Alphabets, strings and formal languages

Lecture 6

Symbols

Definition: A **symbol** is an object endowed with a denotation (*i.e.* literal meaning)

Symbols

Definition: A **symbol** is an object endowed with a denotation (*i.e.* literal meaning)

Examples:

– Variables are symbols that have a meaning (*i.e.* denotation) and also take values

Symbols

Definition: A **symbol** is an object endowed with a denotation (*i.e.* literal meaning)

Examples:

- Variables are symbols that have a meaning (*i.e.* denotation) and also take values
- There are symbols that do not take values. Example: the letter A (as a letter not as a variable, of course)

Symbols

Definition: A **symbol** is an object endowed with a denotation (*i.e.* literal meaning)

Examples:

- Variables are symbols that have a meaning (*i.e.* denotation) and also take values
- There are symbols that do not take values. Example: the letter A (as a letter not as a variable, of course)
- A rock (on its own) is not a symbol

Concatenation and strings

Definition: Let *A* be a set of symbols. A **concatenation of symbols** of *A* is the operation of **joining** together **an n-tuple of symbols of *A*, preserving the order of the symbols in the n-tuple**.

Concatenation and strings

Definition: Let *A* be a set of symbols. A **concatenation of symbols** of *A* is the operation of **joining** together **an n-tuple of symbols of** *A*, **preserving the order of the symbols in the n-tuple**.

Definition: The object **generated** by the concatenation of an *n*-tuple of symbols in A is called **string (of length *n*) over A**

Example of string generation

Thus, in order to generate a string *s* of length *n* over a given set A, select first an ***n*-tuple**

$$(a_1, a_2, ..., a_n) \in A \times A \times \cdots \times A$$

Example of string generation

Thus, in order to generate a string *s* of length *n* over a given set A, select first an ***n*-tuple**

$$(a_1, a_2, ..., a_n) \in A \times A \times \cdots \times A$$

and then**, concatenate its elements preserving the order**:

$$s = a_1 a_2 \cdots a_n$$

Example of string generation

Thus, in order to generate a string *s* of length *n* over a given set A, select first an ***n*-tuple**

$$(a_1, a_2, ..., a_n) \in A \times A \times \cdots \times A$$

and then**, concatenate its elements preserving the order**:

$$s = a_1 a_2 \cdots a_n$$

– Example: Given $A = \{1, a, b\}$

$$\underbrace{(b, a, 1, b)}_{\text{Select an } n\text{-tuple}} \longrightarrow \underbrace{s = ba1b}_{\text{generate the string}}$$

(*n=4 in this case*)

Example of string generation

Thus, in order to generate a string *s* of length *n* over a given set A, select first an ***n*-tuple**

$$(a_1, a_2, ..., a_n) \in A \times A \times \cdots \times A$$

and then**, concatenate its elements preserving the order**:

s

This is a good model for the operation of writing

– Example: G

$$\underbrace{(b, a, 1, b)}_{\text{Select an } n\text{-tuple}} \longrightarrow \underbrace{s = ba1b}_{\text{generate the string}}$$

(*n=4 in this case*)

Properties of strings

Strings inherit all the properties of the tuples. In particular,

- The order of the symbols in the string is important
- The number of symbols in a string is important

Definition: Two **strings are different** if and only if **the tuples** used to write them **are different**

A special string

Definition: The null – string (also called empty string), denoted λ, is the string written from the tuple ( ).

A special string

Definition: The null – string (also called empty string), denoted λ, is the string written from the tuple ( ).

Observations:

1. $\lambda \neq \Phi$
2. For any string s, $s\lambda = \lambda s = s$

A special string

Definition: The null – string (also called empty string), denoted λ, is the string written from the tuple ( ).

Observations:

1. $\lambda \neq \Phi$
2. For any string s, $s\lambda = \lambda s = s$
3. $\lambda\lambda = \lambda$

All knowledge is represented in strings

Strings are fundamental to information **storage and representation**

All knowledge is represented in strings

Strings are fundamental to information **storage and representation**

- A **book** stores its content as a rather long string of alphanumeric and punctuation symbols, and a special blank space character

All knowledge is represented in strings

Strings are fundamental to information **storage and representation**

- A **book** stores its content as a rather long string of alphanumeric and punctuation symbols, and a special blank space character
- **DNA strings** are biochemical strings for storing the phenotype and other characteristics of living beings

