Parallelisable Regular Languages

Mohan. N1*, Kalyani Desikan1, V. Rajkumar Dare2
1School of Advanced Sciences, Vellore Institute of Technology, Chennai.
2Department of Mathematics, Madras Christian College, Tambaram, Chennai.

E-mail: mohana.n@vit.ac.in

Abstract. In this paper, we have defined Parallelisable strings and languages in order to reduce the recognition time of strings. Parallelisable languages have been introduced and their recognition has been studied through parallelisable finite automaton. We have restricted our study to languages that comprise of words that contain a common subword. By introducing parallelism on the subword we have been able to reduce the recognition time. Also, we have studied the properties of parallel regular languages.

Keywords: Formal language theory, Parallelisable languages, Parallelisable regular languages, Parallelisable finite automaton.

1. Introduction
Parallel programming has some advantages that make it suitable as a solution approach for certain types of computing problems that can be best solved using multiprocessors. Nevertheless, the disadvantages of parallel programming must be taken into consideration before embarking on this challenging activity.

The main advantage of parallel programming is the efficient execution of code. Parallel programming saves time by allowing applications to be executed in a shorter duration of time. Since the code is executed efficiently, parallel programming often scales with the size of the problem and therefore, larger problems can be solved. Parallel programming provides concurrency, particularly, multiple actions being performed simultaneously.

Parallel programming surpasses the limits imposed by sequential computing [1,5,10,11]. Sequential programming is often constrained by physical and practical factors. This limits its capability to build faster sequential computers. For example, the speed of a sequential computer is dependent on the speed at which data moves through its hardware. The bandwidth of a physical medium imposes constraints on the transmissions that are made through it. (for example, the speed of light or the transmission limit of copper wire).

The process of dividing a sequential program into processes that can be run in parallel is called parallelisation. Analysing data dependencies is complex in a sequential program, and hence, it becomes difficult to parallelise efficiently. There are two possible approaches for the parallelisation of a program. The first one is to augment the sequential language such that the programmer supplies information to indicate which parts of the program are to be executed concurrently. The other approach is to explicitly write the code for each processor and mechanically identify the parallelisable
elements of a program, and consequently write compliers that automatically transform a sequential program into one that can be efficiently run on a parallel machine.

In this paper, section 2 provides the basic definitions and notations to define Parallelisable languages. Also, the mathematical definition of Parallelisable strings and languages with examples. Section 3 includes finite representation in terms of finite automaton (FA) for Parallelisable languages and their properties.

2. Preliminaries

Consider $\Sigma$ a finite, non-empty collection of alphabets. A finite sequence of symbols called a string $u$ over $\Sigma$. Length of $u$ is given by $|u|$ and defined as the number of symbols $u$. An empty string is denoted by $\epsilon$ or $\lambda$ which has zero length. Concatenation of two strings $v$ and $w$ is $v.w$ or $vw$.

$v=xuy$ is a string for some substrings $x$ and $y$. The substring $u$ is called a prefix of $v$ if $x=\epsilon$. Similarly, $u$ is a suffix of $v$ if $y=\epsilon$.

For example, consider a string $x=aabb$ then the substrings are {$a, aa, aab, abb, bb, ab, b$}. The collection $\epsilon, a, aa, aab, aabb$ are prefixes of the string $x$ and the set of suffixes of $x$ is {$aabb, abb, bb, b, \epsilon$}.

A language $L$ is a subset of $\Sigma^*$ and the collection of all strings of length $n$ over $\Sigma$ is denoted as $\Sigma^n$ where $\Sigma^0 = \epsilon$.

**Definition 2.1**

Two main definitions of regular languages:

- A language $L$ is regular if there exists a DFA or NFA $M$ such that $L(M)=L$
- A language $L$ is regular if there exists a regular expression ‘$r$’ such that $L(r)=L$

**Example: 2.2**

Consider the language $L = \{(ab)^n: n \geq 0\}$. $L$ is regular since it is accepted by the Deterministic finite automaton Figure 1 and Non-deterministic finite automaton Figure 2.

Also, $L$ can be written as a regular expression $(ab)^+$. 

![Deterministic Finite Automaton](image-url)
Parallelisable strings and Parallel languages

Parallelisation happens when we have more than one input. Parallelisation of any string of length more than one can be defined as follows:

**Definition 2.3**
If \( w = w_1 w_2 \ldots w_n \), \( w_i \in \Sigma \) and \( w \in \Sigma^* \) then
\[
P(w) = w_1||w_2||\ldots||w_n
\]
where || is an operator that is used to split or divide the string \( w \) into at least two substrings.

