Undecidability of finite convergence for concatenation, insertion and bounded shuffle operators*

Charles E. Hughes

School of Computer Science, University of Central Florida, Orlando, FL 32816 USA

Abstract

The k-insertion and shuffle operations on formal languages have been extensively studied in the computer science and control systems literature. These operations can be viewed as purely abstract, as representations of biological processes or as models of the interleavings of concurrent processes. Questions naturally arise about closure and decidability. Many have been previously answered, especially as regards closure and non-closure of these operations on regular and context free languages. Here we will show the undecidability of a number of problems concerning the interaction of regular and context free languages under insertion and bounded shuffle, and the interaction of context free languages under self insertion and self bounded shuffle. Most of these proofs are consequences of the fact that the problem to decide if a Turing machine has an upper limit on execution time, independent of input, is undecidable.

Keywords: Concatenation; Shuffle and insertion operators; Bounded shuffle; Self insertion and self shuffle; Context free languages; Regular languages; Mortality; Constant execution time; Undecidability

1 Introduction

Define the *k-insertion* operation $\triangleright^{[k]}$ on pairs of languages over some alphabet Σ by

$$\mathbf{A} \triangleright^{[k]} \mathbf{B} = \{ x_1y_1x_2y_2 \ldots x_ky_kx_{k+1} \mid y_1y_2 \ldots y_k \in \mathbf{A}, x_1x_2 \ldots x_kx_{k+1} \in \mathbf{B}, x_i, y_j \in \Sigma^*\}$$

Read $\triangleright^{[k]}$ as "**A k**-insert into **B**". If **A = B**, we refer to the operation as *k-self insertion*. When **k = 1**, the superscript will be omitted; we will then refer to the operations simply as *insertion* and *self insertion*, respectively.

The insertion operation merely splits elements of **A** into **k** parts, and then breaks an element of **B** apart into **k+1** segments, so there are **k** places into which the parts of the **A** element can be inserted. This is done for all elements of **A** and **B** and for all possible ways to segment and then merge these elements according to the above rules. This means that we could choose to break the element of **B** so that we keep it intact and consider the **k** insert points to be at the start of the **B** word. Thus, the concatenation of **A** with **B**, $\mathbf{A} \bullet \mathbf{B}$, is contained in $\mathbf{A} \triangleright^{[k]} \mathbf{B}$, for all **k**.

Shuffle is an extension of insertion (or insertion is a simplification of shuffle). We define the *shuffle product* on pairs of languages over some alphabet Σ by $\mathbf{A} \diamond \mathbf{B} = \bigcup_{j \geq 1} \mathbf{A} \triangleright^{[j]} \mathbf{B}$. Since a **k+1** insert always includes all the strings in a **k** insert, one is tempted to define shuffle closure

---

* The research presented here is in participation with the Research in Augmented and Virtual Environments (RAVES) effort supported by the Naval Research Laboratory (NRL) VR LAB and the Office of Naval Research (ONR).

E-mail Address: ceh@cs.ucf.edu (C.E. Hughes)

as $\mathbf{A} \diamond \mathbf{B} = \mathbf{A} \triangleright^{[k]} \mathbf{B}$, where **k** is the smallest integer greater than **1** such that $\mathbf{A} \triangleright^{[k]} \mathbf{B} = \mathbf{A} \triangleright^{[k+1]} \mathbf{B}$. The problem is that such a finite **k** might not exist. In fact, in this paper we will show that it is undecidable to determine whether or not such a **k** exists, when **A** is either regular or context free, and **B** is context free. One may define a *bounded shuffle* operation $\mathbf{A} \diamond^{[k]} \mathbf{B} = \bigcup_{1 \leq j \leq k} \mathbf{A} \triangleright^{[j]} \mathbf{B} = \mathbf{A} \triangleright^{[k]} \mathbf{B}$, but it is easily seen that $\mathbf{A} \diamond^{[k]} \mathbf{B} = \mathbf{A} \triangleright^{[k]} \mathbf{B}$, which shows that this operation reduces to **k**-insertion.

The shuffle operation [1] and the more primitive insertion operator [22] have been studied extensively in the computer science literature due to their inherent mathematical interest and their relation to other problems, such as interleaved execution in concurrent systems. More recently, these operations have become of interest in molecular computing, with the proof that contextual insertions and deletions are sufficient to simulate Turing machines, showing the computational completeness of molecular systems based on these two simple operations alone [6], [19].

Issues of closure of classes of languages under the insertion and shuffle operations have been addressed in many papers including [3], [14], [15], [16], [19], [22], with the closure properties of the related deletion operation addressed in [21]. Decidability properties were considered in [19] for shuffle and in [7] for deletion. A comprehensive presentation of these topics and a more complete discussion of the notation used here may be found in [13].

2 Undecidability of convergence of simple self insertion

In [9], the authors presented a very simple proof that one cannot decide, for an arbitrary context free language **L**, whether or not $\mathbf{L} \bullet \mathbf{L} = \mathbf{L}$. The proof, repeated here, is the basis for our first undecidability result concerning insertion.

Theorem 1 (Hughes and Selkow [10])**:**
The problem to determine if $\mathbf{L} = \mathbf{\Sigma^*}$ is Turing reducible to the problem to decide if $\mathbf{L} \bullet \mathbf{L} \subseteq \mathbf{L}$, so long as **L** is selected from a class of languages $\mathfrak{C}$ over the alphabet Σ for which we can decide if $\mathbf{\Sigma} \cup \{\lambda\} \subseteq \mathbf{L}$.

