# An Undecidable Nested Recurrence Relation

Marcel Celaya and Frank Ruskey*

Department of Computer Science, University of Victoria, CANADA

**Abstract.** Roughly speaking, a recurrence relation is *nested* if it contains a subexpression of the form $\ldots A(\ldots A(\ldots)\ldots)$. Many nested recurrence relations occur in the literature, and determining their behavior seems to be quite difficult and highly dependent on their initial conditions. A nested recurrence relation $A(n)$ is said to be *undecidable* if the following problem is undecidable: given a finite set of initial conditions for $A(n)$, is the recurrence relation calculable? Here *calculable* means that for every $n \geq 0$, either $A(n)$ is an initial condition or the calculation of $A(n)$ involves only invocations of $A$ on arguments in $\{0, 1, \ldots, n-1\}$. We show that the recurrence relation

$$\begin{aligned} A(n) = & A(n-4-A(A(n-4))) + 4A(A(n-4)) \\ & + A(2A(n-4-A(n-2)) + A(n-2)). \end{aligned}$$

is undecidable by showing how it can be used, together with carefully chosen initial conditions, to simulate Post 2-tag systems, a known Turing complete problem.

**Keywords:** Nested recurrence relation, undecidable problem, Turing complete problem, tag system, Hofstadter $Q$-sequence.

## 1 Introduction

In the defining expression of a recurrence relation $R(n)$, one finds at least one application of $R$ to some function of $n$. The Fibonacci numbers, for example, satisfy the recurrence $F(n) = F(n-1) + F(n-2)$ for $n \geq 2$. A recurrence relation $R(n)$ is called *nested* when the defining expression of $R$ contains at least two applications of $R$, one of which is contained in the argument of the other.

Many sequences defined in terms of nested recurrences have been studied over the years. One famous example is Hofstadter's $Q$ sequence, which is defined by the recurrence

$$Q(n) = Q(n - Q(n-1)) + Q(n - Q(n-2)), \qquad (1)$$

with initial conditions $Q(1) = Q(2) = 1$. This sequence is very chaotic, and a plot of the sequence demonstrates seemingly unpredictable fluctuation about the

* Research supported in part by an NSERC Discovery Grant.

line $y = x/2$. It remains an open question whether $Q$ is defined on all positive integers, despite its introduction in [7] over 30 years ago. Indeed, if it happens that there exists some $m$ such that $m < Q(m-1)$ or $m < Q(m-2)$, then the calculation of $Q(m)$ would require an application of $Q$ to a negative integer outside its domain. While little is known about the $Q$ sequence, other initial conditions that give rise to much better behaved sequences that also satisfy the $Q$ recurrence have been discovered [6], [11].

Another sequence defined in terms of a nested recurrence is the Conway-Hofstadter sequence

$$C(n) = C(C(n-1)) + C(n - C(n-1)), \tag{2}$$

with initial conditions $C(1) = C(2) = 1$. Unlike the $Q$ sequence, this sequence is known to be well-defined for $n \geq 1$, and in fact Conway proved that $\lim_{n\to\infty} C(n)/n = 1/2$. Plotting the function $C(n) - n/2$ reveals a suprising, fractal-like structure. This sequence is analyzed in depth in [8].

Another sequence whose structure is mainly understood, but is extraordinarily complex, is the Hofstadter-Huber $V$ sequence, defined by

$$V(n) = V(n-V(n-1))+V(n-V(n-4)), \text{ with } V(1) = V(2) = V(3) = V(4) = 1. \tag{3}$$

It was first analyzed by Balamoham, Kuznetsov and Tanny [2] and recently Allouche and Shallit showed that it is 2-automatic [1].

Some of these nested recurrences are well-behaved enough to have closed forms. Hofstadter's $G$ sequence, for example, is defined by

$$G(n) = n - G(G(n-1)), \text{ with } G(0) = 0. \tag{4}$$

This sequence has closed form $G(n) = \lfloor (n+1)/\phi \rfloor$, where $\phi$ is the golden ratio [5]. A sequence due to Golomb [6], defined by $G(1) = 1$ and $G(n) = G(n - G(n-1)) + 1$ when $n > 1$, is the unique increasing sequence in which every $n \geq 1$ appears $n$ times, and has closed form $G(n) = \lfloor (1 + \sqrt{8n})/2 \rfloor$.