Parallelisation is a process by which a string is broken down into a finite number of alphabets. It is used to speed up the recognition of a string. It is obvious that, if \( w \in \Sigma \cup \{ \epsilon \} \) then parallelisation cannot happen on \( w \).

If a string of length \( n \) takes \( n \) units of time to complete through in a sequential process then parallelisation enables the string to be completed in less than \( n \) units of time.

A machine that reads a string in one unit of time is more interesting where in a string of length \( n \) is parallelised \( n \) times.

**Example: 2.4**
Let \( \Sigma = \{a, b\} \), \( x \in \Sigma^* \) and \( x = abaab \) then the parallelisable string \( P(x) \) can be written as,
\[
P(x) = a||b||a||a||b.
\]

**Definition: 2.5**
Let \( S_u = \{ x = vuvw \in \Sigma^* : u = u_1 u_2 \ldots u_n \text{ is a substring of } x, |u| > 1 \} \) then
\[
P(S_u) = \{ xP(u)y : P(u) = u_1||u_2||\ldots||u_n \} \quad \text{where } x, y \in \Sigma^* \text{ and } u_1 \epsilon \Sigma
\]
If \( L = \cup S_u \) then \( L \) is a parallelisable language.

**Example: 2.6**
If \( L = S_{aba} \), where \( aba \) can be a prefix, suffix or infix of \( L \), then \( P(L) = \{ x(a||b||a)y : x, y \in \Sigma^* \} \) where \( a||b||a \) is prefix if \( x = \epsilon \), suffix if \( y = \epsilon \) and infix if \( x, y \epsilon \Sigma^+ \)

**Example: 2.7**
If \( L = \{aa, bb\} \) then \( L \) is not a Parallelisable language. Since there is no common subword \( u \).

**Example: 2.8**
\( L = \{a, b\} \) is not a Parallelisable language. There is no subword of length greater than 1 in \( L \). Therefore, parallelisation does not exist on \( L \).

**Definition: 2.9**
Let \( L_1, L_2 \subseteq \Sigma^* \) be two parallelisable languages. Then, \( L_1||L_2 = \{ x||y : x \in L_1, y \in L_2 \} \)

**Definition: 2.10**
If \( L_1 = \Sigma^* \cup \{ u_1 \}, L_2 = u_2 \Sigma^* \) then \( L_1 L_2 = \cup (S_{u_1u_2}) \), \( P(L_1 L_2) = P(S_{u_1u_2}) \) where \( u_1, u_2 \epsilon \Sigma^+ \)

**Example: 2.11**
The collection of all strings over \( \{0, 1\} \) which contains 11 as a substring is a regular language.
\[ L = \{ u \in \{0, 1\}^*: \text{11 is a substring of } u \} \]
\[ = \{ \nu \in \{0, 1\}^n : \nu \text{w } \in \{0, 1\}^* \} \]
\[ = \{0, 1\}^* 11 \{0, 1\}^* \]
\[ P(L) = \{0, 1\}^* 1 11 \{0, 1\}^* \]

Hence, the corresponding regular expression of \( L \) is \( \{0, 1\}^* 11 \{0, 1\}^* \) and that of \( P(L) \) is \( \{0, 1\}^* 1 11 \{0, 1\}^* \).

**Example: 2.12**

The collection of all strings over \( \Sigma = \{a, b\} \) that contains \( ba \) or \( bb \) as a substring is a regular language.

\[ L = \{ u : \text{ba is a substring of } u \} \cup \{ u : \text{bb is a substring of } u \} \]
\[ = \{ v b a w : v, w \in \{a, b\}^\ast \} \cup \{ x b b y : x, y \in \{a, b\}^\ast \} \]
\[ = \{0, 1\}^* (ba) \Sigma^* \cup \Sigma^* (bb) \Sigma^* \]
\[ P(L) = \{0, 1\}^* (b|a) \Sigma^* \cup \Sigma^* (b|b) \Sigma^* \]

Hence, the corresponding regular expression of \( L \) can be expressed as \( (a+b)^* ba(a+b)^* + (a+b)^* bb(a+b)^* \) and that of \( P(L) \) is \( (a+b)^* (b|a)(a+b)^* + (a+b)^* (b|b)(a+b)^* \).

**2.2 Parallelisable Finite automaton**

Finite automata are main tools to understand regular languages better as language accepting devices. A finite automaton \( A = (Q, \Sigma, \delta, q_0, E) \) is a quintuple. Here \( \Sigma \) is the finite alphabet, \( Q \) is the finite collection of states, \( q_0 \) is an initial (start) state and \( E \subseteq Q \) is the collection of end states and the function \( \delta \) is given by \( \delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q \).

