The Permutational Power of Finite Token Passing Networks

M. H. Albert[1] S. Linton[2] N. Ruškuc[3]

[1]Department of Computer Science
University of Otago

[2]School of Computer Science
University of St. Andrews

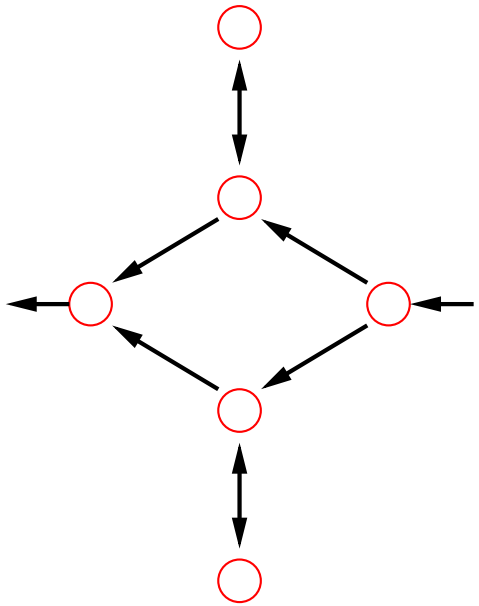
[3]School of Mathematics and Statistics
University of St. Andrews

PP 2005

University of Florida

Outline

- Background and Definitions
  - Token Passing Networks
  - Finite State Machines
  - Rank Encoding
- Previous Results
  - Atkinson, Livesy, Tulley
  - Albert, Atkinson, Ruškuc
- How Rich are TPN Languages?
  - Question
  - Results
- Discussion and Conclusions



Token Passing Networks

- A *token passing network*, $\mathcal{T}$, consists of a finite directed graph G together with a specified input vertex i and a specified output vertex o.
- A token passing network operates by processing a potentially infinite queue of distinct input tokens. There are three possible fundamental steps:

  Move A token on vertex v can be moved along the edge $v \longrightarrow w$ to w provided that w is unoccupied.

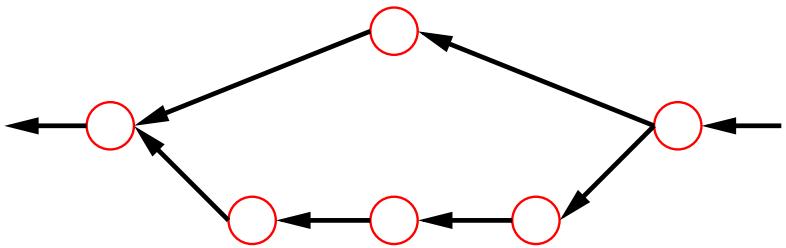
  Input The next token of the input queue can be placed at vertex i provided that i is unoccupied.

  Output A token at vertex o can be removed.

The Output Class of $\mathcal{T}$

- If $\mathcal{T}$ processes a finite series of tokens and stops when no tokens remain on the graph then the sequence of output tokens (in order) forms a permutation of the input.
- Since extra tokens can only get in the way, the set of such permutations is closed under involvement and we call it the *output class* $\mathrm{Out}(\mathcal{T})$.

For example:



has an output class whose basis is 321 and 51234.

Finite State Automata

- A *finite state automaton* (FSA), $\mathcal{A}$, consists of a directed graph whose edges are labelled with symbols from a set $\Sigma \cup \epsilon$. One vertex is designated the initial vertex, and a subset of the vertices are designated as accepting vertices.

Finite State Automata

- A *finite state automaton* (FSA), $\mathcal{A}$, consists of a directed graph whose edges are labelled with symbols from a set $\Sigma \cup \epsilon$. One vertex is designated the initial vertex, and a subset of the vertices are designated as accepting vertices.
- $\mathcal{A}$ *accepts a word* $w \in \Sigma^*$ if there is a walk from the initial vertex to an accepting vertex whose labels read in order (and omitting ϵ's) spell out w. The set of such words is called the *language* of $\mathcal{A}$.

