician, Axel Thue[49]. Essentially, such systems are comprised of a series of "rewrite" rules that control how the system converts symbols in a string into other symbols. Formally, a semi-Thue system can be defined as a 2-tupel

$$T = (A, R)$$

where A describes a finite alphabet. Given *A*, it is possible to define *A** (the Kleene closure)[50]. *A** is simply the set of finite length words over *A* (i.e. the set of finite words resulting from the concatenation of the symbols comprising the alphabet). For example, if *A* were defined as

$$A = \{a, b, c\}$$

then A* would be comprised of

$$A^* = \{e, a, b, c, aa, ab, ac, ba, bb, bc, ca \ldots\}$$

where *e* represents the empty set. Using *A**, the re-write rules, *R*, of the system can be defined as

$$R \subseteq A^* \times A^*$$

R simply defines a set of pairs of strings, where each string is an element of A*. For instance, if

$$A = \{a, b, c\} \text{ and } R = \{(a, b), (aa, bc)\}$$

then the semi-Thue system, *T*, would search an initial string for an instance of *a*, or *aa*, and replace these symbols so that $a \rightarrow b$, or $aa \rightarrow bc$. Each re-write step in the process is performed non-deterministically, i.e. if there is more than one possibility of applying rules from *R*, then

there is no preference as to *which* rule is applied, or *where* it is applied to in the string. Rules continue to be applied until no occurrences of rewritable strings remain.

Using this definition it is possible to create systems capable of computation. For example, one could create a system, *T*, capable of adding two quantities together by defining *A* and *R* as

$$A = \{*, +\} \text{ and } R = \{* + *, **\}$$

where *n* concatenated asterisks represents the natural number, *n*. If given the string "*+**+***+****" (representing 1 + 2 + 3 + 4), the system would then non-deterministically apply the rewrite rule until the string was reduced to "**********" (representing the number, 10).

A useful tool to both create and examine semi-Thue systems is a javascript interpreter (freely available at https://github.com/mvmn/Thue-in-java) for the esoteric programming language, "Thue" (http://esolangs.org/wiki/Thue). Programs in "Thue" consist of a series of rewrite rules followed by the initial string. The rewrite rules are of the form *lefthandside* ::= *righthandside*. The list of rewrite rules terminates with the symbol ::= which is immediately followed by the initial string. For example, the system defined above would be represented in the "Thue" programming language as

```
*+*::=**
::=
*+**+***+****
```

Despite its simplicity, the semi-Thue formalism is nevertheless known to be Turing complete[47, 48]. Thus, given infinite memory and sufficient time, any computable function is calculable using such systems. In other words, any function that can be computed, is computable using the semi-Thue formalism. Other more sophisticated examples of semi-Thue systems implemented in the

"Thue" language can be found at http://lvogel.free.fr/thue.htm.

Conceptualizing the RNA pol II CTD as a semi-Thue String Rewriting System

To conceptualize the CTD as a biologically relevant and naturally selectable semi-Thue string rewriting system, several key observations must be noted. First, that each copy of the YSPTSPS heptad is phosphorylatable on Tyr-1, Ser-2, Thr-4, Ser-5, or Ser-7. Second, that eukaryotic cells modulate the phosphorylation status of each heptad through the *regulated* action of both kinases and phosphatases[12-15, 17-19, 21-25, 31, 32, 34, 40]. Third, that mutations affecting post-translational modification of the CTD profoundly influence phenotype in a wide variety of distinct organisms[12-15, 18, 21, 27-30, 32, 33, 35-40, 42-44, 46, 51-54]. And fourth, that progress through the transcription cycle is controlled (at least in part) by a series of *sequential* phosphorylation events. For example, Kin28 mediated Ser-5 phosphorylation in budding yeast leads to the recruitment of the Bur1/2 Ser-2 kinase complex and the subsequent phosphorylation of Ser-2 residues[55]. Thus, modifications at one residue can influence the subsequent recruitment of interacting proteins that are themselves CTD effectors. Many other examples of such phenomenon can be found in the literature[12-15, 23-27, 29-36, 38, 39, 41, 43, 51, 55-57].

Taking all of the above observations together, it is clear that 1) the CTD possesses symbols, 2) that these symbols can be altered according to explicit rules, and 3) that these modifications influence phenotype. Thus, all the requirements of a simple combinatorial, symbolic system capable of computation (and that is sensitive to natural selection) are satisfied. Significantly, this conceptualization is realized without postulating novel biological mechanisms or introducing unconventional computational paradigms. Thus, in the final analysis, these observations lead directly to the hypothesis that the CTD computes – and does so in a manner analogous to that described for semi-Thue string rewriting systems.