All knowledge is represented in strings

Strings are fundamental to information **storage and representation**

- A **book** stores its content as a rather long string of alphanumeric and punctuation symbols, and a special blank space character
- **DNA strings** are biochemical strings for storing the phenotype and other characteristics of living beings
- A **computer program** stores data and instructions as a string of 0 and 1's

All knowledge is represented in strings

Strings are fundamental to information **storage and representation**

- A **book** stores its content as a rather long string of alphanumeric and punctuation symbols, and a special blank space character
- **DNA strings** are biochemical strings for storing the phenotype and other characteristics of living beings
- A **computer program** stores data and instructions as a string of 0 and 1's
- A **picture** stores an image as an array of strings of pixels

Reading a string

Definition: **Reading** a string is the process of **identifying the symbols** that compose it, one by one, **from left to right**.

Reading a string

Definition: **Reading** a string is the process of **identifying the symbols** that compose it, one by one, **from left to right**.

Since strings are written from *n*-tuples, reading may be interpreted as the reconstruction of the *n*-tuple from the string. Thus,

Reading a string

Definition: **Reading** a string is the process of **identifying the symbols** that compose it, one by one, **from left to right**.

Since strings are written from *n*-tuples, reading may be interpreted as the reconstruction of the *n*-tuple from the string. Thus,

As an operation **reading** is (somewhat) the **inverse** of **writing** (or **concatenation**)

Reading a string

Definition: **Reading** a string is the process of **identifying the symbols** that compose it, one by one, **from left to right**.

Since strings are written from *n*-tuples, reading may be interpreted as the reconstruction of the *n*-tuple from the string. Thus,

As an operation **reading** is (somewhat) the **inverse** of **writing** (or **concatenation**)

- Reading is analysis (identification of components) while concatenation is synthesis (object composition)

Illustration

Consider the string over {x, 1, Ω} s = 1x11x.

The reading of s is, step by step:

1. 1x11x → (1)

Illustration

Consider the string over {x, 1, Ω} s = 1x11x.

The reading of s is, step by step:

1. 1x11x → (1)
2. 1x11x → (1, x)

Illustration

Consider the string over {x, 1, Ω} s = 1x11x.

The reading of s is, step by step:

1. 1x11x → (1)
2. 1x11x → (1, x)
3. 1x11x → (1, x, 1)

Illustration

Consider the string over {x, 1, Ω} s = 1x11x.

The reading of s is, step by step:

1. 1x11x → (1)
2. 1x11x → (1, x)
3. 1x11x → (1, x, 1)
4. 1x11x → (1, x, 1, 1)

Illustration

Consider the string over {x, 1, Ω} s = 1x11x.

The reading of s is, step by step:

1. 1x11x → (1)
2. 1x11x → (1, x)
3. 1x11x → (1, x, 1)
4. 1x11x → (1, x, 1, 1)
5. 1x11x → (1, x, 1, 1, x)

Illustration

Consider the string over {x, 1, Ω} s = 1x11x.

The reading of s is, step by step:

1. 1x11x → (1)
2. 1x11x → (1, x)
3. 1x11x → (1, x, 1)
4. 1x11x → (1, x, 1, 1)
5. 1x11x → (1, x, 1, 1, x)
6. Halt: string is read!

This is a good model for the operation of reading!

Are write and read functions?

Let's put our models in mathematical context:

$$Write : \bigcup_{j=1}^{\infty} (\overset{j}{\underset{i=1}{\mathrm{X}}} \mathrm{A}) \times \bigcup_{j=1}^{\infty} \mathrm{A}^{j}$$

$$Write(a_1, a_2, \cdots, a_j) = \{a_1 a_2 \cdots a_j\}$$

Are write and read functions?

Let's put our models in mathematical context:

Cartesian product of j copies of A

$$Write : \bigcup_{j=1}^{\infty} (\overbrace{\mathrm{X}}^{j}_{i=1} \mathrm{A}) \times \bigcup_{j=1}^{\infty} \mathrm{A}^{j}$$

$$Write(a_1, a_2, \cdots, a_j) = \{a_1 a_2 \cdots a_j\}$$

Are write and read functions?

Let's put our models in mathematical context:

Cartesian product of j copies of A

Set of all strings of length j over A

$$Write : \bigcup_{j=1}^{\infty} \left(\bigtimes_{i=1}^{j} \mathrm{A} \right) \times \bigcup_{j=1}^{\infty} \mathrm{A}^{j}$$

$$Write(a_1, a_2, \cdots, a_j) = \{a_1 a_2 \cdots a_j\}$$

Are write and read functions?

