BLG311E Formal Languages and Automata

Mathematical Foundations of Formal Languages

A.Emre Harmancı Osman Kaan Erol Tolga Ovatman

2012

Outline

1. Inductively Defined Sets
2. Alphabets and Languages
3. Relations and Closure
   - Relations
   - Closure operations on relations
4. Languages and Grammars
5. Chomsky Hierarchy of Grammars
6. Regular Expressions

Formal Languages

Language

Language may refer either to the specifically human capacity for acquiring and using complex systems of communication, or to a specific instance of such a system of complex communication.

Formal Languages

Formal Language

In mathematics, computer science, and linguistics, a formal language is a set of strings of symbols. The alphabet of a formal language is the set of symbols, letters, or tokens from which the strings of the language may be formed which are called words.

Inductively Defined Sets

One way to build a formal language is to inductively define the set of words using a set of symbols.

Inductive Definition

A definition of a term within which the term itself appears, and that is well-founded, avoiding an infinite regress. Also called recursive definition.

Inductively Defined Sets

In order to define a set E inductively, we need to have:

- Basis: An initial set S of elements where $S \subseteq E$. Members of S are called basis elements which are used to derive members of E. An alphabet contains basis elements for natural languages.
- Rules: Functions that are used to transform elements of E into new elements.

$$\begin{aligned} \Omega = \{f_1, f_2, \ldots, f_n\} \wedge \forall f_i \in \Omega \\ f_i : E \times E \times \ldots \times E \rightarrow E \\ \forall x_1, x_2, \ldots, x_p \in E \;\; f_i(x_1, x_2, \ldots, x_p) = X \in E \end{aligned}$$

- Closure: Performing the rules on members of E always produces a member of E. In another way $\Omega(E) \subseteq E$

Inductively Defined Sets

In order to define a set E inductively, we need to have:

- Induction:

$$
\begin{aligned}
&S_0 = S \\
&S_1 = \Omega(S_0) \cup S_0 \text{ Rules applied} \\
&S_2 = \Omega(S_1) \cup S_1 \\
&\dots \\
&S_{i+1} = S_i \cup \Omega(S_i) = S_i \subseteq E \text{ (closure)}
\end{aligned}
$$

  i can be finite or infinite.

- Element height: Defined as the cardinality of the set in which an element is derived for the first time. $H(x) = min\{i | x \in S_i\}$

Inductively Defined Sets

Let's build the set of even numbers inductively:

i $0 \in E$ basis element

ii $n \in E \Rightarrow (n+2) \in E$

iii No other number can be identified as even number, apart from the rule number (ii)

Definitions

- Alphabet: A finite, non-empty set of symbols or characters. Denoted as Σ
- Word: A finite array or string from Σ
- Word Length: Number of symbols in a word.
- Empty string: A word of length 0. Denoted as Λ or ε.

Word concatenation

Concatenation is performed by joining two character strings end-to-end.

$$(x = a_1a_2 \ldots a_n) \wedge (y = b_1b_2 \ldots b_m) \Rightarrow xy = a_1a_2 \ldots a_nb_1b_2 \ldots b_m(x\&y)$$

Identity element of concatenation is Λ
$x = \Lambda \Rightarrow xy = y$
$y = \Lambda \Rightarrow xy = x$

We can build a monoid set of words(Σ^*) from the alphabet using concatenation operation having an empty string as an identity element and holding associativity property.

Reverse of a string

$(baba)^R = abab$

$(ana)^R = ana$

We can define the operation using induction:

1. $|w| = 0 \Rightarrow w^R = w = \Lambda$
2. Assume we have two strings w and u.
   $|u| = n$ $\qquad$ $|w| = n+1$
   $n \in \mathbb{N}$. Following these assumptions
   $(w = ua) \wedge (a \in \Sigma) \Rightarrow w^R = au^R$

For example:

$$\begin{aligned}
|w| = 1 &\Rightarrow w = \Lambda a \\
& w^R = a\Lambda = w \\
|w| = 2 &\Rightarrow w = ua (u = b) \\
& w^R = au^R = au = ab \\
|w| = 3 &\Rightarrow w = ua, u = cb \\
& w^R = au^R = abc
\end{aligned}$$

Reverse of a string

Theorem

$(wx)^R = x^R \cdot w^R \wedge |x| = n, |w| = m \wedge m, n \in \mathbb{N}$

Proof

Basis step:

$|x| = 0 \Rightarrow x = \Lambda$

$(wx)^R = (w\Lambda)^R = w^R = \Lambda w^R = \Lambda^R w^R = x^R w^R$

Inductive step:

Assume $|x| \leq n \Rightarrow (wx)^R = x^R w^R$

For $|x| = n + 1$

$(x = ua) \wedge (|u| = n) \wedge (a \in \Sigma) \wedge (x^r = au^r)$

$(wx)^R = (w(ua))^R = ((wu)a)^R = a(u^R w^R) = au^R w^R = x^R w^R$

For example (snow ball)^R= (ball)^R(snow)^R

Σ^+ denotes the set of non-empty strings over the Σ alphabet

i $a \in \Sigma \Rightarrow a \in \Sigma^+$

ii $(x \in \Sigma^+ \wedge a \in \Sigma) \Rightarrow ax \in \Sigma^+$

iii Σ^+ doesn't contain any elements other than the ones that can be constructed by applying i and ii finitely.

For example:
$\Sigma = \{a,b\} \Rightarrow \Sigma^+ = \{a,b,aa,ba,ab,bb,aaa,aab,\ldots\}$.

Σ^* denotes the set of all strings over the Σ alphabet

i $\Lambda \in \Sigma^*$

ii $(x \in \Sigma^* \wedge a \in \Sigma) \Rightarrow ax \in \Sigma^*$

iii Σ^* doesn't contain any elements other than the ones that can be constructed by applying i and ii finitely.

For example:

$\Sigma = \{a, b\} \Rightarrow \Sigma^* = \{\Lambda, a, b, aa, ba, ab, bb, aaa, aab, \ldots\}$

$\Sigma = \{0, 1\} \Rightarrow \Sigma^* = \{\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

Assuming $(x \in \Sigma^*) \wedge (\forall n \in \mathbb{N})$, a string's n^th power($x^n$) can be formally defined as :

1. $s^0 = \Lambda$
2. $x^{n+1} = x^n \cdot x = (x^n \& x)$

For example:
$\Sigma = \{a, b\} \ \ x = ab$
$x^0 = \Lambda, \ \ x^1 = ab, \ \ x^2 = abab, \ \ x^3 = ababab$

or
$\{a^n b^n | n \geq 0\}$ defines the set: $\{\Lambda, ab, aabb, aaabbb, \dots\}$

Let Σ be a finite alphabet.