Despite their wide variation in behaviour, all of these recursions are defined in terms of only three simple operations: addition, subtraction, and recurrence application. Of these, the latter operation makes them reminiscent of certain discrete systems—particularly the cellular automaton. Consider, for instance, the $Q$ sequence defined above. It is computed at any point by looking at the two values immediately preceeding that point, and using them as "keys" for a pair of "lookups" on a list of previously computed values, the results of which are summed together as the next value. It is well-known that many cellular automata are Turing complete; an explanation of how to simulate any Turing machine in a suitably-defined one-dimensional cellular automaton is given in [12]. With respect to nested recurrences, therefore, two questions naturally arise. First, does there exist, in some sense, a computationally universal nested recurrence defined only in terms of the aforementioned three operations? Second, given a nested recurrence, is it capable of universal computation? In this paper we aim to clarify the first question and answer it in the positive.

## 2 Tag Systems

The *tag system,* introduced by Emil Post in [10], is a very simple model of computation. It has been used in many instances to prove that some mathematical object is Turing complete. This was done, for example, with the one-dimensional cellular automaton known as Rule 110; the proof uses a variant of the tag system [4].

Such a system consists of a finite alphabet of symbols $\Sigma$, a set of production rules $\Delta : \Sigma \to \Sigma^*$, and an initial word $W_0$ from $\Sigma^*$. Computation begins with the initial word $W_0$, and at each step of the computation the running word $w_1 \dots w_k$ is transformed by the operation

$$w_1 \dots w_k \vdash w_3 \dots w_k \Delta(w_1) .$$

In other words, at each step, the word is truncated by two symbols on the left but extended on the right according to the production rule of the first truncated symbol. In this paper, we adopt the convention that lowercase letters represent individual symbols while uppercase letters represent words.

If the computation at some point yields a word of length 1, truncation of the first two symbols cannot occur; it is at this point the system is said to *halt.* The *halting problem for tag systems* asks: given a tag system, does it halt? As is the case for Turing machines, the halting problem for tag systems is undecidable [9].

Although this definition of tag systems has *two* symbols deleted at each step, there's no reason why this number need be fixed at two. In general, an *$m$-tag system* is a tag system where at each step $m$ symbols are removed from the beginning of the running word. The number $m$ is called the *deletion number* of the tag system. It is known that $m = 2$ is the smallest number for which $m$-tag systems are universal [3]. Thus, only 2-tag systems are considered in the remainder of these notes.

*Example 1.* Two tag systems are depicted below, both of which share alphabet $\Sigma = \{a, b, c\}$ and production rules $\Delta$ given by

$$a \to abb \qquad b \to c \qquad c \to a.$$

The initial word of the left tag system is $abcb$, while the initial word of the right tag system is $abab$. The left tag system never halts and is in fact periodic, while the right tag system halts after 11 steps.

```
abcb
  cbabb
    abba
      baabb
        abbc
          bcabb
            abbc
              ...
```

```
abab
  ababb
    abbabb
      babbabb
        bbabbc
          abbcc
            bccabb
              cabbc
                bbca
                  cac
                    ca
                      a
```

## 3 A Modified Tag System

The goal of this paper is to show that the recurrence given in the abstract can simulate some universal model of computation. In particular, we wish to show that if we encode the specification of some abstract machine as initial conditions for our recurrence, then the resulting sequence produced by the recurrence will somehow encode every step of that machine's computation. The tag system model seems like a good candidate for this purpose, since the entire run of a tag system can be represented by a single, possibly infinite string we'll call the *computation string*. For example, the string corresponding to the tag system above and to the right is *ababbabbabbccabbcacaa*. A specially-constructed nested recurrence $A$ would need only generate such a string on $\mathbb{N} = \{0, 1, 2, \ldots\}$ to simulate a tag system; each symbol would be suitably encoded as an integer.

Ideally, the sequence defined by the nested recurrence can be calculated one integer at a time using previously computed values. It would therefore make sense to find some tag-system-like model of computation capable of generating these strings one symbol at a time. That way, the computation of the $n^{\text{th}}$ symbol of a string in this new model can correspond to the calculation of $A(n)$ (or, more likely, some argument linear in $n$). With this motivation in mind, we now introduce a modification of the tag system model.

A *reverse tag system* consists of a finite set of symbols $\Sigma$, a set of production rules $\delta : \Sigma^2 \to \Sigma$, a function $d : \Sigma \to \mathbb{N}$, and an initial word $W_0 \in \Sigma^*$. While an ordinary tag system modifies a word by removing a *fixed* number of symbols from the beginning and adding a *variable* number of symbols to the end, the situation is reversed in a reverse tag system.