Simulating a Turing Machine with a Semi-Thue String Rewriting System

In the following paragraphs the computational power of semi-Thue systems is formally demonstrated by generating an abstract semi-Thue string rewriting system that acts as a Turing machine. As first described by Huet and Langford[58], it is relatively simple to construct a semi-Thue string rewriting system capable of simulating any Turing machine. Briefly, take a Turing machine

$$M = (Q, \Gamma, \delta, q_o, F)$$

where

1. Q is a finite set of internal states, $Q = \{q_{0\ldots}q_p\}$;
2. Γ is the tape alphabet, $\Gamma = \{s_{0\ldots}s_n\}$;
3. δ is the transition function, $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$;
4. $q_o \in Q$ is the initial state; and
5. $F \subseteq Q$ is the set of final states.

The transition function of this machine can then be given as a list of 5-tupels (q_i, s_j, q_l, s_k, d), where

1. $q_i \in Q \cup F$ is the current state;
2. $s_j \in \Gamma$ is the current symbol;
3. $q_l \in Q \cup F$ is the next state;
4. $s_k \in \Gamma$ is the next symbol; and
5. $d \in \{L, R\}$ is the direction of movement of the tape head (left or right).

Next, rewrite rules can be created that correspond to the transition rules in δ. Movements of the tape head to the left correspond to rewrite rules of the form

$$(s_m\ q_i\ s_j, q_l\ s_m\ s_k)$$

where s_m indicates the symbol initially to the left of the tape head ($0 \leq m \leq n$). Movements of the head to the right, on the other hand, correspond to rewrite rules of the form

$$(q_i\ s_j\ s_m, s_k\ q_l\ s_m)$$

where s_m in this case indicates the symbol initially to the right of the head. In this way the position of q_i denotes the position of the head, and the symbol to the right of q_i denotes the current symbol to be read.

Since it is possible to 1) simulate any Turing machine using semi-Thue grammar, as well as 2) define a *universal* Turing machine capable of simulating any other Turing machine (given the code and input word for that machine)[1], it thus follows that the semi-Thue formalism is indeed Turing complete.

A Concrete Example

To further illustrate the ideas presented above, consider a simple Turing machine that takes any binary string and prepends 0 to that string. A description of the transition function for such a machine is shown in Table 1. It is possible to visualize the actions of such a machine by encoding the transition function into the Turing machine simulator of the software package, JFLAP (http://www.jflap.org/). As shown in the animation contained in Supplemental Video SV1, the machine (using a binary string as input) halts after prepending 0 to the string. The ".jff" file encoding this Turing machine is included as Supplemental File S1.

As described in the previous section, it is now trivial to simulate this Turing machine

using a semi-Thue grammar by implementing the following rewrite rules

1. $(s > s_m\ ,\ > r_o\ s_m)$
2. $(r_0\ 0\ s_m\ ,\ 0\ r_o\ s_m)$
3. $(r_0\ 1\ s_m\ ,\ 0\ r_1\ s_m)$
4. $(s_m\ r_0\ *,\ l\ s_m\ 0)$
5. $(r_1\ 0\ s_m\ ,\ 1\ r_o\ s_m)$
6. $(r_1\ 1\ s_m\ ,\ 1\ r_1\ s_m)$
7. $(s_m\ r_1\ *\ ,\ l\ s_m\ 1)$
8. $(s_m\ l\ 0\ ,\ l\ s_m\ 0)$
9. $(s_m\ l\ 1\ ,\ l\ s_m\ 1)$
10. $(l >\ s_m\ ,\ > h\ s_m)$.