Finite State Automata

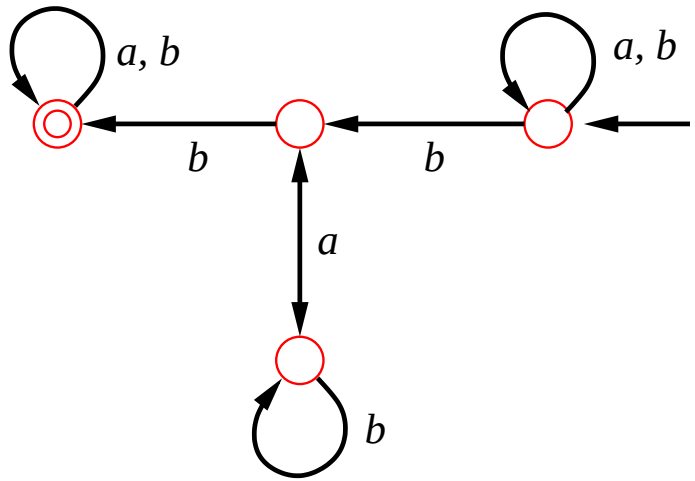
- A *finite state automaton* (FSA), $\mathcal{A}$, consists of a directed graph whose edges are labelled with symbols from a set $\Sigma \cup \epsilon$. One vertex is designated the initial vertex, and a subset of the vertices are designated as accepting vertices.
- $\mathcal{A}$ *accepts a word* $w \in \Sigma^*$ if there is a walk from the initial vertex to an accepting vertex whose labels read in order (and omitting ϵ's) spell out w. The set of such words is called the *language* of $\mathcal{A}$.
- A language is *regular* if it is accepted by some finite automaton. Equivalently (non-trivially) it can be formed from the empty and singleton languages using the operations of union, concatenation, and *.

Finite State Automata

- A *finite state automaton* (FSA), $\mathcal{A}$, consists of a directed graph whose edges are labelled with symbols from a set $\Sigma \cup \epsilon$. One vertex is designated the initial vertex, and a subset of the vertices are designated as accepting vertices.
- $\mathcal{A}$ *accepts a word* $w \in \Sigma^*$ if there is a walk from the initial vertex to an accepting vertex whose labels read in order (and omitting ϵ's) spell out w. The set of such words is called the *language* of $\mathcal{A}$.
- A language is *regular* if it is accepted by some finite automaton. Equivalently (non-trivially) it can be formed from the empty and singleton languages using the operations of union, concatenation, and *.
- The set of regular languages has good closure properties, and any such language has a rational generating function, easily computable from a deterministic FSA accepting it.

FSA example

The FSA:



accepts the language of words satisfying “between some pair of *b*’s there is an even number of *a*’s”. This can be written as:

$$(a \cup b)^* b (ab^* a)^* b (a \cup b)^*.$$

Finite State Transducers

- A *finite state transducer* (or simply transducer) is similar to an FSA.

Finite State Transducers

- A *finite state transducer* (or simply transducer) is similar to an FSA.
- However, the edges are labelled with *pairs* whose first coordinate comes from $\Sigma \cup \epsilon$ and whose second from $\Gamma \cup \epsilon$.

Finite State Transducers

- A *finite state transducer* (or simply transducer) is similar to an FSA.
- However, the edges are labelled with *pairs* whose first coordinate comes from $\Sigma \cup \epsilon$ and whose second from $\Gamma \cup \epsilon$.
- *Acceptance* defines a subset $\Sigma^* \times \Gamma^*$ which can be thought of as a non-deterministic map from (some subset of) the language accepted by the automaton obtained by projecting onto the first coordinate, and the language of that obtained from projecting onto the second.

Finite State Transducers

- A *finite state transducer* (or simply transducer) is similar to an FSA.
- However, the edges are labelled with *pairs* whose first coordinate comes from $\Sigma \cup \epsilon$ and whose second from $\Gamma \cup \epsilon$.
- *Acceptance* defines a subset $\Sigma^* \times \Gamma^*$ which can be thought of as a non-deterministic map from (some subset of) the language accepted by the automaton obtained by projecting onto the first coordinate, and the language of that obtained from projecting onto the second.
- More good preservation properties.

Rank Encoding

- The *rank encoding* of a permutation replaces each symbol in the permutation by its rank among the remaining symbols. So, the rank encoding of 341562 is 331221.