Proof:
Let **L** be an arbitrary language in $\mathfrak{C}$. We claim that $\mathbf{L} = \mathbf{\Sigma^*}$ iff

(1) $\mathbf{\Sigma} \cup \{\lambda\} \subseteq \mathbf{L}$; and

(2) $\mathbf{L} \bullet \mathbf{L} = \mathbf{L}$.

Clearly, if $\mathbf{L} = \mathbf{\Sigma^*}$ then (1) and (2) trivially hold. Conversely, we have

$$\mathbf{\Sigma^*} \subseteq \mathbf{L^*} = \bigcup_{n \geq 0} \mathbf{L^n} \subseteq \mathbf{L},$$

which shows that $\mathbf{L} = \mathbf{\Sigma^*}$ (the first inclusion follows from (1), and the second one from (2)). □

The above property (decidability of $\mathbf{\Sigma} \cup \{\lambda\} \subseteq \mathbf{L}$) holds for the regular, context free and context sensitive classes of languages, but not for type 0, as that class's membership problem is undecidable. In general, this result holds so long as we can determine membership of strings of length at most one, that is, alphabetic characters and the empty string.

Corollary 1:
The problem "is $\mathbf{L} \bullet \mathbf{L} = \mathbf{L}$, for **L** context free or context sensitive?" is undecidable.

Proof:
This follows from the fact that the membership problem for context sensitive languages is decidable and the problem to decide if $\mathbf{L} = \Sigma^*$ is undecidable, for **L** a context free language. □

We will now show the corresponding theorem for any operation, $\otimes$, that subsumes self concatenation, that is where $\mathbf{L} \bullet \mathbf{L} \subseteq \mathbf{L} \otimes \mathbf{L}$. The simplest form of insertion, self insertion, is such an operation, since $\mathbf{L} \bullet \mathbf{L} \subseteq \mathbf{L} \triangleright \mathbf{L}$.

Theorem 2:
The problem to determine if $\mathbf{L} = \Sigma^*$ is Turing reducible to the problem to decide if $\mathbf{L} \otimes \mathbf{L} \subseteq \mathbf{L}$, so long as $\mathbf{L} \bullet \mathbf{L} \subseteq \mathbf{L} \otimes \mathbf{L}$ and **L** is selected from a class of languages $\mathbb{C}$ over Σ for which we can decide if $\Sigma \cup \{\lambda\} \subseteq \mathbf{L}$.

Proof:
Let **L** be an arbitrary language in $\mathbb{C}$. We claim that $\mathbf{L} = \Sigma^*$ iff
(1) $\Sigma \cup \{\lambda\} \subseteq \mathbf{L}$ and
(2) $\mathbf{L} \otimes \mathbf{L} \subseteq \mathbf{L}$.
Clearly, if $\mathbf{L} = \Sigma^*$ then (1) and (2) hold. The converse can be seen from

$$\Sigma^* \subseteq \mathbf{L}^* = \cup_{n \geq 0} \mathbf{L}^n \subseteq \mathbf{L}.$$

The first inclusion follows from (1), and the second from (1), (2) and the fact that $\mathbf{L} \bullet \mathbf{L} \subseteq \mathbf{L} \otimes \mathbf{L}$. □

Corollary 2:
The problem "is $\mathbf{L} \triangleright \mathbf{L} = \mathbf{L}$, for **L** context free or context sensitive?" is undecidable.

Proof:
This follows from the fact that the membership problem for context sensitive languages is decidable and the problem to decide if $\mathbf{L} = \Sigma^*$ is undecidable, for **L** a context free language. □

3 Mortality

The *Mortality Problem* for Turing machines with an infinite input tape is the problem to determine, for an arbitrary machine **M**, whether or not **M** eventually halts no matter in what configuration it is started. This is not the *Halting Problem*, since it means that we cannot just consider well-behaved machines that always start in their start states, positioned to the right of their arguments and which always end up to the right of the answer, which immediately follows these arguments (a convention called *Standard Turing Computation*). It also means that we might start with an infinite number of marked squares on the tape, unlike a normal Turing machine, which must start with its tape only finitely marked.

As is commonly done with Turing machines, we can, without loss of generality, limit the tape alphabet to **{0,1}**, where **0** denotes a blank, and **1** is the only mark (non-blank). Using that limitation on the tape alphabet, consider a function to compute **x+1** from **x**, using Standard

Turing Computation and unary representations of numbers. Such a machine could copy its one argument to the immediate right of the original scanned square and then move to the end of the copy appending a **1**. This machine always halts if it is started on a finitely marked tape, with the Standard Turing conventions obeyed. In fact, it can be written so it will always halt so long as the tape is finitely marked, even if the machine is started in other than the correct state and other than on the correct square. However, this machine is not mortal since, for example, it would run forever if started just to the right of an infinite sequence of 1's; the copy operation could never end.

The Mortality Problem was shown to be undecidable by Hooper [8]. This problem, although known to be decidable for some models of computation other than Turing machines, has been shown to be undecidable for two-counter machines [2]. The undecidability of this problem has turned out to be very useful in showing undecidability results for dynamical systems [2], [4], [5]. Although this paper does not address such results, we strongly believe that the results shown here may have significant applications to these problems.