A language over Σ is a subset of Σ^*.

The rules of choosing a subset from Σ^* can be performed by using a grammar.

For example:
$\{a^m b^n | m, n \in \mathbb{N}\}$ is a language defined over $\{a, b\}$.

This language cannot contain any other symbols than a and b.

In this language b always succeeds a. For instance ba doesn't belong to this language.

Multiplication of Languages

Let A and B be languages defined over Σ. Cartesian product $A \times B$ produces a new language.
$AB = \{xy | x \in A \wedge y \in B\}$
$AB = \{z | z = xy \wedge x \in A \wedge y \in B\}$

For example:
Let $\Sigma = \{a, b\}$ be the alphabet
Let $A = \{\Lambda, a, ab\}$ and $B = \{a, bb\}$ be the languages
$AB = \{a, bb, aa, abb, aba, abbb\}$
$BA = \{a, aa, aab, bb, bba, bbab\}$
$AB \neq BA$

Theorems

Let $\varnothing$ denote empty language; A,B,C,D denote different languages defined over the alphabet Σ.

1. $A\varnothing = \varnothing A = \varnothing$
2. $A\{\Lambda\} = \{\Lambda\}A = A$
3. $(AB)C = A(BC)$
4. $(A \subset B) \wedge (C \subset D) \Rightarrow AC \subset BD$
5. $A(B \cup C) = AB \cup AC$
6. $(B \cup C)A = BA \cup CA$
7. $A(B \cap C) \subset AB \cap AC$
8. $(B \cap C)A \subset BA \cap CA$

Selected Proofs

$A\varnothing = \varnothing A = \varnothing$

$A\varnothing = \{xy | x \in A \wedge y \in \varnothing\}$ doesn't hold since $\forall y; y \notin \varnothing$.

Since there doesn't exist any x, y couples satisfying the condition

$A\varnothing = \varnothing$

Selected Proofs

$(A \subset B) \land (C \subset D) \Rightarrow AC \subset BD$

We are going to use a direct proof. We assume

$A \subset B \land C \subset D$ and $z \in AC$

$$
\begin{aligned}
z = xy &\Leftrightarrow x \in A \land y \in C \\
(A \subset B \land x \in A \Rightarrow x \in B) &\land (C \subset D \land y \in C \Rightarrow y \in D) \\
(x \in B \land y \in D) &\Leftrightarrow xy \in BD \\
(xy \in AC \Rightarrow xy \in BD) &\Leftrightarrow AC \subset BD
\end{aligned}
$$

Selected Proofs

$A(B \cap C) \subset AB \cap AC$

$$\begin{aligned}
A(B \cap C) &\Rightarrow \forall z (z = xy \wedge x \in A \wedge y \in B \cap C) \\
x \in A \wedge y \in B \cap C &\Leftrightarrow x \in A \wedge y \in B \wedge y \in C \\
x \in A \wedge y \in B \wedge y \in C &\Leftrightarrow x \in A \wedge y \in B \wedge x \in A \wedge y \in C \\
x \in A \wedge y \in B \wedge x \in A \wedge y \in C &\Leftrightarrow xy \in AB \wedge xy \in AC \\
xy \in AB \wedge xy \in AC &\Leftrightarrow xy \in (AB \cap AC) \\
\forall z (z \in A(B \cap C) &\Rightarrow z \in AB \cap AC)
\end{aligned}$$

Let's give a counter example to show that $AB \cap AC \subseteq A(B \cap C)$ might not hold all the time.

$A = \{a^n | n \in \mathbb{N}\}$
$B = \{a^n b^n | n \in \mathbb{N}\}$
$C = \{b^n | n \in \mathbb{N}\}$

Let's take $z = a^5 b^2$. $z \in AB \cap AC$
However $B \cap C = \{\Lambda\}$ therefore
$z \notin A(B \cap C)$.

$z = a^5 b^2 = a^3 a^2 b^2$
$z \in AB \cap AC$
$z \notin A(B \cap C)$ because $B \cap C = \{\Lambda\}$

A^n : Let language A be defined over Σ

i $A^0 = \{\Lambda\}$

ii $A^{n+1} = A^n A; \forall n \in \mathbb{N}$

For example:

$\Sigma = \{a, b\}$

$A = \{\Lambda, a, b\}$

$A^1 = A$

$A^2 = \{\Lambda, a, b, aa, ab, ba, bb\}$

Theorems

Let A and B denote different languages defined over the alphabet Σ.

1. $A^m A^n = A^{m+n}$
2. $(A^m)^n = A^{mn}$
3. $A \subset B \Rightarrow A^n \subset B^n \dots$

Proofs

$A^m A^n = A^{m+n}$

We are going to use induction.

For the base case, we use the previous definition:

$n = 1 : A^m A^1 = A^{m+1}$

Inductive step: $A^m A^{n+1} = A^m A^n A^1$

Concat op. is associative: $A^m (A^n A^1) = (A^m A^n) A^1 = A^{m+n} A^1$

Proofs

$(A^m)^n = A^{mn}$

We are going to use induction.

For the base case $n = 1 : A^m = A^m$

Inductive step: $(A^m)^{n+1} = (A^m)^n (A^m)^1 = A^{mn+m} = A^{m(n+1)}$

Proofs

$A \subset B \Rightarrow A^n \subset B^n \dots$
Let's remember $A^2 = A \times A, B^2 = B \times B$ and
$A \subset B \Rightarrow (\forall x \in A \rightarrow x \in B)$
For $n = 2$:
$A^2 = \{xy | x \in A \wedge y \in A\}$
$A \subset B \Rightarrow (x \in A \rightarrow x \in B)$
$A \subset B \Rightarrow (y \in A \rightarrow y \in B)$
$(\forall xy \in A^2 \rightarrow xy \in B^2) \Rightarrow A^2 \subset B^2$
Proof continues similarly for $n := n + 1$

Star closure (Kleene Star, Kleen Closure)

Positive Closure

$A^+ : \bigcup_{n=1}^{\infty} A^n = A \cup \dots$

Star Closure

$A^* : \bigcup_{n=0}^{\infty} A^n = A^0 \cup A \cup \dots$

Theorems

Let A and B denote different languages defined over the alphabet Σ and let $n \in \mathbb{N}$.