Let's put our models in mathematical context:

Cartesian product of j copies of A

Set of all strings of length j over A

$$Write : \bigcup_{j=1}^{\infty} (\underbrace{\mathrm{X}\ \mathrm{A}}_{i=1}^{j}) \times \bigcup_{j=1}^{\infty} \mathrm{A}^{j}$$

$$Write(a_1, a_2, \cdots, a_j) = \{a_1 a_2 \cdots a_j\}$$

Write is a function since a j-tuple over A never generate more than one string!

What about read?

Here is the model:

$$\text{Read} : \bigcup_{j=1}^{\infty} \text{A}^j \times \bigcup_{j=1}^{\infty} (\overset{j}{\underset{i=1}{\text{X}}} \text{A})$$

$$\text{Read}(a_1 a_2 \cdots a_j) = \{(a_1, a_2, \cdots, a_j)\}$$

What about read?

Here is the model:

$$\text{Read} : \bigcup_{j=1}^{\infty} \text{A}^j \times \bigcup_{j=1}^{\infty} (\bigtimes_{i=1}^{j} \text{A})$$

This is not a general description!

$$\text{Read}(a_1 a_2 \cdots a_j) = \{(a_1, a_2, \cdots, a_j)\}$$

This relation **is not**, in general, **a function**

Read is not a function

Counterexample:

Let A = {1, a, b, b1}. Consider the string over A:

$$s = 1ab1a$$

Read is not a function

Counterexample:

Let A = {1, a, b, b1}. Consider the string over A:

$$s = 1ab1a$$

Then,

Read(1ab1a) = {(1, a, b, 1, a), (1, a, b1, a)}

Read is not a function

Counterexample:

Let A = {1, a, b, b1}. Consider the string over A:

$$s = 1ab1a$$

Then,

Read(1ab1a) = {(1, a, b, 1, a), (1, a, b1, a)}

Not a singleton: not a function!!

Alphabets

The **reading** of a string depends on **the set of all symbols available for writing**. Some sets render more than one reading

Alphabets

The **reading** of a string depends on **the set of all symbols available for writing**. Some sets render more than one reading

Definition: A finite, nonempty set is an **alphabet** if each string over it renders a unique reading

Write is one – to – one

Theorem: Over an alphabet, Write is one – to – one

Proof:

Let $(a_1, a_2, \ldots, a_j) \neq (b_1, b_2, \ldots, b_j)$

Then, there is i, $1 \leq j \leq I$ so that $a_i \neq b_i$,

Therefore, by definition of equality of strings:

$Write(a_1, a_2, \ldots, a_j) \neq Write(b_1, b_2, \ldots, b_j)$

An important corollary

Corollary: Over an alphabet, Read is a function and it is in fact, the inverse of Write

Proof: Since Write is one to one, for each string s there is a unique tuple t such that Write(t) = s. Therefore, Read(s) = { t }, is a singleton. Also, Read(Write(t)) = t and Write(Read(s)) = s. Therefore, Read is the inverse of Write.

Examples

Alphabets:

$A=\{a, b, c\}$, $B=\{0, 1\}$

This is called binary alphabet

$C=\{ab, ac, bc\}$, $D=\{00, 01\}$

Examples

Alphabets:

$A=\{a, b, c\}$, $B=\{0, 1\}$

$C=\{ab, ac, bc\}$, $D=\{00, 01\}$

Non-alphabet:

$E=\{a, b, ab\}$

Examples

Alphabets:

$A=\{a, b, c\}$, $B=\{0, 1\}$

$C=\{ab, ac, bc\}$, $D=\{00, 01\}$

Non-alphabet:

$E=\{a, b, ab\}$

$A \cup C = \{a, b, c, ab, ac, bc\}$

Remark: As the last example shows: the union of two alphabets is not always an alphabet!!! (**Despite what people may say!!!**)

Are these artificial examples?

The definition of alphabets containing strings of symbols may look like as an unnatural way of getting a non – alphabet

Are these artificial examples?

The definition of alphabets containing strings of symbols may look like as an unnatural way of getting a non – alphabet

This is definitely not the case! Convincing examples are right ahead!

Are these artificial examples?