We now introduce, the transition called Parallelisable transition \( \delta_p \) for Parallelisable Finite automaton from the transition of Branching automaton [2] which is equivalent to an extended transition in Finite automaton as follows:

\[ \delta_p \subseteq \delta^* \times \delta_{\text{join}} \times \delta^* \times \delta_{\text{fork}} \times \delta^* \]

where \( \delta^*: Q \times \Sigma^* \rightarrow Q \). \( \delta_{\text{fork}} \subseteq Q \times \text{Mn}(Q) \), \( \delta \subseteq Q \times \Sigma \times Q \) and \( \delta_{\text{join}} \subseteq \text{Mn}(Q) \times Q \) and \( \text{Mn}(Q) \) is a set of non-empty, non-singleton multistates as defined in branching automaton [2] but multistates are of the form \( \{q_{i1}, q_{i2}, ..., q_{in}\} \) for some \( i \).

For example, consider the multistates created between the states \( q_{i} \) and \( q_{j} \) for some \( i \) and \( j \) for the string of length \( n \) then \( \text{Mn}(Q) = \{q_{i1}, q_{i2}, ..., q_{in}\} \) in fork transition and \( \text{M}_{\{n\}}(Q) = \{q_{j1}, q_{j2}, ..., q_{jn}\} \) in join transition.

Figure 3 represents the graphical representation (state diagram) of fork and join transitions respectively.

![Figure 3. Fork and Join Transition](image)

Formally, a Parallelisable Finite automaton \( A_p = (Q, \Sigma, \delta_p, q_0, E) \) is a quintuple. Here \( \Sigma \) is the finite collection of input alphabets, \( Q \) is the finite collection of states, \( \delta_p \) is as defined earlier, \( q_0 \) belongs to \( Q \) and \( E \) is a subset of \( Q \).
A string $w \in \Sigma^*$ is accepted by a Parallelisable Finite automaton $A_p$ if $\delta_p ((q_0,w)) \cap \mathcal{E} \neq \emptyset$. The collection of all strings recognised by the Parallelisable Finite automaton is the language accepted by $A_p$ and is written as $L(A_p) = \{ w \in \Sigma^* | \delta_p ((q_0,w)) \in \mathcal{E} \}$.

**Example: 2.13**
If $L = S_{aba}$, where $aba$ is a substring of $L$ then $P(L) = \{ x(a||b||a)y : x,y \in \Sigma^* \}$ Figure 4 depicts $P(L)$.

**Example: 2.14**
The language $L$ over $\Sigma = \{0,1\}$ that contains $01$ as a substring is regular. That is, $L = \{ x01y : x,y \in \Sigma^* \}$ is regular and $P(L) = \{ x(0||1)y : x,y \in \Sigma^* \}$ is parallel regular. Refer Figure 5.

![Figure 4. Parallelisable finite automaton $P(L) = \{ x(a||b||a)y : x,y \in \Sigma^* \}$](image)

![Figure 5. Parallelisable finite automaton $P(L) = \{ x(0||1)y : x,y \in \Sigma^* \}$](image)

### 2.3 Finite Automaton vs Parallelisable Finite Automaton

We now illustrate how a common substring is recognised in a finite automaton and its corresponding parallelisable finite automaton.

Let $\Sigma = \{a,b\}$, $L = S_{aaa}$ where $S_{aaa} = \{ x(aaa)y : x,y \in \Sigma^* \}$. Transitions for the parallelisable substring $aaa$ on FA and Parallelisable Finite Automaton are given below:

- $\delta^*(q_0, aaa) = \delta (\delta^*(q_0, a), a) = \delta (\delta (\delta^*(q_0, e), a), a) = \delta (\delta (q_1, a), a) = \delta (q_2, a)$
where $\delta^*(q_0, x)$ corresponds to the transitions of FA in Figure 6.

Given below is the corresponding transition in a parallelisable finite automaton in Figure 7

$$\delta^p(q_0, aaa) = \delta^\text{join} \, \delta \, \delta^\text{fork}(q_{\{0\}}, \text{aaa})$$

$$= \delta^\text{join} \, \delta \{(q_{\{01\}}, q_{\{02\}}, q_{\{03\}}), \text{aaa}\}$$

$$= \delta^\text{join} \{\delta(q_{\{31\}}, a), \delta(q_{\{32\}}, a), \delta(q_{\{33\}}, a)\}$$

$$= \delta^\text{join}(q_{\{31\}}, q_{\{32\}}, q_{\{33\}})$$

$$= q_{\{3\}}$$

Figure 6. Finite automaton for a string $aaa$

Figure 7. Parallelisable Finite automaton for a parallelizable string $a||a||a$

**Lemma: 2.15**

A parallelisable language $L$ is recognised by a FA $A$ in $|w|$ units of time, where $|w|$ is the length of the string $w \in L$.