1. $A^* = \Lambda \cup A^+$. Derived from the definition
2. $A^n \subseteq A^*, n \geq 0$. Derived from the definition
3. $A^n \subseteq A^+, n \geq 1$. Derived from the definition
4. $A \subseteq AB^*$ Proof tip: $A^0 \subseteq B^* \Rightarrow A \subseteq AB^*$

Theorems

Let A and B denote different languages defined over the alphabet Σ and let $n \in \mathbb{N}$.

5. $A \subseteq B^*A$
6. $A \subseteq B \Rightarrow A^* \subseteq B^*$. Can be proven by using $A \subseteq B \Rightarrow A^n \subseteq B^n$. Once true for all n, then it is true for union of all n.
7. $A \subseteq B \Rightarrow A^+ \subseteq B^+$
8. $AA^* = A^*A = A^+$
9. $\Lambda \in A \Leftrightarrow A^+ = A^*$
10. $(A^*)^* = A^*A^* = A^*$
11. $(A^*)^+ = (A^+)^* = A^*$
12. $A^*A^+ = A^+A^* = A^+$
13. $(A^*B^*)^* = (A \cup B)^* = (A^* \cup B^*)^*$.

Selected Proofs

$AA^* = A^*A = A^+$

$AA^* = A^+ \Rightarrow A(A^0 \cup A^1 \cup \ldots) = A \cup A^2 \cup \ldots = A^+$

Selected Proofs

$\Lambda \in A \Leftrightarrow A^+ = A^*$

First let's show $\Lambda \in A \Rightarrow A^+ = A^*$

$\Lambda \in A \Rightarrow A \cup \{\Lambda\} = A$

$A \cup A^0 = A$

$A^+ = A \cup \{\bigcup_{n=2}^{\infty} A^n\} = A^0 \cup A \cup \{\ldots\} = A^*$

Selected Proofs

$\Lambda \in A \Leftrightarrow A^+ = A^*$

Secondly $A^+ = A^* \Rightarrow \Lambda \in A$

If $A \cup A^2 \cup \ldots = A^0 \cup A \cup \ldots$

$A^0 \subseteq A \cup A^2 \cup \ldots$

This inclusion can be interpreted in two different ways:

- i $\Lambda \in A \Rightarrow A^0 \subseteq A$
- ii $\Lambda \notin A \Rightarrow \exists i, \Lambda \in A^i, i \in (\mathbb{N}^+ - 1)$

However $\Lambda \notin A \Rightarrow \forall x \in A^i (|x| \geq i)$

on the contrary $|\Lambda| = 0 \Rightarrow \Lambda \notin A^i \wedge i \geq 2$.

We have a contradiction proving ii wrong. Therefore i must be true.

Selected Proofs

$(A^*B^*)^* = (A \cup B)^*$

We need to prove both $(A^*B^*)^* \subseteq (A \cup B)^*$ and $(A \cup B)^* \subseteq (A^*B^*)^*$

$$A \subseteq A \cup B \wedge B \subseteq A \cup B$$
$$A^* \subseteq (A \cup B)^* \wedge B^* \subseteq (A \cup B)^*$$
$$A^*B^* \subseteq (A \cup B)^*$$
$$(A^*B^*)^* \subseteq ((A \cup B)^*)^*$$
$$(A^*B^*)^* \subseteq (A \cup B)^*$$

Selected Proofs

$(A^*B^*)^* = (A \cup B)^*$

We need to prove both $(A^*B^*)^* \subseteq (A \cup B)^*$ and $(A \cup B)^* \subseteq (A^*B^*)^*$

$A \subseteq A^* \subseteq A^*B^* \wedge B \subseteq B^* \subseteq A^*B^*$

It is possible to write statements above since A^* and B^* contains Λ

$(A \cup B \subseteq A^*B^*) \Rightarrow (A \cup B)^* \subseteq (A^*B^*)^*$

$(A^*B^*)^* \subseteq (A \cup B)^*$ and $(A \cup B)^* \subseteq (A^*B^*)^*$.

Theorem

Let A and B denote subsets of Σ^* and let $\Lambda \notin A$.
The only solution of $X = AX \cup B$ is $X = A^*B$

Proof

- $X = A^*B$ is a solution
  Considering $\{\Lambda\}B = B$,
  $A^*B = AA^*B \cup B = A^+B \cup B = (A^+ \cup \{\Lambda\})B = A^*B$

Proof

II Let X be a solution.
To show that $X = A^*B$ we need to show that $X \subset A^*B$ and $A^*B \subset X$ separately.

a $A^*B \subset X$.
We are going to start by proving $A^nB \subset X$ by induction.
Base condition: $n = 0, A^n = A^0 = \{\Lambda\} \Rightarrow A^0B = B$
On the other hand
$X = AX \cup B \Rightarrow AX \subset X \wedge B \subset X$
$B \subset X$ and $A^0B = B \Rightarrow A^0B \subset X$
Assumption: $A^nB \subset X$
We shall show that $A^{n+1}B \subset X$ which can be written as $AA^nB \subset X$

Proof

II Let X be a solution.
To show that $X = A^*B$ we need to show that $X \subset A^*B$ and $A^*B \subset X$ separately.

a $A^*B \subset X$.
We know that $AX \subset X$: If we concat A with solution X we have a subset of X
In $AA^nB \subset X$ we concat A with A^nB which we already assumed is a subset of X. If concatenation of A with X is a subset of X, concatenation of A with a subset of X should be a subset of X.
Therefore $\forall n, (A^nB \subset X)$ and $A^*B = \bigcup_i A^iB$
As a result we have $A^*B \subset X$

Proof

II Let X be a solution. To show that $X = A^*B$ we need to show that $X \subset A^*B$ and $A^*B \subset X$ separately.

b $X \subset A^*B$.
We are going to prove $X \subset A^*B$ by using strong induction.
$\forall x(x \in X \Rightarrow x \in A^*B)$
We assume $\forall w\{|w| < |x| \Rightarrow w \in X \Rightarrow w \in A^*B\}$.
We assume for all the words with length shorter than n.
First we need to prove the base condition.
If $|x| = 1$ then $w = \Lambda$ and $\Lambda \in A^*B$ since $\Lambda \in A^*$ and $\Lambda \in B$.
If $|x| = 2$ then since $(X = AX \cup B \wedge w \in X) \Rightarrow (w \in AX \vee w \in B)$
either $w \in AX$ which can't be the case since $\Lambda \notin A$.
Therefore it is $w \in B$ which shows $w \in A^*B$ since $\Lambda \in A^*$

Proof

II Let X be a solution. To show that $X = A^*B$ we need to show that $X \subset A^*B$ and $A^*B \subset X$ separately.