A single computation step of a reverse tag system is described by the operation

$$w_1 \ldots w_k \vdash w_{d(y)+1} \ldots w_k y,$$

where $y = \delta(w_1, w_k)$. Given a word that starts with $w_1$ and ends with $w_k$, the production rule for the pair $(w_1, w_k)$ yields a symbol $y$ which is appended to the end of the word. Then, the first $d(y)$ symbols are removed from the beginning of the word. The number $d(s)$ we'll call the *deletion number* of the symbol $s$. If at some point the deletion number of $y$ exceeds $k$, then the reverse tag system *halts.*

*Example 2.* Let $\Sigma = \{a, b\}$, $d(a) = 0$, and $d(b) = 2$. Define $\delta$ by

$$(a, a) \to b \qquad (a, b) \to b \qquad (b, a) \to b \qquad (b, b) \to a.$$

It takes 12 steps before this reverse tag system with initial word $W_0 = baaab$ becomes periodic.

```
baaab
baaaba
  aabab
    babb
    babba
      bbab
      bbaba
        abab
          abb
            bb
            bba
              ab
                b
                ba
                  b
                  ...
```

## 4 Simulating a Tag System with a Reverse Tag System

Consider a tag system $T = (\Sigma, \Delta, W_0)$ such that each production rule of $\Delta$ yields a nonempty string. The goal of this section is to construct a reverse tag system $R = (\Sigma', \delta, d, W_0')$ which simulates $T$.

This construction begins with $\Sigma'$. Some notation will be useful to represent the elements that are to appear in $\Sigma'$. Let $[s]_j$ denote the symbol "$s_j$", where $s$ is a symbol in $\Sigma$ and $j$ is an integer.

For each $s_i \in \Sigma$, write $\Delta(s_i)$ as $s_{i,\ell_i} \dots s_{i,2} s_{i,1}$. For each symbol $s_{i,j}$ in this word, the symbol $[s_{i,j}]_j$ shall appear in $\Sigma'$. For example, if $a \to abc$ is a production rule of $\Delta$, then $\Sigma'$ contains the three symbols $[a]_3$, $[b]_2$, and $[c]_1$. If $W_0 = q_1 q_2 \dots q_m$, the symbols $[q_1]_1, [q_2]_1, \dots, [q_m]_1$ are also included in $\Sigma'$. Constructed this way, $\Sigma'$ contains no more symbols than the sum of the lengths of the words in $\Delta(\Sigma)$ and the word $W_0$.

The production rules of $\delta$ include the rules

$$\delta([s_i]_*, [*]_1) = [s_{i,\ell_i}]_{\ell_i}$$
$$\delta([s_i]_*, [s_{i,j}]_j) = [s_{i,j-1}]_{j-1}$$

taken over all $s_i \in \Sigma$, all $j \in \{2, 3, \ldots, \ell_i\}$, and all possibilites for the $*$'s. Note that this specification of $\delta$ doesn't necessarily exhaust all possible pairs of $(\Sigma')^2$, however, any remaining pairs can be arbitrarily specified because they are never used during the computation of $R$.

Finally, the deletion numbers are specified by

$$d\left([s]_j\right) = \begin{cases} 0, & j > 1 \\ 2, & j = 1 \end{cases}$$

for all $[s]_j \in \Sigma'$, and if $W_0 = q_1 q_2 \ldots q_m$, then $W_0' = [q_1]_1 [q_2]_1 \cdots [q_m]_1$.

*Example 3.* This example demonstrates a simulation of the tag system $T$ in Example 1 using a reverse tag system $R = (\Sigma', \delta, d, W_0')$.

The production rules in Example 1 are

$$a \to abb \qquad b \to c \qquad c \to a.$$

To properly simulate $T$, the three symbols $a_3, b_2, b_1$ are needed for the first rule, the symbol $c_1$ is needed for the second, and the symbol $a_1$ is needed for the third. The initial word for $R$ is $W_0' = a_1 b_1 c_1 b_1$. Taking all these symbols together, we have $\Sigma' = \{a_1, b_1, c_1, b_2, a_3\}$.

If we take "$*$" to mean "any symbol or subscript, as appropriate," the production rules $\delta$ can be written as follows:

$$\begin{aligned}
&(a_*, *_1) \to a_3 \ (b_*, *_1) \to c_1 \ (c_*, *_1) \to a_1 \\
&(a_*, a_3) \to b_2 \\
&(a_*, b_2) \to b_1
\end{aligned}$$

Finally, every symbol with a subscript of 1 gets a deletion number of two, and zero otherwise:

$$d(a_1) = d(b_1) = d(c_1) = 2 \text{ and } d(b_2) = d(a_3) = 0.$$

The output of $R$ is depicted below. Compare the marked rows with the original output of $T$ in Example 1.