It is again possible to visualize the actions of the machine; this time by encoding the rewrite rules into the software package, "Thue". Since "Thue" is unable to accept subscripted symbols, r_0 becomes r, and r_1 becomes t within the program. In addition l becomes p to avoid confusion with respect to visualizing the symbols 1 and l. Thus, in this incarnation of the machine, the tape head is denoted by a letter (s, r, t, p, or h) that represents q_i. The Turing machine can thus be simulated in "Thue" using the program

```
s>0::=>r0
s>1::=>r1
s>>::=>r>
s>*::=>r*
r00::=0r0
r01::=0r1
r0>::=0r>
r0*::=0r*
r10::=0t0
r11::=0t1
r1>::=0t>
r1*::=0t*
```

```
0r*::=p00
1r*::=p10
>r*::=p>0
*r*::=p*0
t00::=1r0
t01::=1r1
t0>::=1r>
t0*::=1r*
t10::=1t0
t11::=1t1
t1>::=1t>
t1*::=1t*
0t*::=p01
1t*::=p11
>t*::=p>1
*t*::=p*1
0p0::=p00
1p0::=p10
>p0::=p>0
*p0::=p*0
0p1::=p01
1p1::=p11
>p1::=p>1
*p1::=p*1
p>0::=>h0
p>1::=>h1
p>>::=>h>
p>*::=>h*
::=
s>*
```

where the binary string would be inputted between the ">" and "*" in the final line of the program. As shown in the animation contained in Supplemental Video SV2, the machine, upon inputting a binary string, halts after prepending 0. The "Thue" file encoding this Turing machine is included as Supplemental File S2.

Encoding Programs Using a CTD-like Symbolic Structure

From a biological perspective, the key question that now remains is whether the CTD possesses enough symbolic complexity to encode specific programs using a semi-Thue grammar. As previously described, it can be formally shown that the CTD possesses (at minimum) 260 bits of informational entropy that could be exploited to encode such programs[9]. In other words,

symbols comprised of distinct heptad configurations could be used to represent the rewrite rules. Transition from one heptad configuration to another could then be envisioned to result from the phosphorylation dependent recruitment of a specific CTD effector.

In such a scenario, CTD-like symbols could be represented in “Thue” by way of five character strings (representing single heptads) in which "O" denotes a non-phosphorylated residue, and "P" denotes a phosphorylated residue (e.g., a heptad unphosphorylated on Tyr-1, Thr-4, and Ser-5, but phosphorylated on Ser-2 and Ser-7, would be represented by “OPOOP”). Since each heptad can exist in any one of 32 distinct configurations, it becomes possible to encode sophisticated programs using this grammar. For example, one could encode the Turing machine described above using only nine distinct heptad configurations (where *s* in the original "Thue" program is represented by (OOOOO), *r* by (POOOO), *t* by (OPOOO), *p* by (OOOPO), *h* by (PPPPP), 0 by (PPOPP), 1 by (OOPOO), > by (OPPPP) and * by (PPPPO). This machine (encoded using a CTD-like symbolic structure) can now be simulated in "Thue" by the program

```
(OOOOO)(OPPPP)(PPOPP)::=(OPPPP)(POOOO)(PPOPP)
(OOOOO)(OPPPP)(OOPOO)::=(OPPPP)(POOOO)(OOPOO)
(OOOOO)(OPPPP)(OPPPP)::=(OPPPP)(POOOO)(OPPPP)
(OOOOO)(OPPPP)(PPPPO)::=(OPPPP)(POOOO)(PPPPO)
(POOOO)(PPOPP)(PPOPP)::=(PPOPP)(POOOO)(PPOPP)
(POOOO)(PPOPP)(OOPOO)::=(PPOPP)(POOOO)(OOPOO)
(POOOO)(PPOPP)(OPPPP)::=(PPOPP)(POOOO)(OPPPP)
(POOOO)(PPOPP)(PPPPO)::=(PPOPP)(POOOO)(PPPPO)
(POOOO)(OOPOO)(PPOPP)::=(PPOPP)(OPOOO)(PPOPP)
(POOOO)(OOPOO)(OOPOO)::=(PPOPP)(OPOOO)(OOPOO)
(POOOO)(OOPOO)(OPPPP)::=(PPOPP)(OPOOO)(OPPPP)
(POOOO)(OOPOO)(PPPPO)::=(PPOPP)(OPOOO)(PPPPO)
(PPOPP)(POOOO)(PPPPO)::=(OOOPO)(PPOPP)(PPOPP)
(OOPOO)(POOOO)(PPPPO)::=(OOOPO)(OOPOO)(PPOPP)
(OPPPP)(POOOO)(PPPPO)::=(OOOPO)(OPPPP)(PPOPP)
(PPPPO)(POOOO)(PPPPO)::=(OOOPO)(PPPPO)(PPOPP)
(OPOOO)(PPOPP)(PPOPP)::=(OOPOO)(POOOO)(PPOPP)
(OPOOO)(PPOPP)(OOPOO)::=(OOPOO)(POOOO)(OOPOO)
(OPOOO)(PPOPP)(OPPPP)::=(OOPOO)(POOOO)(OPPPP)
(OPOOO)(PPOPP)(PPPPO)::=(OOPOO)(POOOO)(PPPPO)
(OPOOO)(OOPOO)(PPOPP)::=(OOPOO)(OPOOO)(PPOPP)
(OPOOO)(OOPOO)(OOPOO)::=(OOPOO)(OPOOO)(OOPOO)
```

```
(OPOOO)(OOPOO)(OPPPP)::=(OOPOO)(OPOOO)(OPPPP)
(OPOOO)(OOPOO)(PPPPO)::=(OOPOO)(OPOOO)(PPPPO)
(PPOPP)(OPOOO)(PPPPO)::=(OOOPO)(PPOPP)(OOPOO)
(OOPOO)(OPOOO)(PPPPO)::=(OOOPO)(OOPOO)(OOPOO)
(OPPPP)(OPOOO)(PPPPO)::=(OOOPO)(OPPPP)(OOPOO)
(PPPPO)(OPOOO)(PPPPO)::=(OOOPO)(PPPPO)(OOPOO)
(PPOPP)(OOOPO)(PPOPP)::=(OOOPO)(PPOPP)(PPOPP)
(OOPOO)(OOOPO)(PPOPP)::=(OOOPO)(OOPOO)(PPOPP)
(OPPPP)(OOOPO)(PPOPP)::=(OOOPO)(OPPPP)(PPOPP)
(PPPPO)(OOOPO)(PPOPP)::=(OOOPO)(PPPPO)(PPOPP)
(PPOPP)(OOOPO)(OOPOO)::=(OOOPO)(PPOPP)(OOPOO)
(OOPOO)(OOOPO)(OOPOO)::=(OOOPO)(OOPOO)(OOPOO)
(OPPPP)(OOOPO)(OOPOO)::=(OOOPO)(OPPPP)(OOPOO)
(PPPPO)(OOOPO)(OOPOO)::=(OOOPO)(PPPPO)(OOPOO)
(OOOPO)(OPPPP)(PPOPP)::=(OPPPP)(PPPPP)(PPOPP)
(OOOPO)(OPPPP)(OOPOO)::=(OPPPP)(PPPPP)(OOPOO)
(OOOPO)(OPPPP)(OPPPP)::=(OPPPP)(PPPPP)(OPPPP)
(OOOPO)(OPPPP)(PPPPO)::=(OPPPP)(PPPPP)(PPPPO)
::=
(OOOOO)(OPPPP)(PPPPO)
```