- b $X \subset A^*B$.

  We are going to prove $X \subset A^*B$ by using strong induction.
  $\forall x(x \in X \Rightarrow x \in A^*B)$
  We assume $\forall w\{|w| < |x| \Rightarrow w \in X \Rightarrow w \in A^*B\}$.
  Now we need to show for $|x| = n$
  $(X = AX \cup B \wedge x \in X) \Rightarrow (x \in AX \vee x \in B)$

  - i $x \in B \Rightarrow x \in A^*B$ satisfies $X \subset A^*B$ because $\Lambda \in A^*$
  - ii $x \in AX \Rightarrow x = yw \wedge y \in A \wedge w \in X$ because $\Lambda \notin A$
    $\Lambda \notin A \Rightarrow |y| \neq 0$
    $|y| \neq 0 \Rightarrow |w| < |x|$

  $|w| < |x| \wedge w \in X \Rightarrow w \in A^*B$ remember our assumption
  $x = yw \in AA^*B$
  $x \in A^+B \subset A^*B$

Relation

A relation is any association between elements of one set, called the domain or the set of inputs, and another set, called the range or set of outputs

Binary Relation

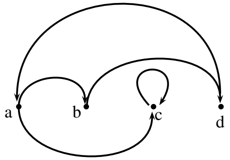
A binary relation on a set A is a collection of ordered pairs of elements of A. In other words, it is a subset of the Cartesian product $R \subseteq A \times A$

Relations can be expressed in different forms. For instance let's take the set $A = \{a, b, c, d\}$. A relation α defined over set A can be expressed using one of those ways

1. Set notation: $\alpha = \{(a,b),(a,c),(b,d),(c,c),(d,a)\}$
2. As a matrix:

$$\alpha = \begin{vmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{vmatrix}$$

3. As a relation graph:



Powers of a relation on a set

n^th power ($n \in \mathbb{N}$) of a relation α is an inductively defined set and is denoted by α^n.

i $\alpha^0 = \{(x,x)|x \in A\}$

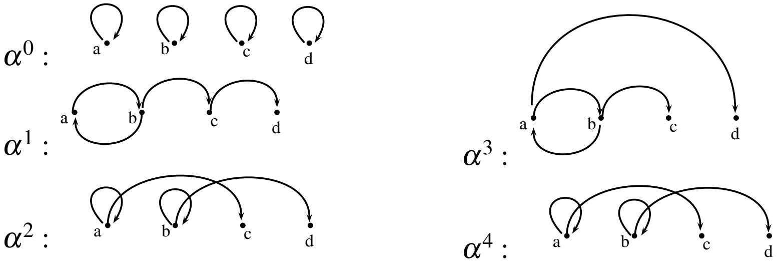
ii $\alpha^{n+1} = \alpha^n \alpha \wedge n \in \mathbb{N}$

Theorems

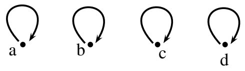
- $\alpha^m \alpha^n = \alpha^{m+n} \wedge (n,m \in \mathbb{N})$
- $(\alpha^m)^n = \alpha^{mn} \wedge (n,m \in \mathbb{N})$
- $\alpha^{-p} = (\alpha^{-1})^p = (\alpha^p)^{-1}$

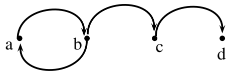
These theorems can be proven using induction.

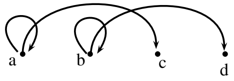
$< x, y > \in \alpha^n \Rightarrow$ x and y are connected in the relation graph and there exists at least one walk of length n. Let's consider the relations below defined over set $A = \{a, b, c, d\}$

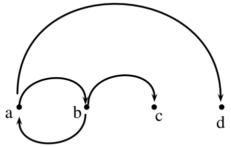


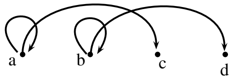
$< x,y > \in \alpha^n \Rightarrow$ x and y are connected in the relation graph and there exists at least one walk of length n. Let's consider the relations below defined over set $A = \{a,b,c,d\}$

α^0 :


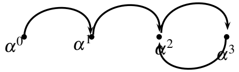
α^1 :


α^2 :


α^3 :


α^4 :


$\alpha^2 = \alpha^4 \quad \alpha^4\alpha = \alpha^2\alpha = \alpha^3$
$\alpha^5 = \alpha^3 \quad \alpha^6 = \alpha^4 = \alpha^2$
$\alpha^{2n+1} = \alpha^3 \quad \alpha^{2n} = \alpha^2$ where $n \geq 1$



Theorem

Let A be a finite set of n elements and α be a relation defined over A. There exists at least one (s,t) couple that satisfies the $\alpha^s = \alpha^t$ equality in the interval $[0, 2^{(n^2)}]$

Proof

$\alpha \subseteq A \times A$
$|A \times A|$ = n^2, $|\mathscr{P}(A \times A)| = 2^{(n^2)}$ There can be at most $2^{(n^2)}$ different relations over set A. On the other hand there are $2^{(n^2)} + 1$ powers of relation α exists in the interval $[0, 2^{(n^2)}]$. According to the pigeonhole principle there may be at least one equivalent relation couple.

Theorem

Let α be a relation defined over set A. For the relations $\alpha^s = \alpha^t$ where $s < t$ and $p = t - s$

a $\alpha^{s+k} = \alpha^{t+k}; \forall k \geq 0$

b $\alpha^{s+kp+i} = \alpha^{s+i}; \forall k > 0 \wedge \forall i (0 < i < p)$

c If $\Sigma = \{\alpha^0, \alpha^1, \ldots, \alpha^{t-1}\}$ then $\forall q \{\alpha^q \in \Sigma \wedge q \geq 0 \wedge q \in \mathbb{N}\}$

a and b can be proven by induction.

Proof of c

If $q < t$ then $\alpha^q \in \Sigma$
If $q \geq t$ then according to b $q = s + kp + i \; (i < p)$.
Therefore $\alpha^q = \alpha^{s+i}$
$i < p \Rightarrow s + i < t \Rightarrow \alpha^q \in \Sigma$

Closure operation

Closed sets

A set is closed under an operation if that operation returns a member of the set when evaluated on members of the set.