$\mathtt{a_1b_1c_1b_1} \leftarrow$
$\mathtt{a_1b_1c_1b_1a_3}$
$\mathtt{a_1b_1c_1b_1a_3b_2}$
$\quad\mathtt{c_1b_1a_3b_2b_1} \leftarrow$
$\qquad\mathtt{a_3b_2b_1a_1} \leftarrow$
$\qquad\mathtt{a_3b_2b_1a_1a_3}$
$\qquad\mathtt{a_3b_2b_1a_1a_3b_2}$
$\qquad\quad\mathtt{b_1a_1a_3b_2b_1} \leftarrow$
$\qquad\qquad\mathtt{a_3b_2b_1c_1} \leftarrow$
$\qquad\qquad\mathtt{a_3b_2b_1c_1a_3}$
$\qquad\qquad\mathtt{a_3b_2b_1c_1a_3b_2}$
$\qquad\qquad\quad\mathtt{b_1c_1a_3b_2b_1} \leftarrow$
$\qquad\qquad\qquad\mathtt{a_3b_2b_1c_1} \leftarrow$
$\qquad\qquad\qquad\ldots$

One point worth mentioning is that if a reverse tag system halts while simulating an ordinary tag system, then the simulated tag system must halt also. However, the converse is not true! A reverse tag system might keep rolling once it's completed the simulation of a halting tag system. The reverse tag system in Example 2 is a good example of this; it can survive even when there's only one symbol, while ordinary tag systems always require at least two.

**Theorem 1.** *Let $T = (\Sigma, \Delta, W_0)$ be a tag system such that each production rule of $\Delta$ yields a nonempty string, and let $R$ be a reverse tag system constructed as above in terms of $T$. Suppose $k > 0$, $w_1 \dots w_k \in \Sigma^*$, and $\Delta(w_1) = z_\ell z_{\ell-1} \dots z_1$. If $i_1, i_2, \dots, i_{k-1}$ are such that $[w_j]_{i_j} \in \Sigma'$, then*

$$[w_1]_{i_1} \dots [w_{k-1}]_{i_{k-1}} [w_k]_1 \vdash^* [w_3]_{i_3} \dots [w_k]_1 [z_\ell]_\ell \dots [z_1]_1$$

*in $R$.*

*Proof.* We have by construction of $R$ that

$$\begin{aligned}
[w_1]_{i_1} \dots [w_{k-1}]_{i_{k-1}} [w_k]_1 &\vdash [w_1]_{i_1} \dots [w_{k-1}]_{i_{k-1}} [w_k]_1 [z_\ell]_\ell \\
&\vdash [w_1]_{i_1} \dots [w_{k-1}]_{i_{k-1}} [w_k]_1 [z_\ell]_\ell [z_{\ell-1}]_{\ell-1} \\
&\vdots \\
&\vdash [w_1]_{i_1} \dots [w_{k-1}]_{i_{k-1}} [w_k]_1 [z_\ell]_\ell [z_{\ell-1}]_{\ell-1} \dots [z_2]_2 \\
&\vdash [w_3]_{i_3} \dots [w_{k-1}]_{i_{k-1}} [w_k]_1 [z_\ell]_\ell [z_{\ell-1}]_{\ell-1} \dots [z_2]_2 [z_1]_1 .
\end{aligned}$$

□

## 5 Simulating a Reverse Tag System with a Recurrence

While it's possible to describe how the recurrence $A$ simulates a reverse tag system, a better approach is to introduce another, simpler recurrence $B$ which does this simulation, then show how $A$ reduces to $B$. The simpler recurrence, without initial conditions, is:

$$B(n) = \begin{cases} B(n-2) + 2B(B(n-1)), & \text{if } n \text{ is even} \\ B(2B(n-2-B(n-1)) + B(n-2)), & \text{if } n \text{ is odd.} \end{cases}$$

Consider a reverse tag system $R = (\Sigma, \delta, d, W_0)$. The simulation of $R$ by $B$ necessitates encoding $\delta$ and $d$ as initial conditions of $B$. In order to do this, every symbol in $\Sigma$ and every possible pair in $\Sigma^2 = \Sigma \times \Sigma$ must be represented by a unique integer. Then, invoking $B$ on such an integer would correspond to evaluating $\delta$ or $d$, whatever the case may be. In order to avoid conflicts doing this, any integer representation of symbols and symbol pairs $\alpha : \Sigma \cup \Sigma^2 \to \mathbb{N}$ must be injective.