where the binary string would be inputted between the "(OPPPP)" and "(PPPPO)" in the final line of the program. Furthermore, it is again possible to visualize the actions of the machine using "Thue" (Supplemental Video SV3). The "Thue" file encoding this Turing machine is included as Supplemental File S3.

Finally, if we were to continue this speculative line of reasoning, and imagined the existence of distinct protein complexes that specifically bound the given tri-heptads of the left-hand side and specifically converted them (through the action of associated kinases/phosphatases) to the tri-heptad configurations of the right-hand side, it would then be possible to envision a CTD-like system capable of behaving as a Turing machine. Of course, it is not being suggested that this is indeed the case *in vivo* (i.e. it is not being suggested that the CTD *literally* behaves as a Turing machine). Instead, these concepts are presented only to demonstrate how easily one could program sophisticated algorithms into the CTD using string rewriting systems and established biological mechanisms.

Final Thoughts

In conclusion, it is important to note that the re-write rules used to construct a given program necessarily determine the computation being performed. This is to say, any number of unique programs could be constructed using different rewrite rules. Moreover, when considering these principles in a biological context, it is crucial to be cognisant of the fact that the rewrite rules would themselves be governed by the biochemical activity of the CTD effectors (e.g. kinases, phosphatases, cis-trans isomerases) present within the cell. Thus, in the final analysis, the "program" encoded would ultimately be under the control of natural selection. Thus, depending on the selective pressures experienced, any number of computational machines could be implemented through the CTD as a function of the rewrite rules.

Lastly, the conspicuous location of the CTD as part of an enzyme complex required for the transcription of all protein coding genes in almost all developmentally complex eukaryotes must also be noted. This last fact raises the fundamental biological question of whether CTD based computations have been exploited over the course of evolutionary time to control the sophisticated temporal/spatial regulation of transcription in these organisms.

