Synchronizing Heuristics for Non-Strongly Connected Automata

Sabancı Üniversitesi — FACULTY OF ENGINEERING AND NATURAL SCIENCES

PURE

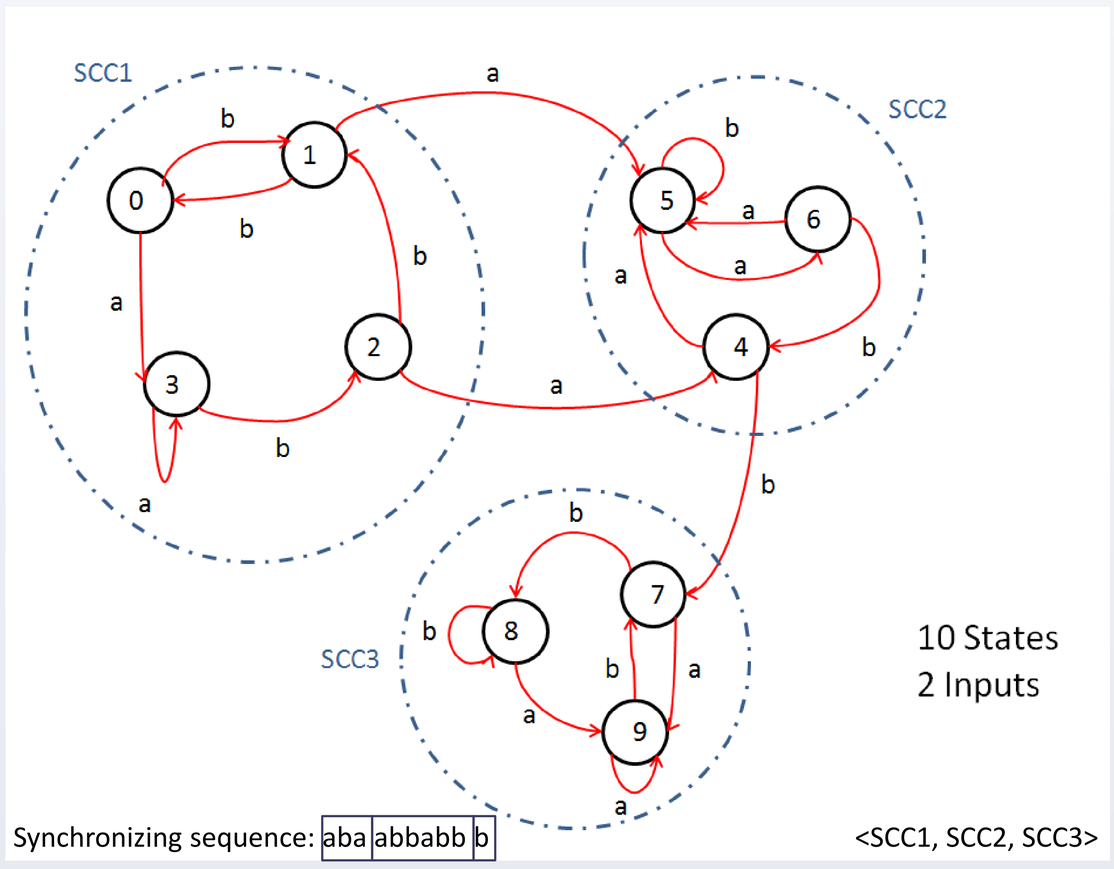
Student(s)

Berk Çirişci
Kerem M. Kahraman
Çağrı Uluç Yıldırımoğlu

Faculty Member(s)