Assuming $\Sigma = \{s_1, s_2, \ldots, s_t\}$, one such injection is defined as follows:

$$\begin{aligned}\alpha(s_i) &= 4^{i+1} + 2 = 2^{2i+2} + 2, \text{ and} \\ \alpha(s_i, s_j) &= 2\alpha(s_i) + \alpha(s_j) = 2^{2i+3} + 2^{2j+2} + 6.\end{aligned}$$

The fact that $\alpha$ is injective can be seen by considering the binary representation of such numbers. Each of the bitstrings of $\alpha(s_1), \ldots, \alpha(s_t)$ are clearly distinct from one another, and the bitstring of $\alpha(s_i, s_j)$ for any $i, j \in \{1, 2, \ldots, t\}$ "interleaves" the bitstrings of $\alpha(s_i)$ and $\alpha(s_j)$. The constant 2 term in the definition of $\alpha$ is important in the next section, when the $A$ recurrence is considered.

The initial conditions of $B$ are constructed so that the encoding of $d$ occurs on $\alpha(\Sigma)$, and the encoding of $\delta$ occurs on $\alpha(\Sigma^2)$. For $i, j \in \{1, 2, \ldots, t\}$, The encoding for $d$ and $\delta$ is done respectively as follows:

$$\begin{aligned}B(\alpha(s_i)) &= 1 - d(s_i) \\ B(\alpha(s_i, s_j)) &= \alpha(\delta(s_i, s_j)).\end{aligned} \tag{5}$$

Is it worth noting that because of (5), $B(n)$ can take on negative values.

The largest value attained by $\alpha$ is

$$\alpha(s_t, s_t) = 3\alpha(s_t) = 3 \cdot 4^{t+1} + 6.$$

Let $c_0 = \alpha(s_t, s_t) + 2$. For the remainder of initial conditions that appear before $c_0$ and don't represent a symbol or symbol pair under $\alpha$, $B$ is assigned zero. One observes that even though the number of initial conditions specified is exponential in the size of $\Sigma$, only a polynomial number of these are actually nonzero.

The way the $B$ recurrence simulates $R$ is that $R$'s computation string, as represented under $\alpha$, is recorded on the odd integers, while the length of the running word is recorded on the even integers. Thus, for large enough $n$, the pair $(B(2n+1), B(2n+2))$ represents exactly one step of $R$'s computation. The simulation begins with the initial word $W_0 = q_1 q_2 \ldots q_m$. Specifically, the $m$ integers $\alpha(q_1), \ldots, \alpha(q_m)$ are placed on the first $m$ odd integers that come after $c_0$. The value $2m-2$ is then immediately placed after the last symbol of $W_0$; it is the last initial condition of $B$ and signifies the length of the initial word. Beyond this point, the recurrence of $B$ takes effect. An illustration of these initial conditions is given in the table below.

| $B(c_0+k)$ | $\alpha(q_1)$ | 0 | $\alpha(q_2)$ | 0 | $\ldots$ | $\alpha(q_{m-1})$ | 0 | $\alpha(q_m)$ | $2m-2$ |
|---|---|---|---|---|---|---|---|---|---|
| $k$ | 1 | 2 | 3 | 4 | $\ldots$ | $2m-3$ | $2m-2$ | $2m-1$ | $2m$ |

We now formalize what is meant by "$B$ simulates $R$." As mentioned previously, $B$ will alternatingly output symbols and word lengths. We encode the symbols and word lengths produced by $B$ in the following manner: any symbol

$s \in \Sigma$ is encoded as the integer $\alpha(s)$, while the length $k$ of some computed word is recorded in the output of $B$ as the value $2k-2$.

Suppose that at the $i^{\text{th}}$ computation step of $R$, the word $W = w_1 w_2 \dots w_k$ is produced. We will say that $B$ *computes the* $i^{th}$ *step of* $R$ *at* $n$ if the following equalities hold:

$$\begin{aligned}(B(n-2k+1), \dots, B(n-3), B(n-1)) &= (\alpha(w_1), \alpha(w_2), \dots, \alpha(w_k)) \\ B(n) &= 2k-2.\end{aligned}$$

This terminology is justified, since if $B$ computes the $i^{\text{th}}$ step of $R$ at $n$, then these equalities allow $W$ to be reconstructed from the output of $B$ near $n$. If there exist constants $r, s$ such that for all $i \in \mathbb{N}$, $B$ computes the $i^{th}$ step of $R$ at $ri+s$ whenever step $i$ exists, then we will say that $B$ *simulates* $R$.

**Theorem 2.** *With the above initial conditions,* $B$ *simulates* $R = (\Sigma, \delta, d, W_0)$.

*Proof.* If we suppose that the $0^{\text{th}}$ step of $R$ yields the initial word $W_0$, then by Table 1 it is clear that $B$ computes the $0^{\text{th}}$ step of $R$ at $c_0 + 2m$.