Closure operator

Given an operation on a set X, one can define the closure C(S) of a subset S in X to be the smallest subset closed under that operation that contains S as a subset

Closure operation

Reflexive closure

Let α be defined over set A. The reflexive closure α' of this relation should hold the following properties

1. α' is reflexive
2. $\alpha \subseteq \alpha'$
3. If $\alpha \subseteq \alpha''$ and α'' is reflexive then $\alpha' \subseteq \alpha''$

Symmetric Closure

Let α be defined over set A. The symmetric closure α' of this relation should hold the following properties

1. α' is symmetric
2. $\alpha \subseteq \alpha'$
3. If $\alpha \subseteq \alpha''$ and α'' is symmetric then $\alpha' \subseteq \alpha''$

Closure operation

Transitive Closure

Let α be defined over set A. The transitive closure α' of this relation should hold the following properties

1. α' is transitive
2. $\alpha \subseteq \alpha'$
3. If $\alpha \subseteq \alpha''$ and α'' is transitive then $\alpha' \subseteq \alpha''$

- Reflexive closure relation is denoted as $r(\alpha)$
- Symmetric closure relation is denoted as $s(\alpha)$
- Transitive closure relation is denoted as $t(\alpha)$

Let E be an identity relation defined over an arbitrary set A.
$E = \{< x, x > | x \in A\}, \forall R \subseteq A \times A (E = R^0)$

Theorem

$r(\alpha) = R_\alpha \cup E$[1]

For instance:

1. $r(R(<)) = R(\leq)$ ($<$ is called a strict order and $\leq$ a partial order)
2. $r(E) = E$
3. $r(R(\neq)) = A \times A$ universal relation.

Please note that $r(R(\varnothing)) = E$

[1] in a directed graph each node can have at most one loop

Theorem

If α is symmetric then $\alpha = \alpha^{-1}$

Theorem

$s(\alpha) = \alpha \cup \alpha^{-1}$
Making each arc bi-directional in a directed graph.

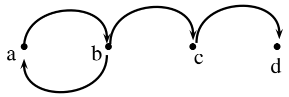
For instance:

1. $s(R(<)) = R(\neq)$
2. $s(R(\leq)) = A \times A$
3. $s(E) = E$
4. $s(R(\neq)) = R(\neq)$

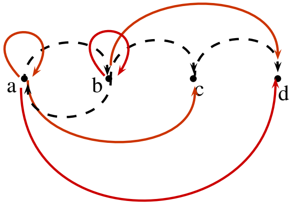
Transitive closure

Let D be the graph of relation α. The graph of $t(\alpha)$ denoted as D' can be obtained by adding an arc between each node pair in D that contains a path between them.

D:



D':



Theorem

Let α be a binary relation defined over set A.
$t(\alpha) = \bigcup_{i=1}^{\infty} \alpha^i = \alpha \cup \alpha^2 \cup \alpha^3 \cup \dots$

Proof

We need to prove $t(\alpha) \subseteq \bigcup_{i=1}^{\infty} \alpha^i$ and $\bigcup_{i=1}^{\infty} \alpha^i \subseteq t(\alpha)$ separately.

1. $\bigcup_{i=1}^{\infty} \alpha^i \subseteq t(\alpha)$ can be expressed as $\forall <a,b> \in \alpha^i \Rightarrow <a,b> \in t(\alpha)$
   We can prove using induction.
   Base condition holds by definition $\alpha \subseteq t(\alpha)$
   Inductive step, we assume $\alpha^n \subseteq t(\alpha)$
   $\forall <a,b> \in \alpha^{n+1}$ can be re-written as
   $\exists c, <a,c> \in \alpha^n \wedge <c,b> \in \alpha$
   $<a,c> \in t(\alpha)$ Since $<a,c> \in \alpha^n \subseteq t(\alpha)$
   $<c,b> \in t(\alpha)$ Since $<c,b> \in \alpha \subseteq t(\alpha)$
   $<a,b> \in t(\alpha)$ Transitivity of $t(\alpha)$
   Hence $\forall i(\alpha^i \subseteq t(\alpha))$ and therefore $\bigcup_{i=1}^{\infty} \alpha^i \subseteq t(\alpha)$

Theorem

Let α be a binary relation defined over set A.
$t(\alpha) = \bigcup_{i=1}^{\infty} \alpha^i = \alpha \cup \alpha^2 \cup \alpha^3 \cup \ldots$

Proof

We need to prove $t(\alpha) \subseteq \bigcup_{i=1}^{\infty} \alpha^i$ and $\bigcup_{i=1}^{\infty} \alpha^i \subseteq t(\alpha)$ separately.

b $t(\alpha) \subseteq \bigcup_{i=1}^{\infty} \alpha^i$
Let $< a, b > \in \alpha^s$ and $< b, c > \in \alpha^t$ be two couples in $\bigcup_{i=1}^{\infty} \alpha^i$
$< a, c > \in \alpha^{s+t}$ therefore $< a, c > \in \bigcup_{i=1}^{\infty} \alpha^i$
This makes $\bigcup_{i=1}^{\infty} \alpha^i$ transitive and by definition $t(\alpha)$ is a subset of all transitive relations that contain α
$t(\alpha) \subseteq \bigcup_{i=1}^{\infty} \alpha^i$

Theorem

Let α be a binary relation defined over set A.
$t(\alpha) = \bigcup_{i=1}^{\infty} \alpha^i = \alpha \cup \alpha^2 \cup \alpha^3 \cup \ldots$

Some examples

a Given the relation $b = a + 1 \ \forall (a, b) \in \mathbb{Z}$, transitive closure of this relation is $R(<)$.

b Let α be the $R(<)$ relation defined over a subset of integers A. Sorting the set A using $R(<)$ is performed by finding the minimal α' relation that satisfies $R(<) = t(\alpha')$. If set A is finite we know that $t(\alpha) = \bigcup_{i=1}^{2^{(n^2)}} \alpha^i$

Theorem to construct $t(\alpha)$

Theorem

Let α be defined on set A with n elements.
$t(\alpha) = \bigcup_{i=1}^{n} \alpha^i$

Proof

It is sufficient to show $\forall k(k > 0 \land \alpha^k \subset \bigcup_{i=1}^{n} \alpha^i)$
$\forall < x, y > \in \alpha^k$ statement represents all node couples that has a walk of length k between them in the directed graph D of α.
For the statement $< x, y > \in \alpha^i$, $i = n$ represents a walk consisting of all nodes forming a transitive relation.

Theorems

a If α is reflexive then $s(\alpha)$ and $t(\alpha)$ are reflexive as well.

b If α is symmetric then $r(\alpha)$ and $t(\alpha)$ are symmetric as well.

c If α is transitive then $r(\alpha)$ is transitive as well, however $s(\alpha)$ may not be transitive.