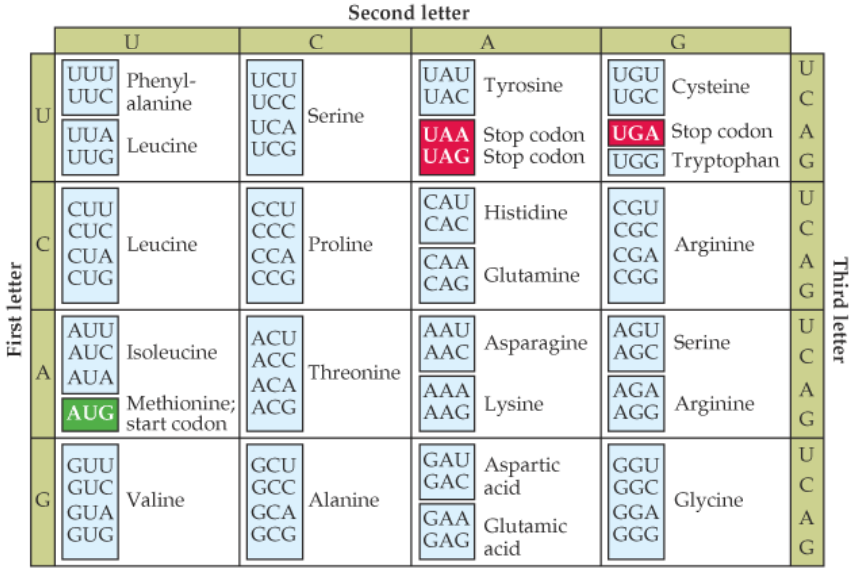
The definition of alphabets containing strings of symbols may look like as an unnatural way of getting a non – alphabet

This is definitely not the case! Convincing examples are right ahead!

It is worth pointing out that, in general, proving that a set is indeed an alphabet may be difficult: not only sets that contain strings produce more than one reading!

Some biochemical alphabets

- **DNA alphabet**:
  - D = {A, T, G, C}
- **RNA alphabet**:
  - R = {A, U, G, C}
- **Protein transcription alphabet**: symbols are “codons”. Each codon is formed with three RNA symbols

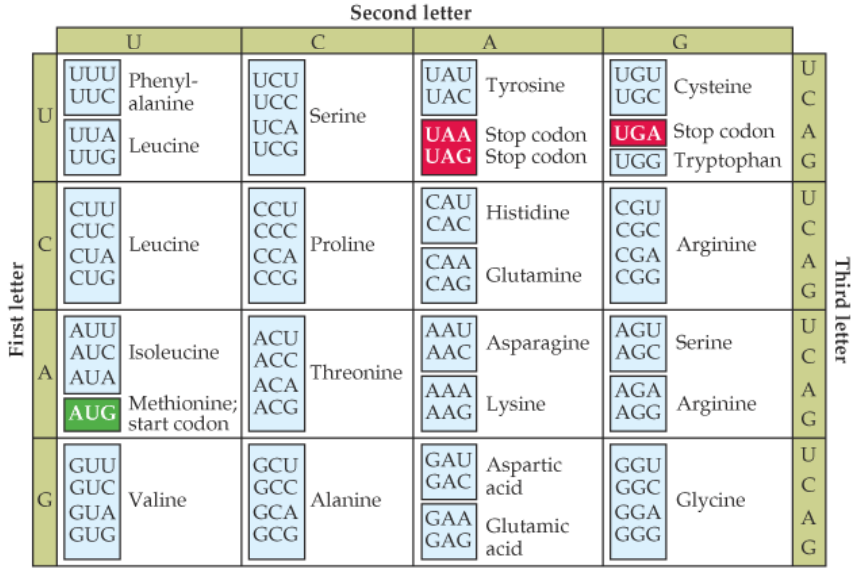


First letter	Second letter: U	Second letter: C	Second letter: A	Second letter: G	Third letter
U	UUU, UUC: Phenylalanine	UCU, UCC, UCA, UCG: Serine	UAU, UAC: Tyrosine	UGU, UGC: Cysteine	U C A G
	UUA, UUG: Leucine		UAA, UAG: Stop codon Stop codon	UGA: Stop codon; UGG: Tryptophan	
C	CUU, CUC, CUA, CUG: Leucine	CCU, CCC, CCA, CCG: Proline	CAU, CAC: Histidine	CGU, CGC, CGA, CGG: Arginine	U C A G
			CAA, CAG: Glutamine		
A	AUU, AUC, AUA: Isoleucine	ACU, ACC, ACA, ACG: Threonine	AAU, AAC: Asparagine	AGU, AGC: Serine	U C A G
	AUG: Methionine; start codon		AAA, AAG: Lysine	AGA, AGG: Arginine	
G	GUU, GUC, GUA, GUG: Valine	GCU, GCC, GCA, GCG: Alanine	GAU, GAC: Aspartic acid	GGU, GGC, GGA, GGG: Glycine	U C A G
			GAA, GAG: Glutamic acid		