Rank Encoding

- The *rank encoding* of a permutation replaces each symbol in the permutation by its rank among the remaining symbols. So, the rank encoding of 341562 is 331221.
- A permutation is *k-bounded* if no symbol larger than *k* occurs in its rank encoding.

Rank Encoding

- The *rank encoding* of a permutation replaces each symbol in the permutation by its rank among the remaining symbols. So, the rank encoding of 341562 is 331221.
- A permutation is *k-bounded* if no symbol larger than k occurs in its rank encoding.
- For each k the set, $\mathcal{B}(k)$, of k-bounded permutations forms a closed class whose basis consists of the $k!$ permutations in S_{k+1} that begin with $k+1$.

Rank Encoding

- The *rank encoding* of a permutation replaces each symbol in the permutation by its rank among the remaining symbols. So, the rank encoding of 341562 is 331221.
- A permutation is *k-bounded* if no symbol larger than k occurs in its rank encoding.
- For each k the set, $\mathcal{B}(k)$, of k-bounded permutations forms a closed class whose basis consists of the $k!$ permutations in S_{k+1} that begin with $k+1$.
- **Key observation**: The rank encodings of $\mathcal{B}(k)$ form a regular language in $\{1, 2, \ldots, k\}^*$.

TPN's are FSA's

In *Permutations generated by token passing in graphs* (*TCS*, **178**, 1997) Atkinson, Livesey and Tulley showed:

Theorem

The rank encoding of $\mathrm{Out}(\mathcal{T})$ *is a regular language.*

This is most easily seen by considering the states of the underlying automaton as finite sequences of vertices of G representing the current locations of tokens (in rank order), and then noting that the basic operations can be interpreted naturally as transitions between such states.

Regularity is Robust

In *Regular closed sets of permutations* (*TCS*, **306**, 2003) Albert, Atkinson and Ruškuc showed:

Theorem

Let $\mathcal{C}$ be a closed class contained in $\mathcal{B}(k)$. If the rank encoding of $\mathcal{C}$ is a regular language, then so is the rank encoding of the basis of $\mathcal{C}$ relative to $\mathcal{B}(k)$. Moreover, if $\mathcal{A} \subseteq \mathcal{B}(k)$ is any subset whose rank encoding is a regular language then the rank encoding of $\mathrm{Av}(\mathcal{A}) \cap \mathcal{B}(k)$ is also regular.

Loosely, “a class is regular if and only if its basis is regular”. These results are, in principle, effective.

A Trivial Diversion

- Using the methods of AAR (implemented in GAP) we can confirm that the antichain reported by ALT in the basis of the output class of two stacks of size two in parallel is in fact the entire basis of the class modulo 5-boundedness.
- Consider a TPN which consists of a simple two way path of length 7 with input at position 3 and output at position 4.
  - The minimal automaton for the class has (only) 29 states.
  - The minimal automaton for its basis has 61 states.
  - The *basis* has exponential growth rate, whose asymptotic rate is a little more than 2.2399 (it is the root of an irreducible polynomial of degree 19).

The Main Questions

- What can we say about the regular classes that are produced by token passing networks?
- Can we find some necessary (and sufficient?) conditions which they satisfy?
- Can we determine whether a specific regular class is produced by a TPN?
- Are they common or rare?

Boundedness of a TPN

There are two types of boundedness to consider when dealing with TPN's:

Natural boundedness The maximum value of k such that a permutation of rank k can be produced.

Capacity restriction Consider TPN's required not to contain more than k tokens at any one time.