**Proof:**

If $L=S_u$ then $L$ is a parallelisable language, where $S_u=\{w=xuy: u \in \Sigma^* \text{ is a substring of } w \text{ and } x,y \in \Sigma^*\}$. Consider $A=(Q, \Sigma, \delta, q_0, F)$ a FA and $w \in S_u$ then $\delta^*(q_0, w) \in E$.

$$L(A)=\{wcS_u: \delta^*(q_0, w) \in E\}$$

Assume that, $\delta(q,a)=p$ takes one unit of time for transition where $q,p \in Q$ and $a \in \Sigma$ then $\delta^*(q,w)=p$ takes $|w|$ units of time for transition where $w \in \Sigma^*$ and $q,p \in Q$.

Hence, a parallelisable language $L$ takes $|w|=|x|+|u|+|y|$ units of time in finite automaton.

**Lemma: 2.16**

For every finite automaton $A$ which accepts a parallelisable language in $|x|$ units of time, there exists a Parallelisable Finite Automaton $A_p$ which accepts the string $x$ in less than $|x|$ time units.

**Proof:**

Let $A=(Q, \Sigma, \delta, q_0, F)$ be an finite automaton which accepts a parallelisable language. Construct $A_p=(Q_p, \Sigma, \delta_p, q_0, F)$. 

\[ \delta p \subseteq \delta^* X \delta \text{join} X \delta^* X \delta \text{fork} X \delta^* \]

where \( Q_p = M_n(Q) \cup Q \).

\[ \delta^*: Q X \Sigma^* \rightarrow Q, \delta \text{fork} \subseteq Q X M_n(Q), \delta \text{join} \subseteq M_n(Q) X Q \]

and \( M_n(Q) \) is a set of non-empty, non-singleton multistates as defined in section 2.2.

We, now, prove by induction on the size of a parallelisable string \( u = u_1 u_2 \), \( P(u) = u_1 || u_2 \) and \( u \in \Sigma \).

Consider a parallelisable language \( L = S_u \) where \( u = u_1 u_2 \).

By the above lemma, \( L \) is recognized by a finite automaton \( A \) in \( |x| \) units of time where \( u \) is a subword of \( x \).

We now define a parallelisable finite automaton for accepting \( L = S_u \) where \( u = u_1 u_2 \) by parallelising \( u \) as \( P(u) = u_1 || u_2 \) as given below.

\[ \delta p(q_{(i)},u) = \delta \text{join} \delta^* \delta \text{fork}(q_{(i)}, u_{(1)} || u_2) = \delta \text{join} \delta^* \delta \text{fork}(q_{(i1)}, q_{(i2)}, u_2) = \delta \text{join}(q_{(j1)}, q_{(j2)}) \]

The above construction gives only the transition of parallelising substring. Prefix and suffix of the strings from \( L \) can be recognized by usual finite automaton.

The substring \( u = u_1 u_2 \) takes two units of time in \( A \). But, this can be recognized in one unit of time by \( A_p \).

That is, \( \delta p(q_{(i)}, x) = q_{(j)} \) if and only if \( \delta^*(q_{(i)}, x) = q_{(j)} \) for some \( i \) and \( j \).

We prove that for all \( x \in \Sigma^* \), Since parallelisation happens only on the strings of length more than one.

\[ \delta p(q_{(0)}, x) = q_{(j)} \] if and only if \( \delta^*(q_{(0)}, x) = q_{(j)} \) for some \( j \).

This proves the result, because

\[ xL(A) \Leftrightarrow \delta^*(q_{(0)}, x) \cap F \neq \phi \]

\[ \Leftrightarrow \delta p(q_{(0)}, x) \in \phi \]

Let \( u \in \Sigma^* \) with \( |u| = n \) and \( 1 \leq n \leq |x| \) By inductive hypothesis,

\[ \delta p(q_{(i)}, u) = q_{(j)} \] if and only if \( \delta^*(q_{(i)}, u) = q_{(j)} \) for some \( i \) and \( j \) and

\[ \delta \text{fork}(q_{(i)} = \{q_{(i1)}, q_{(i2)}...q_{(in)}\}) \text{and} \delta \text{join}(\{q_{(j1)}, q_{(j2)}...q_{(jn)}\}) = q_{(j)} \]

Moreover, the string \( u \) takes \( n+1 \) units of time in \( A \) and one unit of time in \( A_p \).