Proof of a

If α is reflexive then $\alpha = E \cup \alpha'$
$s(\alpha) = E \cup \alpha' \cup E^{-1} \cup (\alpha')^{-1} = E \cup \alpha' \cup (\alpha')^{-1} = E \cup r(\alpha')$
$t(\alpha) = \bigcup_{i=1}^{n} (E \cup \alpha)^{i}$ and $E\alpha = \alpha E = \alpha \wedge E^{n} = E$
$t(\alpha) = E \cup \alpha' \cup (\alpha')^{2} \cup \ldots \cup (\alpha')^{n}$ is a reflexive relation.

Theorems

a $rs(\alpha) = sr(\alpha)$

b $rt(\alpha) = tr(\alpha)$

c $st(\alpha) \subseteq ts(\alpha)$

Proofs

a $sr(R) = s(R \cup E) = (R \cup E) \cup (R \cup E)^{-1} = R \cup E \cup R^{-1} \cup E^{-1}$
Remember $E \cup E^{-1} = E$ and $R \cup R^{-1} = s(R)$
$sr(R) = (R \cup R^{-1}) \cup E = s(R) \cup E = rs(R)$

Theorems

a $rs(\alpha) = sr(\alpha)$

b $rt(\alpha) = tr(\alpha)$

c $st(\alpha) \subseteq ts(\alpha)$

Proofs

b $tr(R) = t(R \cup E) \wedge rt(R) = t(R) \cup E$

$ER = RE \wedge E^n = E$

$(R \cup E)^n = E \bigcup_{i=1}^{n} R$

$$\begin{aligned} tr(R) = t(R \cup E) &= R \cup E \cup (R \cup E)^2 \cup \ldots \cup (R \cup E)^n \cup \ldots \\ &= E \cup R \cup R^2 \cup \ldots \cup R^n \cup \ldots \\ &= E \cup t(R) = rt(R) \end{aligned}$$

Theorems

a $rs(\alpha) = sr(\alpha)$

b $rt(\alpha) = tr(\alpha)$

c $st(\alpha) \subseteq ts(\alpha)$

Proofs

c $R_2 \subseteq R_1 \Rightarrow s(R_2) \subseteq s(R_1) \wedge t(R_2) \subseteq t(R_1)$
Since $R \subseteq s(R) \wedge R \subseteq t(R)$
$t(R) \subseteq ts(R) \wedge s(R) \subseteq st(R)$
If $s(R)$ is symmetric then $t(s(R))$ is symmetric as well.
Hence $st(s(R)) = ts(R) \Rightarrow st(R) \subseteq ts(R)$.

Let α be a relation defined over A. $t(\alpha)$ is denoted as α^+ and $tr(\alpha)$ is denoted as α^*. These definitions are primarily used in formal languages, computation models and compiler design.
For instance:
Let $P = \{P_1, P_2, \ldots, P_n\}$ be a subroutine set in a program library. Let the $\hookrightarrow$ relation between P_i and P_j represent a subroutine call between two programs.
Under these definitions $\hookrightarrow^+$ relation can be used to represent all the subroutine calls during a program run. For example $P_i \hookrightarrow^+ P_j$ can be used to represent the P_j call during P_i's run.
On the other hand $\hookrightarrow^*$ can be used to represent the subset of programs or subroutines that can be active at a given moment during program run.
For the case $P_i \hookrightarrow^* P_j$, the statement $\forall i(P_i \hookrightarrow P_i)$ always holds for any call from P_i to P_j since P_i should be active in order to make that call. On the other hand $P_i \hookrightarrow^+ P_i$ only holds if P_i is recursive.

Formal Language

A formal language is a set of strings of symbols.
The alphabet of a formal language is the set of symbols, letters, or tokens from which the strings of the language may be formed; frequently it is required to be finite.
The strings formed from this alphabet are called words, and the words that belong to a particular formal language are sometimes called well-formed words
A formal language is often defined by means of a formal grammar such as a regular grammar or context-free grammar, also called its formation rule.

Let
S be the set of strings
S^* be the set of all string sequences
$L \subset S^*$ be the set of well-formed words
it is still a difficult problem to construct L.

Formal Grammar

A formal grammar is a set of formation rules for strings in a formal language. The rules describe how to form strings from the language's alphabet that are valid according to the language's syntax.

In our context
Σ denotes the alphabet
Σ^* is the set of all words that can be constructed using Σ
Grammar is the set of formation rules to construct a subset of Σ^*
For example:
To construct arithmetic expressions following can be used:
$\mathbb{Z} \ + \ - \ \times \ / \ (\ )$. All the well-formed arithmetic expressions are meaningful except division by zero.
$(((2-1)/3)+4\times 6)$ is a well-formed and meaningful arithmetic expression. $2+(3/(5-(10/2)))$ is well-formed as well but not meaningful because of division by zero.

The Syntax of Grammars

In the classic formalization of generative grammars first proposed by Noam Chomsky in the 1950s, a grammar G is formally defined as the tuple $(N, \Sigma, n_0, \mapsto)$:

- N is a set of *nonterminal symbols*[2].
- Σ is a set of *terminal symbols* that is disjoint from N
- n_0 is the start symbol
- $\mapsto$ is the set of production rules.
  If we call $V = N \cup \Sigma$, $\mapsto$ is a relation defined over V^*.
  It has the structure: $V^*NV^* \to V^*$
  where $*$ is a Kleene star operator.

Formal grammars are also sometimes denoted as phase structure grammars in the literature.

[2]A terminal symbol cannot be replaced with another set of symbols

The Semantics of Grammars

The operation of a grammar can be defined in terms of relations on strings:

- The direct derivability relation over a grammar $G = (N, \Sigma, n_0, \mapsto)$ on strings V^* is denoted as $\Rightarrow_G$ and formally defined as follows:
  $x \Rightarrow_G y \leftrightarrow \exists l, w, r \in V^* : (x = lwr) \wedge (w \rightarrow w' \in \mapsto) \wedge (y = lw'r)$
- The reflexive transtive closure of $\Rightarrow_G$ gives us the reachability relation $\Rightarrow^*{}_G$.
  The reachability of direct derivability denotes the words than can be derived in a finite number of steps.
  Therefore for any $\sigma \in \Sigma^*$ the statement $n_0 \Rightarrow^*{}_G \sigma$ denotes a well-formed string derived by the grammar of a language L,
  in other words $L(G) = \{\sigma | n_0 \Rightarrow^* \sigma\}$.