Hüsnü Yenigün
Kamer Kaya

ABSTRACT

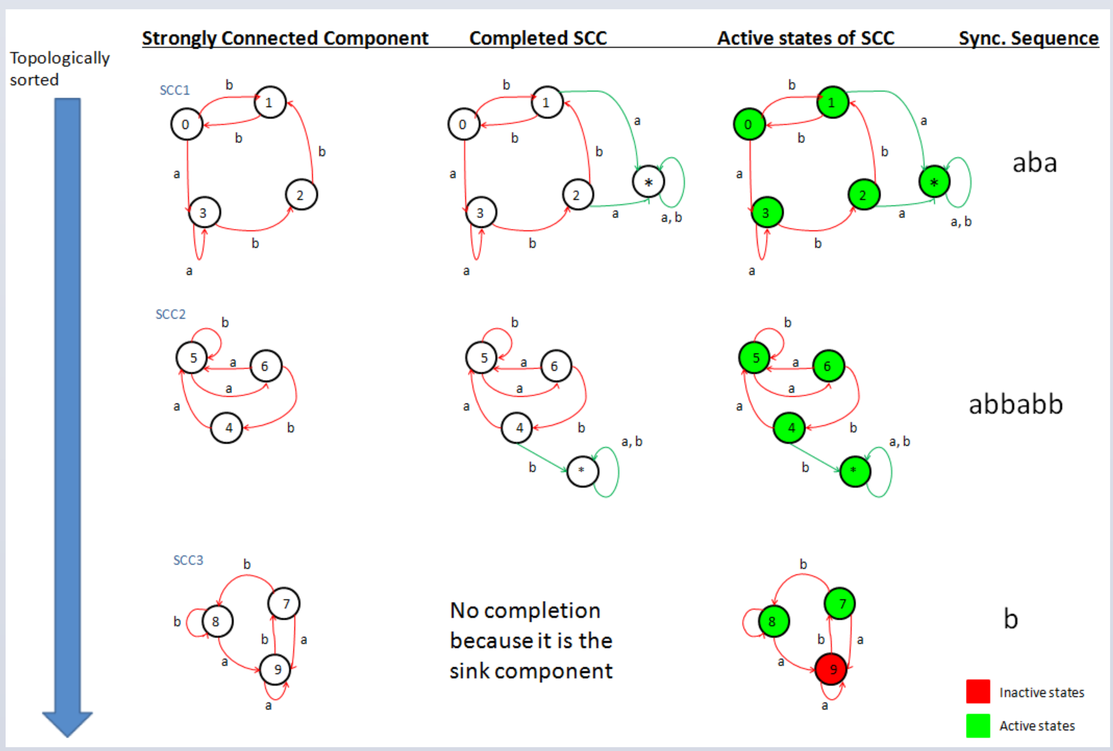


- Finding a shortest synchronizing sequence is an NP-Hard problem (Eppstein, 1990). There are heuristics (Eppstein, 1990; Roman and Szykula, 2015) to find a short synchronizing sequence.
- Some heuristics work fast but produce longer synchronizing sequences and some work slow but produce shorter synchronizing sequences.
- We propose a method for using these heuristics taking into account the connectedness of automata to make the heuristics work faster than their original versions, without sacrificing the quality of the synchronizing sequences.

OBJECTIVES

Using synchronizing heuristics with our method to make the heuristics work faster than their original versions, without sacrificing the quality of the synchronizing sequences.

PROJECT DETAILS



- A deterministic *automaton* is defined by a tuple $A = (S, \Sigma, D, \delta)$ where
  - S is a finite set of n states,
  - Σ is a finite alphabet consisting of p input letters.
  - $D \subseteq S \times \Sigma$ is called the *domain*
  - $\delta : D \rightarrow S$ is a transition function.
- When $D = S \times \Sigma$, then A is called *complete*, otherwise A is called *partial*.
- A *synchronizing sequence* w for an automaton A is a sequence of inputs such that without knowing the current state of A, when w applied to automaton A, A will have a final particular active state at the end (Eppstein, 1990).
- An automaton A is called *strongly connected* if every state is reachable from every other state by using a sequence of inputs. Otherwise, A is called as *non-strongly connected.*
- When an automaton is non-strongly connected, it can be represented as a group of strongly connected automata, called *strongly connected components* (*SCCs*).
- Given a non-strongly connected automaton A, we suggest a method to build a synchronizing sequence for A by using the synchronizing sequences of the SCCs of A.

PROJECT DETAILS II

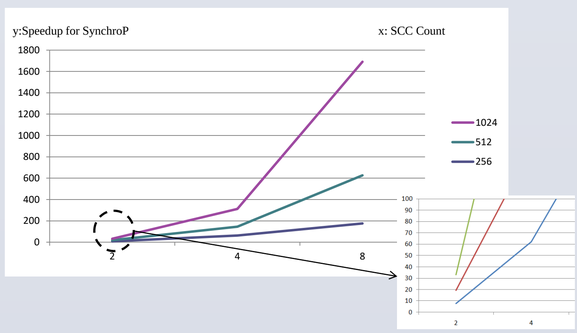
- A complete automaton $A=(S, \Sigma, S\times\Sigma, \delta)$ can be decomposed into a set of SCCs $\{A_1, A_2, \ldots, A_k\}$, where $A_i=(S_i, \Sigma, D_i, \delta_i)$, for $1 \leq i \leq k$, such that for all $1 \leq i < j \leq k$, $S_i \cap S_j = \emptyset$, and $S_1 \cup S_2 \cup \ldots \cup S_k = S$.
- An SCC $A_i=(S_i, \Sigma, D_i, \delta_i)$ is called a *sink component* if $D_i = S_i\times\Sigma$. In other words, for a sink component all the transitions of the states in S_i in A are preserved in A_i.
- Let $A=(S, \Sigma, S\times\Sigma, \delta)$ be an automaton and $\{A_1, A_2, \ldots, A_k\}$ be the SCCs of A. We consider the SCCs of A sorted as $\langle A_1, A_2, \ldots, A_k\rangle$ such that for any $1 \leq i < j \leq k$, there do not exist $s_i \in S_i$, $s_j \in S_j$, $w \in \Sigma^*$ where $\delta(s_j, w) = s_i$.