4 Constant time execution

A Turing machine, **M**, is said to run in *constant time* if there is some finite positive integer **s** such that **M**, when started on any arbitrary finitely marked tape, executes at most **s** steps before halting. Formally, the set of all such machines can be described as **Constant_Time = { M | ∃ s ∀C [STP(C, M, s)] }**, where **STP** is the computable predicate that returns true if and only if **M**, when started in the configuration **C**, halts in no more than **s** steps. This **STP** function is the one used in standard proofs of the existence of a universal machine. It is actually primitive recursive, and hence always halts no matter what input it is given. When one looks at this description of the set **Constant_Time**, it may seem that the problem of membership is not recursively enumerable since there are two alternating unbounded quantifiers. This is, however, deceptive, since in time **s** we cannot navigate over more than **s** tape squares. Thus, ∀**C** can be replaced by the bounded version $\forall \mathbf{C}_{|\mathbf{C}| \leq \mathbf{s}}$, where **|C|** means the number of tape squares in **C**. This then reduces the predicate to a single unbounded existential quantifier, showing the problem to be recursively enumerable. Intuitively, we can check to see if the machine always halts in at most 1 step by writing down all configurations with one square (the scanned one) and an arbitrary state. If **M** halts in all cases, then it runs in constant time of 1. If this fails, then try all configuration of length at most 2. Continue this process, successively increasing the configuration length until a value is found for which **M** halts on all configurations of no greater length. This is guaranteed to halt if **M** ∈ **Constant_Time**. If, however, **M** is not in this set, then the procedure we just described runs forever. Thus, **Constant_Time** is semi-decidable (equivalently, recursively enumerable).

The undecidability of **Constant_Time** was shown in [10] in 1981, but this result has remained essentially unknown, perhaps because the paper's primary goal was showing a formal languages result, with **Constant_Time** being a vehicle to attain that result. In any case, we will outline the proof from that paper, since this is the key result we need to show the undecidability of a number of properties of the insertion and shuffle operations.

Consider an arbitrary Turing machine **M** operating on a finitely marked tape. If **M** ∈ **Constant_Time**, then **M** clearly is mortal since it would never care about infinite

markings, always stopping after some fixed number of steps, **s**, no matter what is on the tape. If, on the other hand, $\mathbf{M} \notin$ **Constant_Time**, then there are three cases we must consider.
(1) **M**, starting on some finite **C**, loops within some finite set of configurations; or
(2) **M**, starting on some finite **C**, runs forever without repeating any previously visited configuration; or
(3) **M** halts on all finite configurations, but there is no fixed maximum running time.
If either of cases (1) or (2) holds, then **M** is not mortal. For the other case, we need to start with a simple notation. Define $\mathcal{J}$ to be a set of configurations such that if $\mathbf{C} \in \mathcal{J}$ then **M** will scan all squares of ***C*** before it scans a square that is not part of **C. Let** $\{\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_m}\}$ be the states of **M**. We create a tree as follows. The root is an abstract node without a label. Its children are the **m** nodes labeled with each of the states of **M**, $\mathbf{q_i}$, one node per state. If $\mathbf{C_0}, \mathbf{C_1} \in \mathcal{J}$ and $\mathbf{q_j}$ is a symbol of $\mathbf{C_0}$ and $\mathbf{C_1}$, and $\mathbf{C_1} = \alpha\mathbf{C_0}$ or $\mathbf{C_1} = \mathbf{C_0}\alpha$, where α is a tape symbol (0 or 1), then $\mathbf{C_0}$ is the parent of $\mathbf{C_1}$ in the subtree whose root is labeled $\mathbf{q_j}$. Since **C** is not in **Constant_Time**, but every finite configuration causes it to halt, at least one of the subtrees must be infinite. Since the degree of each node is finite (all but the root have degree at most 2, and the root has degree **m**), König's Infinity Lemma states that at least one of the trees must have an infinite branch. Therefore, there must exist an infinite configuration that causes **M** to travel an infinite distance on the tape. It follows that **M** is immortal.

Theorem 3 (Hughes and Selkow [10])**:**
The set of mortal Turing machines is precisely the same as the set of Constant Running Time Turing machines.

Theorem 4 (Hooper [8]; Hughes and Selkow [10])**:**
The set of Constant Running Time Turing machines is recursively enumerable, non-recursive.

5 Valid traces and false traces of Turing computation

Turing machine computations are often represented by traces. A trace displays successive configurations, where configuration i+1 in the trace is the immediate successor of configuration i. Such traces typically require separators between configurations, and often have every other configuration displayed as the reverse of the actual configuration string. The reason for the reversals is that a single step computation (2 successive configurations) can be expressed by a context free language. Expressing successive Turing configurations without the reversal requires a context sensitive grammar, although other forms of computation such as single letter, single premise Post Canonical Systems do not require string reversal (a configuration is just a single number in unary notation).

One form of a *valid trace* of computation by a Turing machine **M** is a word
$\mathbf{C_1 \# C_2^R \$ C_3 \# C_4^R \ \ldots \$ C_{2k-1} \# C_{2k}^R \$}$, where $\mathbf{k \geq 1}$ and $\mathbf{C_i \Rightarrow_M C_{i+1}}$, for $\mathbf{1 \leq i < 2k}$. Here, $\Rightarrow_\mathbf{M}$ just means derive in **M**, and $\mathbf{C^R}$ means **C** with its characters reversed. An alternative notation, and the one primarily used in this paper is
$\mathbf{\#C_1\#C_3\#\ldots\#C_{2k-1}@!C_{2k}^R\$\%!\ldots C_4^R\ \$\%!C_2^R\$\%}$, where $\mathbf{k \geq 1}$ and $\mathbf{C_i \Rightarrow_M C_{i+1}}$, for $\mathbf{1 \leq i < 2k}$.The fact that such traces are even length is not a loss of generality, as has been

observed in numerous other papers. Our use of many separators, #, @, **!** and **$%**, may seem to be overkill, but their uses will be evident in later proofs.