List of Abbreviations

CTD, Carboxy Terminal Domain

pol, Polymerase

Ser, Serine

Thr, Threonine

Tyr, Tyrosine

Competing Interests

The authors declare that they have no competing interests.

Acknowledgements

The authors' would like to thank members of the UWO Biology and Biochemistry Departments for helpful discussions and/or critical reading of the manuscript.

References

1. Turing AM. On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, Series 2 1936; 42:230-65.

2. Minsky ML. Computation: Finite and Infinite Machines. Englewood Cliffs, New Jersey: Prentice-Hall, 1966.

3. Wolfram S. A New Kind of Science. Champaign, IL: Wolfram Media Inc., 2002.

4. Smith A. Universality of Wolfram's 2, 3 Turing Machine. 2007:http://www.wolframscience.com/prizes/tm23/TMProof.pdf.

5. Turlough Neary DW. Four Small Universal Turing Machines. Fundam Inform 2009; 91:123-44.

6. Herken R. The Universal Turing Machine: A Half-Century Survey. New York, NY, USA: Springer-Verlag Wien, 1995.

7. Davis M. The Universal Computer: From Leibniz to Turing. Boca Raton, FL, USA: CRC Press, 2000.

8. Damien Woods TN. The complexity of small universal Turing machines: A survey. Theoretical Computer Science 2009; 410:443-50.

9. Karagiannis J. Decoding the informational properties of the RNA polymerase II Carboxy Terminal Domain. BMC Res Notes 2012; 5:241.

10. Lee TI, Young RA. Transcription of eukaryotic protein-coding genes. Annu Rev Genet 2000; 34:77-137.

11. Lelli KM, Slattery M, Mann RS. Disentangling the many layers of eukaryotic transcriptional regulation. Annu Rev Genet 2012; 46:43-68.

12. Buratowski S. The CTD code. Nat Struct Biol 2003; 10:679-80.

13. Egloff S, Dienstbier M, Murphy S. Updating the RNA polymerase CTD code: adding gene-specific layers. Trends Genet 2012; 28:333-41.

14. Egloff S, Murphy S. Cracking the RNA polymerase II CTD code. Trends Genet 2008; 24:280-8.

15. Eick D, Geyer M. The RNA Polymerase II Carboxy-Terminal Domain (CTD) Code. Chem Rev 2013; 113:8456-90.

16. Jasnovidova O, Stefl R. The CTD code of RNA polymerase II: a structural view. Wiley Interdiscip Rev RNA 2013; 4:1-16.

17. Meinhart A, Kamenski T, Hoeppner S, Baumli S, Cramer P. A structural perspective of CTD function. Genes Dev 2005; 19:1401-15.

18. Phatnani HP, Greenleaf AL. Phosphorylation and functions of the RNA polymerase II CTD. Genes Dev 2006; 20:2922-36.

19. Prelich G. RNA polymerase II carboxy-terminal domain kinases: emerging clues to their function. Eukaryot Cell 2002; 1:153-62.

20. Stiller JW, Hall BD. Evolution of the RNA polymerase II C-terminal domain. Proc Natl Acad Sci U S A 2002; 99:6091-6.

21. Zhang DW, Rodriguez-Molina JB, Tietjen JR, Nemec CM, Ansari AZ. Emerging Views on the CTD Code. Genet Res Int 2012; 2012:347214.

22. Guo Z, Stiller JW. Comparative genomics of cyclin-dependent kinases suggest co-evolution of the RNAP II C-terminal domain and CTD-directed CDKs. BMC Genomics 2004; 5:69.

23. Komarnitsky P, Cho EJ, Buratowski S. Different phosphorylated forms of RNA polymerase II and associated mRNA processing factors during transcription. Genes Dev 2000; 14:2452-60.

24. Heidemann M, Hintermair C, Voss K, Eick D. Dynamic phosphorylation patterns of RNA polymerase II CTD during transcription. Biochim Biophys Acta 2013; 1829:55-62.

25. Napolitano G, Lania L, Majello B. RNA polymerase II CTD modifications: How many tales from a single tail. J Cell Physiol 2013.

26. Howe KJ. RNA polymerase II conducts a symphony of pre-mRNA processing activities. Biochim Biophys Acta 2002; 1577:308-24.