Example 1

S =Harry,Sally,runs,swims,fast,often,long

N =sentecte,verbal sentence,noun,verb,adverb

n_0 =sentence

Grammar:

sentence $\mapsto$ noun verbal sentence

noun $\mapsto$ Harry

noun $\mapsto$ Sally

verbal sentence $\mapsto$ verb adverb

adverb $\mapsto$ fast

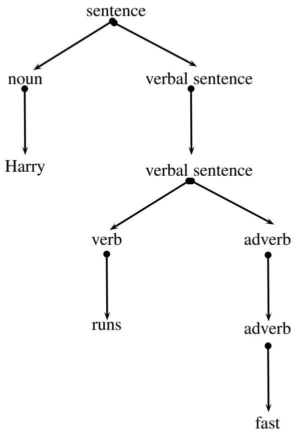
adverb $\mapsto$ often

adverb $\mapsto$ long

verb $\mapsto$ runs

verb $\mapsto$ swims

We can apply these rules in one of two ways to derive a well-formed syntax. We can either always expand the leftmost expression first or



$S =$Harry,Sally,runs,swims,fast,often,long

$N =$sentence,verbal

sentence,noun,verb,adverb

$n_0 =$sentence

Grammar:

sentence $\mapsto$ noun, verbal sentence

noun $\mapsto$ Harry

noun $\mapsto$ Sally

verbal sentence $\mapsto$ verb, adverb

adverb $\mapsto$ fast

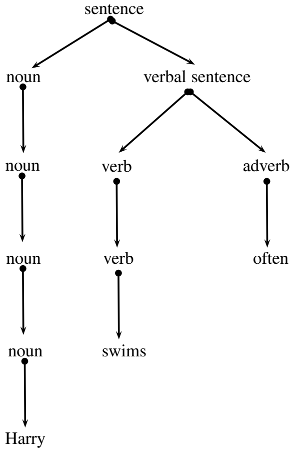
adverb $\mapsto$ often

adverb $\mapsto$ long

verb $\mapsto$ runs

verb $\mapsto$ swims

We can apply these rules in one of two ways to derive a well-formed syntax. We can either always expand the rightmost expression first



$S =$Harry,Sally,runs,swims,fast,often,long

$N =$sentence,verbal

sentence,noun,verb,adverb

$n_0 =$sentence

Grammar:

sentence $\mapsto$ noun, verbal sentence

noun $\mapsto$ Harry

noun $\mapsto$ Sally

verbal sentence $\mapsto$ verb, adverb

adverb $\mapsto$ fast

adverb $\mapsto$ often

adverb $\mapsto$ long

verb $\mapsto$ runs

verb $\mapsto$ swims

and derive a canonical parse tree.

Parsing

Parsing, or, syntactic analysis, is the process of analyzing a text, made of a sequence of tokens (for example, words), to determine its grammatical structure with respect to a given formal grammar.

Example 2

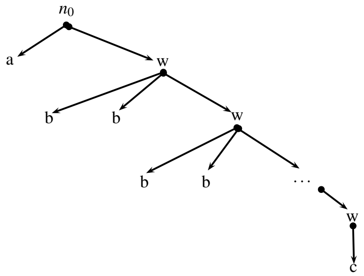
$S = a, b, c$

$N = n_0, w$

$\mapsto = \{n_0 \rightarrow aw | w \rightarrow bbw | w \rightarrow c\}$

Example 2

$S = a, b, c$
$N = n_0, w$
$\mapsto = \{n_0 \to aw | w \to bbw | w \to c\}$



$L(G) = \{a\} \cdot \{bb\}^* \cdot \{c\}$
$= a(bb)^*c$
$\{a(bb)^n c | n \in \mathbb{N}\}$

Example 3

$S = a, b, c$
$N = n_0, w$
$\mapsto = \{n_0 \to an_0b | n_0b \to bw | abw \to c\}$

$$\begin{aligned} n_0 &\Rightarrow an_0b \\ an_0b &\Rightarrow a(an_0b)b \Rightarrow a^n n_0 b^n \\ a^n(n_0b)b^{n-1} &\Rightarrow a^n bwb^{n-1} \\ a^{n-1}(abw)b^{n-1} &\Rightarrow a^{n-1}cb^{n-1} \end{aligned}$$

Or in a general form: $L(G) = \{a^m cb^m | m \in \mathbb{N}\}$

Example 4

$S = a, b, c$

$N = n_0, A, B, C$

$\mapsto = \{$

$n_0 \rightarrow A$

$A \rightarrow aABC$

$A \rightarrow abC$

$CB \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

$\}$

n_0

$\underline{\text{A}}$

$\text{a}\underline{\text{A}}\text{BC}$

$\text{aa}\underline{\text{A}}\text{BCBC}$

$\text{aaab}\underline{\text{C}}\underline{\text{B}}\text{CBC}$

$\text{aaa}\underline{\text{b}}\underline{\text{B}}\text{C}\underline{\text{CB}}\text{C}$

$\text{aaabb}\underline{\text{C}}\underline{\text{B}}\text{CC}$

$\text{aaab}\underline{\text{b}}\underline{\text{B}}\text{CCC}$

$\text{aaab}\underline{\text{b}}\underline{\text{C}}\text{CC}$

$\text{aaabb}\underline{\text{c}}\underline{\text{C}}\text{C}$

$\text{aaabbbc}\underline{\text{c}}\underline{\text{C}}$

aaabbbccc

$a^3 b^3 c^3$

$L(G) = \{a^k b^k c^k | k \in \mathbb{N}^+\}$

Chomsky Hierarchy

The Chomsky hierarchy is a containment hierarchy of classes of formal grammars. For a formal grammar $G = \{N, \Sigma, n_0, \mapsto\}$

Type 0 grammars

Type 0 grammars (unrestricted grammars) include all formal grammars.
They generate exactly all languages that can be recognized by a Turing machine.
These languages are also known as the recursively enumerable languages.
Example 3 conforms to a Type 0 grammar.

Chomsky Hierarchy

The Chomsky hierarchy is a containment hierarchy of classes of formal grammars. For a formal grammar $G = \{N, \Sigma, n_0, \mapsto\}$

Type 1 grammars

Type 1 grammars (context-sensitive grammars) have transformation rules s.t. for $w_1 \to w_2$ rule $|w_1| \leq |w_2|$ should hold.
Their rules follow the form $lwr \to lw'r$
The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton.
Example 4 conforms to a Type 1 grammar.