A *false trace* is a word that has the above form, but for which there is some $\mathbf{i}$, $\mathbf{1 \le i < 2k}$, for which it is not the case that $\mathbf{C_i \Rightarrow_M C_{i+1}}$. Realize that a false trace just requires one mistake, whereas a valid trace requires all pairs to be correct. It can and has been proven many times that valid traces are context sensitive, non-context free languages, but false traces are context free. The idea is that one error can be checked for non-deterministically by a PDA, but any correct trace with three or more configurations requires a more sophisticated store than is provided by a PDA.

The set of valid traces of constant time Turing machines has the interesting property of having a fixed bound on the number of configurations in any trace. That fixed value is the constant time. It is this feature that we will use here to prove the undecidability of determining convergent properties for shuffle and insertion. In fact, this is what was also used to prove that the *Finite Power Property* for context free languages is undecidable.

Theorem 5 (Hughes and Selkow [9])**:**
The problem to determine, for an arbitrary context free language $\mathbf{L}$, if there exist a finite $\mathbf{n}$ such that $\mathbf{L^n = L^{n+1}}$ is undecidable, where $\mathbf{L^k}$ is a shorthand for $\mathbf{L \bullet L \bullet \ldots \bullet L}$, where concatenation is repeated $\mathbf{k}$ times.

Proof:
The purpose for presenting this already-published proof is to provide context for later proofs. Part of that context will be seeing how much easier concatenation is than insertion; this realization will help the reader to understand the manner chosen for subsequent constructions. The notation for traces used here is the first form presented above.

We will show that for each Turing machine $\mathbf{M}$ we can define a language $\mathbf{L}$ such that $\mathbf{M}$ is in **Constant_Time** iff there exists an $\mathbf{n}$ for which $\mathbf{L^n = L^{n+1}}$. Let $\mathbf{M}$ be a Turing machine. Define the languages:

1. $\mathbf{L_1 = \{ C_1 \# C_2^R \$ \mid C_1, C_2}$ are configurations $\}$,
2. $\mathbf{L_2 = \{ C_1 \# C_2^R \$ C_3 \# C_4^R \ldots \$ C_{2k-1} \# C_{2k}^R \$ \mid}$ where $\mathbf{k \ge 1}$ and, for some $\mathbf{i}$, $\mathbf{1 \le i < 2k}$, it is not true that $\mathbf{C_i \Rightarrow_M C_{i+1}}$, for $\mathbf{1 \le i < 2}$ $\}$,
3. $\mathbf{L = L_1 \cup L_2 \cup \{\lambda\}}$.

It is easy to see that $\mathbf{L}$ is context free. Moreover, any product of $\mathbf{L_1}$ and $\mathbf{L_2}$, which contains $\mathbf{L_2}$ at least once, is $\mathbf{L_2}$. For instance, $\mathbf{L_1 L_2 = L_2 L_1 = L_2 L_2 = L_2}$. This property shows that $\mathbf{(L_1 \cup L_2)^n = L_1^n \cup L_2}$. Thus, $\mathbf{L^n = \{\lambda\} \cup L_1 \cup L_1^2 \ldots \cup L_1^n \cup L_2}$. Analyzing $\mathbf{L_1}$ and $\mathbf{L_2}$ we see that $\mathbf{L_1^n \cap L_2 \ne \emptyset}$ just in case there is some word $\mathbf{C_1 \# C_2^R \$ C_3 \# C_4^R \ldots \$ C_{2n-1} \# C_{2n}^R \$}$ in $\mathbf{L_1^n}$ which is not also in $\mathbf{L_2}$. But this is so just in case there is some valid trace of length $\mathbf{2n}$. Clearly then, $\mathbf{L}$ has the finite power property if and only if $\mathbf{M}$ is in **Constant_Time**. ☐

6 Undecidability of convergence for limited shuffle and insertion of regular into context free

To simplify some of what follows, we will introduce a small amount of new notation. For languages **L1**, **L2**, define $\mathbf{L1\ (m)} \triangleright^{[n]} \mathbf{L2}$ to be shorthand for the successive **n**-insertion set

$$\mathbf{L1} \triangleright^{[n]} (\mathbf{L1} \triangleright^{[n]} (\mathbf{L1} \triangleright^{[n]} \ldots (\mathbf{L1} \triangleright^{[n]} \mathbf{L}))),$$

where this repetition occurs **m** times. More precisely:

- $\mathbf{L1\ (1)} \triangleright^{[n]} \mathbf{L2} = \mathbf{L1} \triangleright^{[n]} \mathbf{L2};$
- $\mathbf{L1\ (k+1)} \triangleright^{[n]} \mathbf{L2} = \mathbf{L1} \triangleright^{[n]} (\mathbf{L1\ (k)} \triangleright^{[n]} \mathbf{L2}).$

The self insertion $\mathbf{L\ (m)} \triangleright^{[n]} \mathbf{L}$ is simplified to $\mathbf{L\ (m)} \triangleright^{[n]}$.