Assume that $B$ computes the $i^{\text{th}}$ step of $R$ at $2n$, where, again, we assume the word produced at step $i$ is $w_1 w_2 \dots w_k$. We would like to show that $B$ computes the $(i+1)^{\text{th}}$ step of $R$ at $2n+2$. Showing this, by induction, would prove the theorem.

If $y = \delta(w_1, w_k)$, then the word produced by $R$ at step $i+1$ is $w_{d(y)+1} \dots w_k y$. The last symbol of this word is $y$ and length of this word is $k+1-d(y)$. Therefore, to prove the theorem, we need only show that

$$\begin{aligned}B(2n+1) &= \alpha(y) \\ B(2n+2) &= 2(k+1-d(y)) - 2 \\ &= 2(k-d(y)).\end{aligned}$$

We first consider the point $2n+1$. Since this point is odd, we have

$$\begin{aligned}B(2n+1) &= B(2B(2n-1-B(2n)) + B(2n-1)) \\ &= B(2B(2n-1-2(k-1)) + B(2n-1-2(k-k))) \\ &= B(2\alpha(w_1) + \alpha(w_k)) \\ &= B(\alpha(w_1, w_k)) \\ &= \alpha(\delta(w_1, w_k)) \\ &= \alpha(y).\end{aligned}$$

The point $2n+2$ is even, thus

$$\begin{aligned}B(2n+2) &= B(2n) + 2B(B(2n+1)) \\ &= 2k-2+2B(\alpha(y)) \\ &= 2k-2+2(1-d(y)) \\ &= 2(k-d(y)).\end{aligned}$$

□

The above theorem describes the behaviour of $B$ when $R$ does not halt. If $R$ halts at any point, then there exists some even $n$ such that $B(n) = -2$. Then, $B(n+1) = B(2B(n+1) + B(n-1))$, and so $B$ is not calculable. Thus, $B$ with the prescribed initial conditions is calculable if and only if $R$ does not halt.

## 6 Reducing $A$ to $B$

It remains to show that the output of $B$ is effectively the same as the output of the recurrence

$$\begin{aligned} A(n) = & A(n-4-A(A(n-4))) + 4A(A(n-2)) \\ & + A(2A(n-4-A(n-2)) + A(n-4)), \end{aligned} \tag{6}$$

given the right initial conditions.

Once more, suppose we have a reverse tag system $R = (\Sigma, \delta, d, W_0)$. One restriction that will be made on $R$ is that $d(\Sigma) = \{0, 2\}$. Section 3 demonstrated how, despite this restriction, $R$ can still simulate an ordinary tag system. The goal at the beginning of these notes, to show that $A$ is Turing complete, is therefore still in reach.

Assume there are $t$ symbols in $\Sigma$, and $m$ symbols in the initial word $W_0$. Let $c_0 = \alpha(s_t, s_t) + 2$, as before. We now specify the initial conditions of $A$. For $n = 0, 1, \ldots, c_0$, $A$ and $B$ will share the same initial conditions. Immediately after, we'll have

$$A(c_0 + 4n + j) = \begin{cases} 0, & j = 0, 2 \\ B(c_0 + 2n + 1), & j = 1 \\ 2B(c_0 + 2n + 2), & j = 3 \end{cases} \tag{7}$$

for $0 \leq n < m$ and $0 \leq j < 4$.

The next theorem demonstrates how to obtain the sequence $B$ from $A$.

**Theorem 3.** *Using the given initial conditions for $A$ and $B$, $A$ is calculable if and only if $B$ is calculable. If $B$ is calculable, then equation (7) holds for all $n \geq 0$.*

*Proof.* We first consider the case when the argument of $A$ is even, that is, $j$ equals 0 or 2. By the initial conditions,

$$A(c_0 + 4m - 2) = A(c_0 + 4m - 4) = 0.$$

Suppose $A(y-2) = A(y-4) = 0$ for some $y \geq c_0 + 4m$ where $y \equiv 0 \bmod 2$. Then $A(y) = 0$ since all three terms of $A(y)$ are zero:

$$\begin{aligned} A(y-4-A(A(y-4))) &= A(y-4-A(0)) \\ &= A(y-4) \\ &= 0. \end{aligned}$$

$$\begin{aligned} 4A\left(A\left(y-2\right)\right) &= 4A\left(0\right) \\ &= 0. \end{aligned}$$

$$\begin{aligned} A\left(2A\left(y-4-A\left(y-2\right)\right)+A\left(y-4\right)\right) &= A\left(2A\left(y-4\right)+A\left(y-4\right)\right) \\ &= A\left(3A\left(y-4\right)\right) \\ &= A\left(0\right) \\ &= 0. \end{aligned}$$

Because every number in $\alpha\left(\Sigma\cup\Sigma^2\right)$ is congruent to 2 mod 4, we have $A\left(4n\right)=0$ for all $n\in\mathbb{N}$ whenever $A\left(4n\right)$ is defined.