27. Ahn SH, Kim M, Buratowski S. Phosphorylation of serine 2 within the RNA polymerase II C-terminal domain couples transcription and 3' end processing. Mol Cell 2004; 13:67-76.

28. Patturajan M, Schulte RJ, Sefton BM, Berezney R, Vincent M, Bensaude O, et al. Growth-related changes in phosphorylation of yeast RNA polymerase II. J Biol Chem 1998; 273:4689-94.

29. Meinel DM, Burkert-Kautzsch C, Kieser A, O'Duibhir E, Siebert M, Mayer A, et al. Recruitment of TREX to the Transcription Machinery by Its Direct Binding to the Phospho-CTD of RNA Polymerase II. PLoS Genet 2013; 9:e1003914.

30. Gu B, Eick D, Bensaude O. CTD serine-2 plays a critical role in splicing and termination factor recruitment to RNA polymerase II in vivo. Nucleic Acids Res 2013; 41:1591-603.

31. Bataille AR, Jeronimo C, Jacques PE, Laramee L, Fortin ME, Forest A, et al. A universal RNA polymerase II CTD cycle is orchestrated by complex interplays between kinase, phosphatase, and isomerase enzymes along genes. Mol Cell 2012; 45:158-70.

32. Cho EJ, Kobor MS, Kim M, Greenblatt J, Buratowski S. Opposing effects of Ctk1 kinase and Fcp1 phosphatase at Ser 2 of the RNA polymerase II C-terminal domain. Genes Dev 2001; 15:3319-29.

33. Czudnochowski N, Bosken CA, Geyer M. Serine-7 but not serine-5 phosphorylation primes RNA polymerase II CTD for P-TEFb recognition. Nat Commun 2012; 3:842.

34. Devaiah BN, Singer DS. Cross-talk among RNA polymerase II kinases modulates C-terminal domain phosphorylation. J Biol Chem 2012; 287:38755-66.

35. Domingues MN, de Campos BM, de Oliveira ML, de Mello UQ, Benedetti CE. TAL effectors target the C-terminal domain of RNA polymerase II (CTD) by inhibiting the prolyl-isomerase activity of a CTD-associated cyclophilin. PLoS One 2012; 7:e41553.

36. Hintermair C, Heidemann M, Koch F, Descostes N, Gut M, Gut I, et al. Threonine-4 of mammalian RNA polymerase II CTD is targeted by Polo-like kinase 3 and required for transcriptional elongation. EMBO J 2012; 31:2784-97.

37. Licatalosi DD, Geiger G, Minet M, Schroeder S, Cilli K, McNeil JB, et al. Functional interaction of yeast pre-mRNA 3' end processing factors with RNA polymerase II. Mol Cell 2002; 9:1101-11.

38. Mayer A, Heidemann M, Lidschreiber M, Schreieck A, Sun M, Hintermair C, et al. CTD tyrosine phosphorylation impairs termination factor recruitment to RNA polymerase II. Science 2012; 336:1723-5.

39. Schwartz JC, Ebmeier CC, Podell ER, Heimiller J, Taatjes DJ, Cech TR. FUS binds the CTD of RNA polymerase II and regulates its phosphorylation at Ser2. Genes Dev 2012; 26:2690-5.

40. Schwer B, Sanchez AM, Shuman S. Punctuation and syntax of the RNA polymerase II CTD code in fission yeast. Proc Natl Acad Sci U S A 2012; 109:18024-9.

41. Vidyalakshmi S, Ramamurthy V. Phosphoserines of the carboxy terminal domain of RNA polymerase II are involved in the interaction with transcription-associated proteins (TAPs). OMICS 2013; 17:130-5.

42. Hoffman K, Yoo H, Karagiannis J. Synthetically engineered rpb1 alleles altering RNA polymerase II carboxy terminal domain phosphorylation induce discrete morphogenetic defects in Schizosaccharomyces pombe. Commun Integr Biol 2013; 6:e23954.

43. Carlson M. Genetics of transcriptional regulation in yeast: connections to the RNA polymerase II CTD. Annu Rev Cell Dev Biol 1997; 13:1-23.

44. Hajheidari M, Koncz C, Eick D. Emerging roles for RNA polymerase II CTD in Arabidopsis. Trends Plant Sci 2013; 18:633-43.

45. Karagiannis J, Balasubramanian MK. A cyclin-dependent kinase that promotes cytokinesis through modulating phosphorylation of the carboxy terminal domain of the RNA Pol II Rpb1p sub-unit. PLoS One 2007; 2:e433.