We also use a little more shorthand, letting $\triangleright^{[1]}$ be abbreviated $\triangleright$ in all contexts. For example, $\mathbf{L1\ (m)} \triangleright^{[1]} \mathbf{L2}$ is abbreviated $\mathbf{L1\ (m)} \triangleright \mathbf{L2}$ and $\mathbf{L\ (m)} \triangleright^{[1]}$ is abbreviated $\mathbf{L\ (m)} \triangleright$.

Theorem 6:
The problem to decide whether or not $\exists \mathbf{m}\ \mathbf{R\ (m)} \triangleright \mathbf{L} = \mathbf{R\ (m+1)} \triangleright \mathbf{L}$ is undecidable for **L** a context free language and **R** a regular language.

Proof:
Let **M = (Q, {0, 1}, T)** be an arbitrary Turing Machine. Here **Q** is the finite set of states, **0** is the blank tape symbol, **1** is the only non-blank tape symbol and **T** is the set of 4-tuples (the rules in Post notation).

Consider the following four languages, two content free (**L1** and **L2**) and two regular (**L3** and **R**) over the alphabet **Σ = ({#, $, 1 , 0 } ∪ Q)**

$\mathbf{L1} = \{ \#C_1\#C_3\# \ldots\#C_{2k-1}@\%^k \mid k \geq 1$ and each $\mathbf{C_i}$, $1 \leq i \leq k$, is a configuration of **M** }

L1 is just strings that look like the starts of traces for **M** up to the **@** center split and followed by separators for the even configurations of the trace.

$\mathbf{L2} = \{ \#C_1\#C_3\# \ldots\#C_{2k-1}@X_{2k}\%X_{2k-2}\% \ldots X_2\%$, where $\mathbf{k} \geq 1$ and each $\mathbf{X_{2p}}$ is either empty or of the form $!C_{2p}^R\$$ and, for some $1 \leq i < 2k$, it's false that $C_i \Rightarrow_M C_{i+1}\}$

L2 is similar to a false trace for **M**, with a twist. We allow some of the even configurations to be missing; just including the **%**'s to stand in for these missing configurations. We still, however, require at least one even configuration to be there so we can have a derivation error.

L3 = { w | w has a single **@** and either (1) a **!** precedes the **@**; or (2) a **!** without matching **$%** follows the **@** and a series of balanced **%** (empty) and **! … $%** (configuration) segments **}**

$\mathbf{R} = \{ !C^R\$ \mid \mathbf{C}$ is a configuration of $\mathbf{M} \} \cup \{\lambda\}$

Context free grammars (**L1** and **L2**) and regular expressions (**L3**) are shown in Figure 1. We use **<A_BadPairInM>** for a set of rules that generates strings of the form $\mathbf{C1\ A\ C2^R}$ where it's false that $\mathbf{C1} \Rightarrow_M \mathbf{C2}$; and **<A_BadPairInMRev>** for a set of rules that generates $\mathbf{C1\#C3\ A\ X4^R\%!C2^R}$ where it's false that $\mathbf{C2} \Rightarrow_M \mathbf{C3}$; here **A** is a non-terminal, **C1** is a configuration, and **X4** is either empty or of the form $\mathbf{!C4^{R}\$}$, where **C4** is a configuration. These latter two languages are known to be context free.

```
L1:
   S1              →  # C S1 %
                   |  # C @ %
L2:
   S2              →  # <Step1Error> %
                   |  # <C> S2' ! <CReversed> $%
                   |  # <C> S2' %
   S2'             →  # <LaterError> %
                   |  # <C> S2' ! <CReversed> $%
                   |  # <C> S2' %
   GotError        →  # <C> A ! <CReversed> $%
                   |  # <C> A %
                   |  @
   <Step1Error>    →  < GotError_BadPairInM>
   <LaterError>    →  < GotError_BadPairInM>
                   |  < GotError_BadPairInMRev>
```

$\mathbf{C = ((1(0+1)^* + \lambda)\ Q\ (0+1)\ ((0+1)^*1 + \lambda)}$

$\mathbf{L3a = (0+1+\#+!+\$+Q)^*\ !\ (0+1+\#+!+\$+Q)^*\ @\ (0+1+\%+!+\$+Q)^*}$

$\mathbf{L3b = (\#C)^{+}\ @\ ((!C^{R}\$ + \lambda)\%)^*\ ((!C^{R}\$)^{+} + ((0+1+Q)^*\ !\ (0+1+Q)^*\ (!+\%)\ (0+1+\%+!+\$+Q)^*)))}$

$\mathbf{L3 = L2a + L3b}$

$\mathbf{R1 = !C^{R}\$ + \lambda}$

Figure 1: Grammars and regular expressions

Now, let $\mathbf{L = L1 \cup L2 \cup L3}$.

It's possible to see that $\mathbf{R1\ (1) \triangleright L = L \cup T1}$, where

$\mathbf{T1 = \{ \#C_1\#C_3\# \ldots \#C_{2k-1}@X_{2k}\%X_{2k-2}\% \ldots X_2\%}$, where $\mathbf{k \geq 1}$ and each $\mathbf{X_{2p}}$ is empty except one that is of the form $\mathbf{!C_{2p}{}^{R}\$}$ and either or both of the following hold, $\mathbf{C_{2p-1} \Rightarrow_M C_{2p}}$, $\mathbf{C_{2p} \Rightarrow_M C_{2p+1}}$, the second choice only being possible if $\mathbf{p < k\ \}}$

This can be explained by realizing the following:

R1 ▷ **L1** results in some string already in **L2** or **L3**, plus strings in **T1**, if any exist. Inserting an element of **R1** before the **@** produces a string already in **L3** (see **L3a**). Inserting it after the last **%** also produces an element in **L3** (see **L3b**). Placing it before a **%** gets us an element that is either already in **L2** (a trace error is produced) or an element in **T1**. Note that some exist in **T1** if there are valid traces of length at least one.