We generalize the above induction that any parallelisable substring \( u \) of length greater than 1 can be recognized in one unit of time in \( A_p \).

Also, a string \( x \) belongs to \( L \) is recognised by \( A_p \) in \( |x| - |u| + 1 \) units of time which less than \( |x| \).

Hence, by induction, the statement is true.

**Lemma: 2.17**

For every parallelisable Finite Automaton \( A_p \) there exists a Finite Automaton \( A \) such that \( L(A_p) = L(A) \).

**Proof:**

If \( x \) is a parallelisable string then \( \delta^*(q, x) \) is equivalent to some \( \delta^*(q, x) \). If \( P(x) = x_1 || x_2 || ... || x_n \) is accepted by \( A_p \) then \( x = x_1 x_2 ... x_n \) is a string which is accepted by \( A \).

**Theorem: 2.18**

If \( L \) is parallelisable regular then \( L \) is regular.

**Proof:**
If \( L = \cup S_u \) then \( L \) is parallelisable where \( S_u = \{ x = vuw\Sigma^* : u = u_1u_2\ldots u_n \text{ is a substring of } x, |u| > 1 \} \) is recognised by parallelisable finite automaton. Then by lemma 2.17, \( S_u \) is accepted by finite automaton. Since \( S_u \) is regular, \( L = \cup S_u \) also regular. Hence, proved

**Theorem: 2.19**
If \( L \) is a regular then \( L \) is may not be parallelisable regular.

**Proof:**
We prove the theorem by example.
Consider a regular language \( L = \{(ab)^n : n \geq 0 \} \). \( L = \{a, ab, abbb, abbbb, \ldots \} \). Here \( L \) is not parallelisable. Since it contains a string ‘a’ of length one which cannot be parallelised. Hence, \( L \) is not parallelisable and therefore \( L \) is not parallelisable regular.

**Theorem: 2.20**
If \( L \) is a non-regular language then \( L \) is not parallelisable regular.

**Proof:**
We prove the statement by giving argument. By lemma 2.15 and 2.16, for every parallelisable language we can construct finite automaton and parallelisable finite automaton. It means that if a finite automaton accepts parallelisable language then there exists a parallelisable finite automaton. Non-regular languages did not have finite automata for their recognition, so that we cannot have a parallelisable finite automaton to accept the language. Hence proved.

**Theorem: 2.21**
The set of parallelisable regular languages is a subclass of regular languages.

**Proof:**
Result is obvious by theorem 2.18, 2.19 and 2.20.

3. **Conclusion**
In this paper, we have defined parallelisable strings and languages from \( \Sigma^* \). Finite representation of parallelisable languages has been provided through parallel regular expression and parallelisable finite automaton. Parallelisable regular language has been introduced and studied their properties.

**References:**
[1] J. Hopcroft and J. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, Reading, MA, 1979.
[2] K. Lodaya and P. Weil, Series parallel posets: algebra, automata and languages, in Proc. STACS (Paris 98), LNCS 1373 (M. Morvan, C. Meinel, D. Krob, eds.) (1998) 555-565.
[3] K. Lodaya and P. Weil, Series-Parallel languages and the bounded-width property, Theoretical Computer Science, 237 (2000) 347-380.
[4] D. Kuske, Infinite series-parallel posets: logic and languages, ICALP 2000, Lecture Notes in Computer Science Vol. 1853, Springer-Verlag, 2000, 648-662.
[5] W. Thomas, Languages, automata and logic, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997, 389-455.
[6] Ivan M. Havel, Finite Branching Automata, Kybernetika, 10 (1974) 281—302.
[7] Nicolas Bedon, Logic and Branching automata, Logical methods in computer science, 11 (2015) 1—38.
[8] K. Lodaya, P. Weil, A Kleene iteration for parallelism, Foundations of Software Technology and Theoretical Computer Science, (1998) 355-366.
[9] K. Lodaya, P. Weil, Rationality in algebras with a series operation, Information and Computation,(2001) 269-293.
[10] D. Beauquier and J.E. Pin, Languages and Scanners, Theoretical Computer Science, 84 (1991) 3-21.
[11] Eilenberg. S, Automata, Languages and Machines, Vol. A (1974) Academic Press.