Chomsky Hierarchy

The Chomsky hierarchy is a containment hierarchy of classes of formal grammars. For a formal grammar $G = \{N, \Sigma, n_0, \mapsto\}$

Type 2 grammars

Type 2 grammars (context-free grammars) generate the context-free languages.
These are defined by rules of the form $A \to \gamma$ with a nonterminal(A) and a string of terminals and nonterminals(γ).
These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton.
Context-free languages are the theoretical basis for the syntax of most programming languages.
Example 1 conforms to a Type 2 grammar.

Chomsky Hierarchy

Type 3 grammars

Type 3 grammars (regular grammars) generate the regular languages.
Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a number of terminals, possibly followed by a single nonterminal.
These languages are exactly all languages that can be decided by a finite state automaton.
Additionally, this family of formal languages can be obtained by regular expressions.
Regular languages are commonly used to define search patterns.
Example 2 conforms to a Type 3 grammar.

Only Type-2 and Type-3 grammars have syntax trees.
$n_0 \rightarrow \Lambda$ rule can be added to any type to support empty sentences
Type 3 $\subseteq$ Type 2 $\subseteq$ Type 1 $\subseteq$ Type 0

Chomsky Hierarchy

Type	Language (Grammars)	Form of Productions in Grammar	Accepting Device
0	Recursively enumerable (unrestricted)	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*,$ α contains a variable)	Turing machine
1	Context-sensitive	$\alpha \to \beta$ $(\alpha, \beta \in (N \cup \Sigma)^*, \lvert\beta\rvert \geq \lvert\alpha\rvert,$ α contains a variable)	Linear-bounded automaton
2	Context-free	$A \to \alpha$ $(A \in N, \alpha \in (N \cup \Sigma)^*)$	Pushdown automaton
3	Regular	$A \to aB, A \to \Lambda$ $(A, B \in N, a \in \Sigma)$	Finite automaton

Backus-Naur Form

BNF

BNF (Backus Normal Form or Backus-Naur Form) is a notation technique for context-free grammars, often used to describe the syntax of languages used in computing, such as computer programming languages, instruction sets and communication protocols. It can be applied wherever exact descriptions of languages are needed. In the syntax of BNF

- $\mapsto$ is denoted as ::=
- non-terminals denoted inside < >
- terminals denoted as they are
- repetitons are separated by a pipe |

Backus-Naur Form

A Type 0 Grammar

$$
\begin{aligned}
< n_0 > &::= a < n_0 > b \\
< n_0 > b &::= b < w > \\
ab < w > &::= c
\end{aligned}
$$

Backus-Naur Form

A Type 1 Grammmar

$$\begin{aligned}
< n_0 > &::= < A > \\
< A > &::= a < A >< B >< C > | ab < C > \\
< C >< B > &::= < B >< C > \\
b < B > &::= bb \\
b < C > &::= bc \\
c < C > &::= cc
\end{aligned}$$

Backus-Naur Form

A Type 2 Grammmar

$$\begin{aligned} < \text{sentence} > &::= < \text{noun} >< \text{verbal sentence} > \\ < \text{noun} > &::= \text{Harry}|\text{Sally} \\ < \text{verbal sentence} > &::= < \text{verb} >< \text{adverb} > \\ < \text{verb} > &::= \text{runs}|\text{swims} \\ < \text{adverb} > &::= \text{fast}|\text{often}|\text{long} \end{aligned}$$

Backus-Naur Form

A Type 3 Grammmar

$$< n_0 > ::= a < w >$$
$$< w > ::= bb < w > | c$$

Rules of the form $w \rightarrow bbw$ are called recursive rules. If a recursive rule's non-terminal symbol is at the rightmost side it is called a normal rule.

Regular Expression

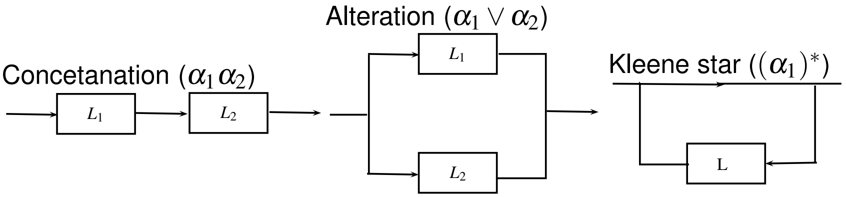
Regular expressions provide a concise and flexible means to "match" (specify and recognize) strings of text, such as particular characters, words, or patterns of characters. Common abbreviations for "regular expression" include regex and regexp. A regular expression can be defined over an alphabet Σ by induction:

1. Each and every element of Λ and Σ is a regular expression
   $L(\Lambda) = \{\Lambda\}; L(a) = \{a\}; \forall a \in \Sigma$
2. Concetanation operation$(\cdot)$ on two regular expressions produce another regular expression
   $L(\alpha\beta) = L(\alpha)L(\beta)$
3. Alternation operation$(\vee)$ on two regular expressions produce another regular expression
   $L(\alpha \vee \beta) = L(\alpha) \vee L(\beta)$
4. Kleene star operation$(*)$ on a regular expression produce another regular expression.
   $L(\alpha^*) = L^*(\alpha)$

Theorem

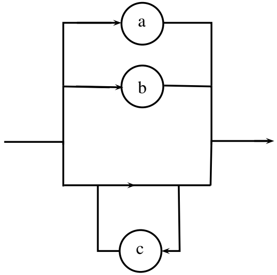
Let L be a language described over S s.t. $L \subseteq S^*$. L is called a regular language if it conforms to a regular grammar G. In other words $L = L(G)$.
Regular expressions describe regular languages in formal language theory. There exists an isomorphism between the described language and the regular expression. They have the same expressive power as regular grammars.

Syntax diagrams can be used define regular expressions. The following three diagrams describe basic properties of regular expressions.

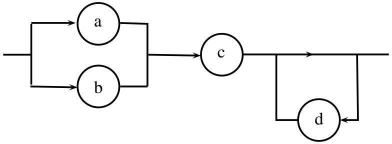

Concetanation $(\alpha_1 \alpha_2)$
L_1
L_2
Alteration $(\alpha_1 \vee \alpha_2)$
L_1
L_2
Kleene star $((\alpha_1)^*)$
L

Following are some additional examples on syntax diagrams.

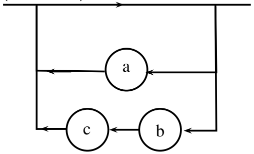
$a \vee b \vee c^*$



$(a \vee b)cd^*$



$(a \vee bc)^*$



For further practice check out the eclipse plugin Graphrex[3].

[3]Web site: http://crotonresearch.com/graphrex/