R1 ▷ **L2** results in strings already in **L2** or **L3**. The additions to **L2** occur when the element of **R1** is placed in an available slot (between the symbols **@%** or **%%**). This improves the appearance of the trace (filling in open slots), but does not change a false trace into a valid one (the error is still there). If the element of **R1** is placed before the @, after the last #, or in the middle of a slot that was already taken, the result is a member of **L3**.

R1 ▷ **L3** results in strings already in **L3**. Elements of **L3** cannot be repaired by insertions from **R1**. That is critical to the effectiveness of our construction, and is the reason we used so many separators. For instance, if we had omitted the **$**, then elements of this new **R1** could fix errors, e.g., by being inserted into an ill-formed slots that doesn't have a state symbol, fixing it and creating an element of **T1**. The problem is then that all valid traces could be created in just one step, destroying the connection between the number of insertion stages to reach convergence and the constant associated with execution time, should this machine **M** be a member of **Constant_Time**.

The second insertion must deal with the fact that elements of **T1** with one correct configuration to the right of the @ can be formed after the first insertion, but then that adds just one new type of string, members of **T1** with one correct configuration to the right of the @. Subsequent insertions have the similar property of adding one more correct configuration, if a valid trace of that length exists. Note that our inclusion of λ in **R1** lets us carry forward all the strings formed in early stages.

$\mathbf{R1\ (2) \rhd L = L \cup \{ \#C_1\#C_3\# \ldots\#C_{2k-1}@X_{2k}\%X_{2k-2}\% \ldots X_2\%}$, where $\mathbf{k \geq 2}$ and each $\mathbf{X_{2p}}$ is empty except two that are of the form $\mathbf{C_{2p}{}^R\$}$ and either or both of the following hold, $\mathbf{C_{2p-1} \Rightarrow_M C_{2p}}$, $\mathbf{C_{2p} \Rightarrow_M C_{2p+1}}$, the second choice only being possible if $\mathbf{p < k\ \}}$

…

$\mathbf{R1\ (j) \rhd L = L \cup \{ \#C_1\#C_3\# \ldots\#C_{2k-1}@X_{2k}\%X_{2k-2}\% \ldots X_2\%}$, where $\mathbf{k \geq j}$ and each $\mathbf{X_{2p}}$ is empty except **j** that are of the form $\mathbf{!C_{2p}{}^R\$}$ and either or both of the following hold, $\mathbf{C_{2p-1} \Rightarrow_M C_{2p}}$, $\mathbf{C_{2p} \Rightarrow_M C_{2p+1}}$, the second choice only being possible if $\mathbf{p < k\ \}\ \}}$

From this we see that if there is a fixed bound, **K**, on the number of steps for all computations in **M** and hence of the length of any valid trace, then $\mathbf{R1\ (K) \rhd L = R1\ (K+1) \rhd L}$. If no such bound exists then there is always a longer trace than any fixed value, and hence there is no **m**, such that $\mathbf{R1\ (m) \rhd L = R1\ (m+1) \rhd L}$. ☐

Theorem 7:

The problem to decide whether or not $\mathbf{\exists n\ R2 \diamond^{[\,n\,]} L = R2 \diamond^{[\,n+1\,]} L}$ is undecidable for **L** a context free language and **R2** a regular language.

Proof:

Based on an observation made earlier in this paper, we can recast this to the question $\mathbf{\exists n\ R2 \rhd^{[\,n\,]} L = R2 \rhd^{[\,n+1\,]} L}$. This is possible since $\mathbf{A \rhd^{[\,k\,]} B \subseteq A \rhd^{[\,k+1\,]} B}$. As we noted before, this substitution is only acceptable for bounded shuffle closure, not for the more standard unbounded shuffle closure operation. However, if there exists an **n**, as described in the theorem, then the unbounded shuffle closure, $\mathbf{R2 \diamond L}$, produces no more than $\mathbf{R2 \diamond^{[\,n\,]} L = R2 \rhd^{[\,n\,]} L}$.

We use the same notation and sets as in Theorem 6, except that

$\mathbf{R2 = (!C\$)^+}$, or in terms of trace components,

$\mathbf{R2 = \{!C_1\$!C_2\$! \ldots\$!C_k\$ \mid k \geq 1}$ and $\mathbf{C_i}$, $\mathbf{1 \leq i \leq k}$, is a configuration of $\mathbf{M\ \} \cup \{\lambda\}}$.

With this simple change, we get the results that correspond to those seen before. That is,

$\mathbf{R2 \rhd^{[\,1\,]}\text{\`{}}L = R1\ (1) \rhd L}$

$\mathbf{R2} \triangleright^{[2]} \mathbf{L} = \mathbf{R1}\ (2) \triangleright \mathbf{L}$

…

$\mathbf{R2} \triangleright^{[j]} \mathbf{L} = \mathbf{R1}\ (j) \triangleright \mathbf{L}$

The key here is that the value of **j** in $\mathbf{R2} \triangleright^{[j]} \mathbf{L}$ tells us how many segments into which we split an element of **R2**. These splits, when they occur at the right spots (just after each **$**) in a string of the form $(\mathbf{!C\$})^{\mathbf{j}}$, act like the **j**-th stage in the iterated insertion of Theorem 6. When the splits are made in the wrong places, or the word chosen from **R2** does not have precisely **j** configuration segments, we get garbage words that are already in **L3**.