Some biochemical alphabets

- **DNA alphabet**:
  - D = {A, T, G, C}
- **RNA alphabet**:
  - R = {A, U, G, C}
- **Protein transcription alphabet**: symbols are “codons”. Each codon is formed with three RNA symbols

Here is an example of a natural alphabet that contains strings!



First letter	Second letter: U	Second letter: C	Second letter: A	Second letter: G	Third letter
U	UUU, UUC: Phenylalanine	UCU, UCC, UCA, UCG: Serine	UAU, UAC: Tyrosine	UGU, UGC: Cysteine	U C A G
	UUA, UUG: Leucine		UAA, UAG: Stop codon Stop codon	UGA: Stop codon; UGG: Tryptophan	
C	CUU, CUC, CUA, CUG: Leucine	CCU, CCC, CCA, CCG: Proline	CAU, CAC: Histidine	CGU, CGC, CGA, CGG: Arginine	U C A G
			CAA, CAG: Glutamine		
A	AUU, AUC, AUA: Isoleucine	ACU, ACC, ACA, ACG: Threonine	AAU, AAC: Asparagine	AGU, AGC: Serine	U C A G
	AUG: Methionine; start codon		AAA, AAG: Lysine	AGA, AGG: Arginine	
G	GUU, GUC, GUA, GUG: Valine	GCU, GCC, GCA, GCG: Alanine	GAU, GAC: Aspartic acid	GGU, GGC, GGA, GGG: Glycine	U C A G
			GAA, GAG: Glutamic acid		

Languages

Definition: A **language** or, more specifically, a **formal language over an alphabet** *A* is a set of strings over *A* or the empty set

Languages

Definition: A **language** or, more specifically, a **formal language over an alphabet** *A* is a set of strings over *A* or the empty set

Examples:

1. Every alphabet is a formal language over itself

Languages

Definition: A **language** or, more specifically, a **formal language over an alphabet** *A* is a set of strings over *A* or the empty set

Examples:

1. Every alphabet is a formal language over itself
2. The set of natural numbers is a language over the alphabet *A={0, 1, 2, …,9}*

More examples of languages

- Let *A={0, 1}* then *A* is a language over *A*
- Languages over the binary alphabet *{0, 1}* are pervasive in computation
- $A=\{\lambda\}$ is a language (and an alphabet)
- Natural languages are formal languages that in addition to **denotation** have **connotation** (intended meaning)

Basic facts about languages

Theorem 5.3*:*

(a) The union of two languages over the same alphabet is a language

(b) The intersection of two languages over the same alphabet is a language

(c) The difference of two languages over the same alphabet is a language

Proof of (a)

Let L and M be languages over the same alphabet A.

Then, $(\forall s)(s \in L \cup M) \Rightarrow s \in L \vee s \in M$

$\Rightarrow s$ is a string over A

Thus, $L \cup M$ is a language over A.

Proof of (b)

Let L and M be languages over the same alphabet A.

Then, $(\forall s)(s \in L \cap M) \Rightarrow s \in L \wedge s \in M$

$\Rightarrow s$ is a string over A

Thus, $L \cap M$ is a language over A.

Proof of (c)

Let L and M be languages, $L \subseteq M$.

Then, $(\forall s)(s \in M)$ s is a string over A

Since $M - L \subseteq M$

$(\forall s)(s \in M - L \Rightarrow s \in M)$

Therefore, $(\forall s)(s \in M - L$ is a string over $A)$

String concatenation

Definition: Let *L* and *S* be nonempty languages. The **string concatenation over *L*** is defined to be the operation

$$Cat(s, t)=st$$

We define $Cat(L)= \{st : s \in L \wedge t \in L\}$

String concatenation

Definition: Let *L* and *S* be nonempty languages. The **string concatenation over *L*** is defined to be the operation

$$Cat(s, t)=st$$

We define *Cat(L)*= $\{st : s \in L \wedge t \in L\}$

Example*:* Let *L* be the language of all strings over the alphabet *A={a, b, 1}.* Let's take *s=abaab* and *t=1a1ab1* (two strings in *L).* Their concatenation is