Now, let $n\geq m$, and let $y=c_0+4n+1$. Suppose that

$$\begin{aligned} A\left(y-2\right) &= 2B\left(c_0+2n\right) \\ A\left(y-4\right) &= B\left(c_0+2n-1\right). \end{aligned}$$

If $k$ is the length of the word produced in the $(n-m)^{\text{th}}$ step of $R$'s computation, then $B\left(c_0+2n\right)=2k-2$. If $k=0$, then $R$ halts at this step. When this happens, $A\left(y\right)$ cannot be computed since

$$A\left(y\right)=4A\left(A\left(y-2\right)\right)+\cdots=4A\left(4k-4\right)+\cdots=4A\left(-4\right)+\cdots,$$

and $A$ is not defined on negative integers. Hence, if $R$ halts, then $A$ with the given initial conditions is not calculable.

For the remainder of the proof, then, assume $R$ never halts, that is, $k\geq 1$. The length of the word on $(n-m)^{\text{th}}$ step can be no longer than $m+(n-m)=n$, thus $k\leq n$.

If $s$ is the last symbol of the word at the $(n-m)^{\text{th}}$ step of $R$ (which exists since $k\geq 1$), then

$$B\left(c_0+2n-1\right)=\alpha\left(s\right)=4^i+2$$

for some $i\geq 2$.

We next show the first term of $A\left(y\right)$ is zero:

$$\begin{aligned} A\left(y-4-A\left(A\left(y-4\right)\right)\right) &= A\left(y-4-A\left(\alpha\left(s\right)\right)\right) \\ &= A\left(y-4-B\left(\alpha\left(s\right)\right)\right) \\ &= A\left(y-4-\left(1-d\left(s\right)\right)\right). \end{aligned}$$

Since $d\left(s\right)\in\{0,2\}$, this is equal to either $A\left(y-3\right)$ or $A\left(y-5\right)$. However, since $y$ is odd, both of these are equal to zero.

Since $A$ vanishes on multiples of 4, the second term of $A\left(y\right)$ also vanishes:

$$\begin{aligned} 4A\left(A\left(y-2\right)\right) &= 4A\left(2B\left(c_0+2n\right)\right) \\ &= 4A\left(4k-4\right) \\ &= 0. \end{aligned}$$

For the last term, we have

$$\begin{aligned}
&A\left(2A\left(y-4-A\left(y-2\right)\right)+A\left(y-4\right)\right)\\
&=A\left(2A\left(y-4-\left(4k-4\right)\right)+A\left(y-4\right)\right)\\
&=A\left(2A\left(c_0+4\left(n-k\right)+1\right)+A\left(y-4\right)\right)\\
&=A\left(2B\left(c_0+2\left(n-k\right)+1\right)+B\left(c_0+2n-1\right)\right)\\
&=A\left(2B\left(c_0+2n-1-\left(2k-2\right)\right)+B\left(c_0+2n-1\right)\right)\\
&=A\left(2B\left(c_0+2n-1-B\left(c_0+2n\right)\right)+B\left(c_0+2n-1\right)\right)\\
&=B\left(2B\left(c_0+2n-1-B\left(c_0+2n\right)\right)+B\left(c_0+2n-1\right)\right)\\
&=B\left(c_0+2n+1\right).
\end{aligned}$$

Therefore, $A\left(y\right)=A\left(c_0+4n+1\right)=B\left(c_0+2n+1\right)$.

It remains to consider the case $j=3$. As before, let $n\geq m$ and this time let $y=c_0+4n+3$. Assume

$$\begin{aligned}
A\left(y-2\right)&=B\left(c_0+2n+1\right)\\
A\left(y-4\right)&=2B\left(c_0+2n\right).
\end{aligned}$$

We wish to show that $A\left(y\right)=2B\left(c_0+2n+2\right)$. Again, $k$ is the length of the word after $n-m$ steps of $R$, so that $B\left(c_0+2n\right)=2k-2$. $k$ must be larger than zero, otherwise $A\left(y-2\right)$ would not be well-defined. The number $c_0+2n+1$ is odd, so $B\left(c_0+2n+1\right)=\alpha\left(s'\right)=4^i+2$ for some $s'\in\Sigma$, $i\geq 2$.