From this we see that if there is a fixed bound, **K**, on the number of steps for all computations in **M** and hence of the length of any valid trace, then $\mathbf{R2} \triangleright^{[K]} \mathbf{L} = \mathbf{R2} \triangleright^{[K+1]} \mathbf{L}$. If no such bound exists then there is always a longer trace than any fixed value, and hence there is no **m**, such that $\mathbf{R2} \triangleright^{[m]} \mathbf{L} = \mathbf{R2} \triangleright^{[m+1]} \mathbf{L}$. □

7 Undecidability of convergence for limited self shuffle and self insertion of context free

Theorem 8:

The problem to decide whether or not $\exists \mathbf{m}\ \mathbf{L}\ (\mathbf{m}) \triangleright = \mathbf{L}\ (\mathbf{m+1}) \triangleright$ is undecidable for **L** a context free language.

Proof:

We use the same notation and sets as in Theorem 6, except that

$\mathbf{L} = \mathbf{L1} \cup \mathbf{L2} \cup \mathbf{L3} \cup \mathbf{L4} \cup \mathbf{L5} \cup \mathbf{R1}$, where **L4** and **L5** are the regular sets

$\mathbf{L4} = \{@,\#,!,\$,0,1\} \cup Q)^*\ @\ \{@,\#,!,\$,0,1\} \cup Q)^*\ @\ \{@,\#,!,\$,0,1\} \cup Q)^*$

= { **w** | **w** is over $(\{@,\#,!,\$,0,1\} \cup Q)$ and has at least two occurrences of the symbol @ } and

$\mathbf{L5} = !\ (\{!,\$,0,1\} \cup Q)^*\ !\ (\{!,\$,0,1\} \cup Q)^*$

= { **w** | **w** is over $(\{!,\$,0,1\} \cup Q)$, starts with **!**, and has two **!**'s }

The only new interactions here over Theorem 6 are

$\mathbf{L1} \triangleright \mathbf{L}$ results in **L1** plus strings already in **L4**

$\mathbf{L2} \triangleright \mathbf{L}$ results in **L2** plus strings already in **L4**

$\mathbf{L3} \triangleright \mathbf{L}$ results in **L3** plus strings already in **L4**

$\mathbf{L4} \triangleright \mathbf{L}$ results in strings already in **L4**

$\mathbf{L5} \triangleright \mathbf{L}$ results in **L5** plus strings already in **L3**

$\mathbf{R1} \triangleright \mathbf{R1}$ results in **R1** plus strings already in **L5**

From this we see that if there is a fixed bound, **K**, on the number of steps for all computations in **M** and hence of the length of any valid trace, then $\mathbf{L}\ (\mathbf{K}) \triangleright = \mathbf{L}\ (\mathbf{K+1}) \triangleright$. If no such bound exists then there is always a longer trace than any fixed value, and hence there is no **m**, such that $\mathbf{L}\ (\mathbf{m}) \triangleright = \mathbf{L}\ (\mathbf{m+1}) \triangleright$. □

Theorem 9:
The problem to decide whether or not $\exists$**n L** $\diamond^{[\,n\,]}$ **L** = **L** $\diamond^{[\,n+1\,]}$ **L** is undecidable for **L** a context free language.

Proof:
Again, based on an observation made earlier in this paper, we can recast this to the question $\exists$**n L** $\triangleright^{[\,n\,]}$ = **L** $\triangleright^{[\,n+1\,]}$. Details of our reasoning can be found in the start of Theorem 7.
We use the same notation and sets as in Theorem 7, except that
L = **L1** $\cup$ **L2** $\cup$ **L3** $\cup$ **L4** $\cup$ **L6** $\cup$ **R2**, where **L6** is the regular set
L6 = ! ({!,$,0,1} $\cup$ Q)* ({!,0,1} $\cup$ Q) ! {!,$,0,1} $\cup$ Q)* + !!({!,$,0,1} $\cup$ Q)*
= { **w** | **w** is over **({!,$,0,1}$\cup$Q)**, starts with **!**, and has a later **!** not immediately preceded by **$**}

The only new interactions here over Theorem 7 are
L1 $\triangleright^{[\,k\,]}$ **L** results in **L1** plus strings already in **L4**
L2 $\triangleright^{[\,k\,]}$ **L** results in **L2** plus strings already in **L4**
L3 $\triangleright^{[\,k\,]}$ **L** results in **L3** plus strings already in **L4**
L4 $\triangleright^{[\,k\,]}$ **L** results in strings already in **L4**
L6 $\triangleright^{[\,k\,]}$ **L** results in **L6** plus strings already in **L3**
R2 $\triangleright^{[\,k\,]}$ **R2** results in **R2** plus strings already in **L6**

From this we see that if there is a fixed bound, **K**, on the number of steps for all computations in **M** and hence of the length of any valid trace, then **L** $\triangleright^{[\,k\,]}$ = **L** $\blacktriangleright\triangleright^{[\,k+1\,]}$. If no such bound exists then there is always a longer trace than any fixed value, and hence there is no **m**, such that **L** $\triangleright^{[\,m\,]}$ = **L** $\blacktriangleright\triangleright^{[\,m+1\,]}$.