46. Saberianfar R, Cunningham-Dunlop S, Karagiannis J. Global gene expression analysis of fission yeast mutants impaired in Ser-2 phosphorylation of the RNA pol II carboxy terminal domain. PLoS One 2011; 6:e24694.

47. Dershowitz N, Jouannaud JP. Handbook of Theoretical Computer Science Volume B: Formal Methods and Semantics. Amsterdam: Elsevier, 1990.

48. Book RV, Otto F. String-rewriting systems. New York, NY, USA: Springer-Verlag, 1993.

49. Thue A. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. Skrifter utg av Videnskapsselsk i Kristiania 1 Matem-naturv Klasse, 1914; 10.

50. Kleene SC. Mathematical Logic. New York, NY, USA: John Wiley & Sons, 1967.

51. Coudreuse D, van Bakel H, Dewez M, Soutourina J, Parnell T, Vandenhaute J, et al. A gene-specific requirement of RNA polymerase II CTD phosphorylation for sexual differentiation in S. pombe. Curr Biol 2010; 20:1053-64.

52. Sukegawa Y, Yamashita A, Yamamoto M. The fission yeast stress-responsive MAPK pathway promotes meiosis via the phosphorylation of Pol II CTD in response to environmental and feedback cues. PLoS Genet 2011; 7:e1002387.

53. Jeong SJ, Kim HJ, Yang YJ, Seol JH, Jung BY, Han JW, et al. Role of RNA polymerase II carboxy terminal domain phosphorylation in DNA damage response. J Microbiol 2005; 43:516-22.

54. Ostapenko D, Solomon MJ. Budding yeast CTDK-I is required for DNA damage-induced transcription. Eukaryot Cell 2003; 2:274-83.

55. Qiu H, Hu C, Hinnebusch AG. Phosphorylation of the Pol II CTD by KIN28 enhances BUR1/BUR2 recruitment and Ser2 CTD phosphorylation near promoters. Mol Cell 2009; 33:752-62.

56. Qiu H, Hu C, Gaur NA, Hinnebusch AG. Pol II CTD kinases Bur1 and Kin28 promote Spt5 CTR-independent recruitment of Paf1 complex. EMBO J 2012; 31:3494-505.

57. Dronamraju R, Strahl BD. A feed forward circuit comprising Spt6, Ctk1 and PAF regulates Pol II CTD phosphorylation and transcription elongation. Nucleic Acids Res 2014; 42:870-81.

58. Huet G, Lankford D. On the Uniform Halting Problem for Term Rewriting Systems. National Institute for Research in Computer Science and Control 1978; Report 283.

Tables

Table 1 – Transition function for a Turing machine that prepends 0 to any binary string.

Current State	Current Symbol	Next State	Next Symbol	Direction
$\boldsymbol{s}$	>	r_0	>	R
$\boldsymbol{r_0}$	0	r_0	0	R
$\boldsymbol{r_0}$	1	r_1	0	R
$\boldsymbol{r_0}$	*	l	0	L
$\boldsymbol{r_1}$	0	r_0	1	R
$\boldsymbol{r_1}$	1	r_1	1	R
$\boldsymbol{r_1}$	*	l	1	L
$\boldsymbol{l}$	0	l	0	L
$\boldsymbol{l}$	1	l	1	L
$\boldsymbol{l}$	>	h	>	-

The symbol *s* represents the starting state; r_0 represents the state in which the head moves to the right and prints 0; r_1 represents the state in which the head moves to the right and prints 1; *l* represents the state in which the head moves to the left and prints the last read symbol; *h* represents the halting state; > is the symbol denoting the start of the binary string; * is the symbol denoting the end of binary string; *R*, and *L* represent the direction of movement of the head (right or left, respectively).

Supplemental Files

Supplemental File S1 – ".jff" file associated with Supplemental Video File SV1.

Supplemental File S2 – "Thue" file associated with Supplemental Video SV2.

Supplemental File S3 – "Thue" file associated with Supplemental Video SV3.

Supplemental Video SV1 – Animation of a Turing machine (encoded within "JFLAP") that prepends 0 to any binary string.

Supplemental Video SV2 – Animation of a Turing machine (encoded within "Thue") that prepends 0 to any binary string.

Supplemental Video SV3 – Animation of a CTD-like Turing machine (encoded within "Thue") that prepends 0 to any binary string.