Finitely Many Languages of Each Boundedness

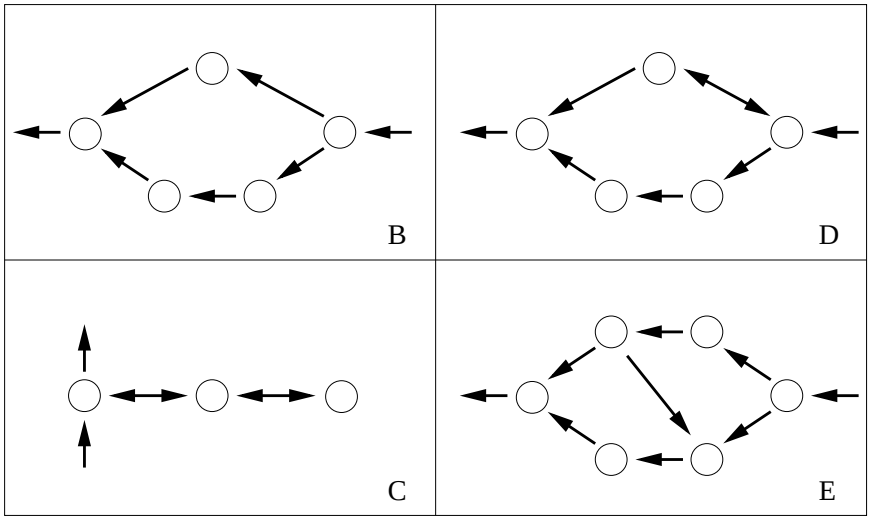
It doesn't matter (much):

Theorem

For both notions of boundedness, for each fixed k there are only finitely many different languages which occur as the rank encodings of the output classes of any token passing networks of that boundedness.

In both cases the proof relies on arguing that the boundedness restriction implies that were the underlying graph "too big" then some vertices would be superfluous.

Catalogue of 3-bounded Examples



- B Avoids 321
- C Avoids 312
- D Avoids both 31542 and 32541
- E Has an infinite basis whose language is 322321 and 3231(31)*321. E.g.:

3 2 5 1 7 4 9 6 11 10 8.

Open Questions

- Is there an efficient algorithm for determining the natural boundedness of $\mathcal{T}$?

Open Questions

- Is there an efficient algorithm for determining the natural boundedness of $\mathcal{T}$?
- If $\mathcal{T}$ has natural boundedness c is it ever necessary for it to contain more than c tokens?

Open Questions

- Is there an efficient algorithm for determining the natural boundedness of $\mathcal{T}$?
- If $\mathcal{T}$ has natural boundedness c is it ever necessary for it to contain more than c tokens?
- In producing an output permutation that lies in $\mathcal{B}(c)$, what is the maximum number of tokens that $\mathcal{T}$ might need to contain? (for $c = 4$ we have an example where 5 tokens must be in the network at some point).

Open Questions

- Is there an efficient algorithm for determining the natural boundedness of $\mathcal{T}$?
- If $\mathcal{T}$ has natural boundedness c is it ever necessary for it to contain more than c tokens?
- In producing an output permutation that lies in $\mathcal{B}(c)$, what is the maximum number of tokens that $\mathcal{T}$ might need to contain? (for $c = 4$ we have an example where 5 tokens must be in the network at some point).
- Can $\mathcal{T}$ be operated efficiently? That is, given $\mathcal{T}$ and a permutation is there a polynomial time algorithm to determine
  - Whether the permutation can be produced by $\mathcal{T}$, and
  - If so, how.

  (Exponential preprocessing in $|\mathcal{T}|$ not allowed!)

Open Questions

- Is there an efficient algorithm for determining the natural boundedness of $\mathcal{T}$?
- If $\mathcal{T}$ has natural boundedness c is it ever necessary for it to contain more than c tokens?
- In producing an output permutation that lies in $\mathcal{B}(c)$, what is the maximum number of tokens that $\mathcal{T}$ might need to contain? (for $c = 4$ we have an example where 5 tokens must be in the network at some point).
- Can $\mathcal{T}$ be operated efficiently? That is, given $\mathcal{T}$ and a permutation is there a polynomial time algorithm to determine
  - Whether the permutation can be produced by $\mathcal{T}$, and
  - If so, how.

  (Exponential preprocessing in $|\mathcal{T}|$ not allowed!)
- Given $\mathcal{T}$ determine the maximum c such that $\mathrm{Out}(\mathcal{T}) \supseteq \mathcal{B}(c)$.

Conclusions

- Our main result states that the languages generated by token passing networks are relatively restricted.
- Nevertheless, there are a wide variety of “interesting” behaviours and examples to explore.
- Certain algorithmic problems, particularly in the acyclic case *may* be of “applied” interest.