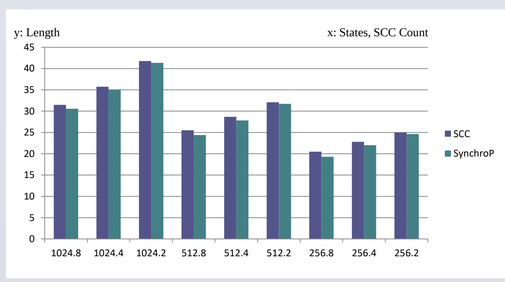
Theorem: Let $A=(S, \Sigma, S\times\Sigma, \delta)$ be a complete automaton and $\langle A_1, A_2, \ldots, A_k\rangle$ be the sorted SCCs of A, where $A_i=(S_i, \Sigma, D_i, \delta_i)$. For $1 \leq i < k$, let A'_i be the completion of A_i. $S'_k = S_k \cap \delta(S, \sigma_{k-1})$ and α_k is a S'_k-synchronizing sequence for A_k. Then $\sigma_{k-1}\alpha_k$ is a synchronizing sequence for A.

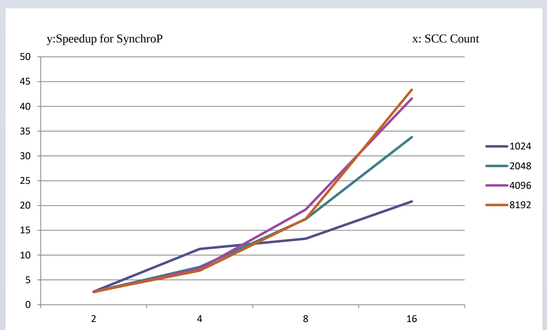
```
1 global_actives = S //All states are active initially
2 global_path = "" // Synchronizing word is initially empty
3 foreach A_x in {A_1, A_2, ... A_k} do
4     actives = global_actives ∩ A'_x // Intersect global actives with SCC to find the
          active states of that SCC after applying global path
5     path = Heuristic(A'_x, actives) // send the SCC to Heuristic to find local sync. sequence
          which is path
6     global_path = global_path + path // append global sequence with local sequence
7     global_actives = applypath(A, path) // Find actives after applying last local path
8 return global_path
```

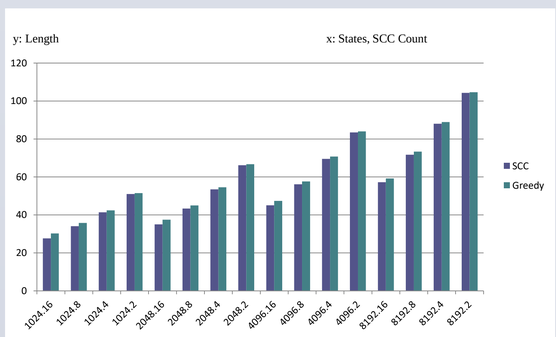
CONCLUSIONS

- Our method can be used with any synchronizing heuristic to make it work faster on non-strongly connected automata.
- The improvement in the speed increases as the number of SCCs increases.
- In case of Greedy, our method can find shorter synchronizing sequences in shorter time compared to the application of Greedy directly to automata.
- SynchroP finds shorter synchronizing sequences compared to Greedy but it takes more time. With our method, we can use SynchroP to find shorter synchronizing sequences than Greedy in a shorter time than Greedy as the number of SCCs increase.
- Greedy requires $O(n^3)$ and SynchroP requires $O(n^5)$ time, where n is the number of states.
- If there are k strongly connected components with equal sizes, the complexity of Greedy and SynchroP applied with our method becomes $O(k(n/k)^3)$, $O(k(n/k)^5)$ respectively.









REFERENCES

Altun, O. F., Atam, K.T., Karahoda, S., Kamer, K., 2017. Synchronizing Heuristics: Speeding Up The Slowest. In: IFIP International Conference on Testing Software and Systems. Lecture Notes in Computer Science 10533, Springer International Publishing

Eppstein, D., 1990. Reset sequences for monotonic automata. SIAM J. Comput. 19 (3), 500 - 510.

Karahoda, S., Erenay, O. T., Kaya, K., Türker, U. C., Yenigün, H., 2016. Parallelizing heuristics for generating synchronizing sequences. In: IFIP International Conference on Testing Software and Systems. Springer International Publishing, pp. 106 -122.

Karahoda, S., Kamer, K., Yenigün, H., 2017. Synchronizing Heuristics: Speeding Up The Fastest (*under review*)

Kudlacik, R., Roman, A., Wagner, H., 2012. Effective synchronizing algorithms. Expert Systems with Applications 39 (14), 11746-11757.

Roman, A., Szykula, M., 2015. Forward and backward synchronizing algorithms. Expert Systems with Applications 42 (24), 9512-9527.

Trahtman, A. N., 2004. Some results of implemented algorithms of synchronization. In: 10th Journees Montoises d'Inform.