The first term of $A\left(y\right)$ is

$$\begin{aligned}
A\left(y-4-A\left(A\left(y-4\right)\right)\right)&=A\left(y-4-A\left(4k-4\right)\right)\\
&=A\left(y-4\right)\\
&=2B\left(c_0+2n\right).
\end{aligned}$$

The second term of $A\left(y\right)$ is

$$\begin{aligned}
4A\left(A\left(y-2\right)\right)&=4A\left(B\left(c_0+2n+1\right)\right)\\
&=4A\left(\alpha\left(s'\right)\right)\\
&=4B\left(\alpha\left(s'\right)\right)\\
&=4B\left(B\left(c_0+2n+1\right)\right).
\end{aligned}$$

Finally, it remains to show that the third term of $A\left(y\right)$ is zero. The quantity

$$\begin{aligned}
h&:=y-4-A\left(y-2\right)\\
&=y-4-\alpha\left(s'\right)\\
&=y-4-\left(4^i+2\right)\\
&=c_0+4\left(n-4^{i-1}-1\right)+1
\end{aligned}$$

is larger than zero, smaller than $y$, and is congruent to 1 mod 4. Thus, $A\left(h\right)=0$ if $h\leq c_0$, and $A\left(h\right)\in\alpha\left(\Sigma\right)$ if $h>c_0$. In either case, $A\left(h\right)$ is even and $0\leq A\left(h\right)\leq\max\alpha\left(\Sigma\right)$. Since $0<k\leq n$, we have

$$2A\left(h\right)+A\left(y-4\right)\geq 0$$

and

$$\begin{aligned} 2A(h) + A(y-4) &= 2A(h) + 4k - 4 \\ &\leq 2\max\alpha(\Sigma) + 4k - 4 \\ &< c_0 + 4n + 3 \\ &= y. \end{aligned}$$

Because $2A(h) + A(y-4) \equiv 0 \bmod 4$, $A(2A(h) + A(y-4)) = 0$. Therefore,

$$\begin{aligned} A(y) &= 2B(c_0 + 2n) + 4B(B(c_0 + 2n + 1)) \\ &= 2B(c_0 + 2n + 2). \end{aligned}$$

□

## 7 Concluding Remarks

In this paper, we have shown the existence of an undecidable nested recurrence relation. Furthermore, like its more well known cousins (1), (2), (3) and (4), our recurrence relation (6) is formed only from the operations of addition, subtraction, and recursion. Thus the result lends support to the idea that, in general, it will be difficult to prove broad results about nested recurrence relations. It will be interesting to try to determine whether other nested recurrence relations, such as (1), are decidable or not. If it is undecidable then it will certainly involve extending the techniques that are presented here, since the form of the recursion seems to prevent lookups in the manner we used.

## References


1. Allouche, J.P., Shallit, J.: A variant of Hofstadter's sequence and finite automata. arXiv:1103.1133v2 (2011)
2. Balamohan, B., Kuznetsov, A., Tanny, S.: On the behavior of a variant of Hofstadter's $Q$-sequence. J. Integer Sequences 10, 29 pages (2007)
3. Cocke, J., Minsky, M.: Universality of tag systems with $p = 2$. J. ACM 11(1), 15–20 (Jan 1964)
4. Cook, M.: Universality in elementary cellular automata. Complex Systems 15(1), 1–40 (2004)
5. Downey, P., Griswold, R.: On a family of nested recurrences. Fibonacci Quarterly 22(4), 310–317 (1984)
6. Golomb, S.: Discrete chaos: sequences satisfying "strange" recursions. preprint (undated, likely late eighties or early nineties) (1991)
7. Hofstadter, D.R.: Gödel, Escher, Bach: An Eternal Golden Braid. Basic Books (1979)
8. Kubo, T., Vakil, R.: On Conway's recursive sequence. Discrete Mathematics 152(1), 225–252 (1996)
9. Minsky, M.L.: Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. The Annals of Mathematics 74(3), 437–455 (Nov 1961)

10. Post, E.L.: Formal reductions of the general combinatorial decision problem. American Journal of Mathematics 65(2), 197–215 (Apr 1943)
11. Ruskey, F.: Fibonacci meets Hofstadter. Fibonacci Quarterly 49(3), 227–230 (2011)
12. Smith, A.R.: Simple computation-universal cellular spaces and self-reproduction. In: IEEE Conference Record of the 9th Annual Symposium on Switching and Automata Theory, 1968. pp. 269–277 (Oct 1968)