8 Conclusions and future avenues of investigation

We have shown the undecidability of a number of problems concerning the interaction of regular and context free languages under insertion and bounded shuffle, and the interaction of context free languages under self insertion and self bounded shuffle. All the problems addressed here are in some way connected to finite convergence, whether based on the number of iterations of a single insert or the degree of an insertion (the number of splits to a string in a single bounded shuffle).

While the application of these results is not addressed here, we believe that they are significant and important to the areas of concurrency, molecular computing, dynamical systems and evolutionary computing. Specifically, the inability to determine fixed finite convergence criteria is potentially important to areas where the interaction of a population with its own members, or the cross interactions with another population is one of the bases for evolution. A simple example of this can be seen by realizing that a generalization of the genetic algorithm crossover operation fits the characteristics of Theorem 2, where **A** $\otimes$ **B** = { **ux, wv** | **uv** $\in$ **A** and **wx** $\in$ **B** }, since **L** $\bullet$ **L** $\subseteq$ **L** $\otimes$ **L**

Finally, we believe that the proofs shown here, while not terribly complex, can be greatly simplified if the starting point were a model other than Turing machines. In particular, we believe that Factor Replacement Systems, a simple form of the single premise, one-letter Post canonical systems, are the right starting point [9], [12]. Unfortunately, the constant-time execution property for this class of systems remains an open problem.

9 Acknowledgements

The motivation for this paper came from a talk presented at the University of Central Florida by Masami Ito in November 2004. His lucid presentation inspired the author to return to his roots in computability theory. Ferucio Tiplea was kind enough to listen to the author's meanderings, offer encouragement as the proofs were first conceived, and provide significant simplifications in the final forms of several of the proofs and presentations.

References

[1] T. Araki and N. Tokura, Decision problems for regular expressions with shuffle and shuffle closure operators, *Transactions of Institute of Electronics and Communication Engineers of Japan* **J64-D** (1981) 1069-1073.

[2] V. D. Blondel, O. Bournez, P. Koiran, C. H. Papadimitriou, and J. N. Tsitsiklis, Deciding stability and mortality of piecewise affine dynamical systems, *Theoretical Computer Science* **255(1-2)** (2001) 687-696.

[3] C. Campeanu, K. Salomaa and S. Vagvolgyi, Shuffle quotient and decompositions, *Lecture Notes in Computer Science* **2295**, Springer (2002) 186-196.

[4] P. Collins and J. H. van Schuppen, Observability of piecewise-affine hybrid systems, *Lecture Notes in Computer Science* **2993**, Springer-Verlag (2004) 265-279.

[5] P. Collins and J. H. van Schuppen, Observability of hybrid systems and Turing machines, *Proceedings of the 43rd IEEE Conference on Decision and Control*, Bahamas, December, 2004.

[6] M. Daley, L. Kari, G. Gloor and R. Siromoney, Circular contextual insertions/deletions with applications to bomolecular computation, *String Processing and Information Retrieval Symposium / International Workshop on Groupware*, Cancun, Mexico (1999) 47-54.

[7] M. Domarattzki and A. Okhotin, Representing recursively enumerable languages by iterated deletion, *Theoretical Computer Science* **314(3)** (2004) 451-457.

[8] P. K. Hooper, The undecidability of the Turing machine immortality problem, *Journal of Symbolic Logic* **31(2)** (1966) 219-234.

[9] W. H. Hosken, Some Post canonical systems in one letter, *BIT* **12** (1972) 509-515.

[10] C. E. Hughes and S. M. Selkow, The finite power property for context-free languages, *Theoretical Computer Science* **15(1)** (1981) 111-114.

[11] C. E. Hughes, The equivalence of vector addition systems to a subclass of Post canonical forms, *Information Processing Letters* **7(4)** (1978) 201-204.

[12] C. E. Hughes, Single premise Post canonical forms defined over one-letter alphabets," *Journal of Symbolic Logic* **39** (1974) 489-495.

[13] M. Ito, *Algebraic Theory of Automata and Languages*, World Scientific Publishing Co. Pte. Ltd., Singapore, 2004.

[14] M. Ito, "Shuffle decomposition of regular languages," *Journal of Universal Computer Science* **8(2)** (2002) 257-259.

[15] M. Ito, B.Imreh and M. Katsura, "On shuffle closures of commutative regular languages," *Proceedings of DMTS'96, Combinatorics, Complexity and Logic*, edited by D.S. Bridges et al., Springer, Singapore (1996) 276 – 288.

[16] M. Ito, L. Kari and G. Thierrin, Shuffle and scattered deletion closure of languages, *Theoretical Computer Science* 245(1) (2000) 115-134.

[17] M. Ito, L. Kari and G. Thierrin, Insertion and deletion closure of languages, *Theoretical Computer Science* **183(1)** (1997) 3-19.

[18] M. Ito and R. Sugiura, *n*-Insertion on languages, *Lecture Notes in Computer Science* **2950**, Springer (2004) 213-218.

Kari_ L_ Deletion operations_ Closure properties_ International Journal of

[19] J. Jedrzejowicz, Undecidability results for shuffle languages, *Journal of Automata, Languages and Combinatorics* **1(2)** (1996) 147-159.
[20] L.Kari and G. Thierrin, Contextual insertions/deletions and computability, *Information and Computation* **131(1)** (1996) 47-61.
[21] L. Kari, Deletion operations: closure properties, *International Journal of Computer Mathematics* **52** (1994), 23-42.
[22] L. Kari, *On insertion and deletion in formal languages*, Ph.D. Thesis, University of Turku, Finland, 1991.