$$Cat(s,t)=st=abaab1a1ab1$$

Properties of concatenation

Associativity $(\forall s,t,r)(s,t,r \text{ strings over } A)$

$$Cat(s,Cat(t,r)) = Cat(Cat(s,t),r)$$

Non-commutativity

$$(\exists s,t)(s,t \text{ strings over } A) \wedge Cat(s,t) \neq Cat(t,s)$$

Existence of an identity (null string)

$$(\exists \lambda)(\lambda \text{ string}) \wedge ((\forall s)(s \text{ string over } A)$$

$$Cat(\lambda,s) = Cat(s,\lambda) = s)$$

Observation: the null string **is not** the empty set

Closure under string concatenation

Definition: A language *L* is said to be **closed under string concatenation** if *L=Cat(L)*

Examples:

- The language $L = \{\lambda\}$ is closed under string concatenation
- The language $L = \phi$ is also closed under string concatenation (Why?)
- The language *L* of all strings over an alphabet is closed under string concatenation

Closure under string concatenation

Definition: A language *L* is said to be **closed under string concatenation** if *L=Cat(L)*

Examples:

– The language $L = \{\lambda\}$ is closed under string concatenation

Closure under string concatenation

Definition: A language *L* is said to be **closed under string concatenation** if *L=Cat(L)*

Examples:

- The language $L = \{\lambda\}$ is closed under string concatenation
- The language $L = \phi$ is also closed under string concatenation (Why?)

Closure under string concatenation

Definition: A language *L* is said to be **closed under string concatenation** if *L=Cat(L)*

Examples:

- The language $L = \{\lambda\}$ is closed under string concatenation
- The language $L = \phi$ is also closed under string concatenation (Why?)
- The language *L* of all strings over an alphabet is closed under string concatenation

More examples

The language L = {0, 00, 000, … } = $\{0^n : n \geq 1\}$

is closed under concatenation. In order to see this, take any two strings in L, this is $s = 0^n$ and $t = 0^m, s, t \geq 1.$

Then, $st = 0^n 0^m = 0^{n+m} \in L$

The language $L = \{01^n : n \geq 1\}$ **is not closed under string concatenation** since, for instance: $Cat\,(01,011) = 01011 \notin L$

Why does closure matters?

String concatenation is pervasive as a language operation If a language is closed under string concatenation, the operation of concatenation is a **language invariant** (*i.e.* does not alter the language). In particular, if the language is closed under string concatenation, any concatenation satisfies the predicate that defines membership in the language

Example:

$Q(w):"\text{w string over}\{0,1\}\wedge w \text{ ends with } 0"$

$L=\{w:Q(w)=1\}$

$\text{Now, since the concatenation of any finite collection of strings over } \{0,1\} \text{ ending with } 0 \text{ is a string that ends with } 0, L \text{ is closed under string concatenation. Therefore,}$

$(\forall s,w\in L)Q(sw)=1$

Theorem on string concatenation

Theorem 5.4: If *L* is a nonempty language that is closed under string concatenation, then *L* is finite if and only if $L = \{\lambda\}$.

Proof: If $L = \{\lambda\}$, since $Cat(\lambda, \lambda) = \lambda\lambda = \lambda$ we have that $Cat(L) = \{\lambda\} = L$. Therefore, *L* is closed under string concatenation, and finite.

If $L \neq \{\lambda\}$, then there exists $s \in L \wedge s \neq \lambda$.

Proof (cont.)

Since *L* is closed under string concatenation the following inductive construction produces a infinite subset of *L*

Base case:

$Cat(s,s) = ss = s^2 \in L$ (true because *L* is closed)

Induction:

$(\forall n) Cat(s, s^{n-1}) = s^n \in L \Rightarrow Cat(s, s^n) = s^{n+1} \in L$

(true because *L* is closed)

Thus, the set $\{s, s^2, ..., s^n, ...\} = \{s^n : n \geq 1\} \subseteq L$ is an infinite subset of *L.* And therefore, *L* is infinite.

Closure of a language

Definition: Let A be an alphabet and L be a language over A. Then, the **star – closure** (or Klein – closure) of L is the smallest language over A that contains L and is closed under string concatenation. The star – closure of L is denoted L*.

Remark: In general, L is a subset of L*. In fact, L = L* if and only if L is star – closed

Star – closure of an alphabet

Theorem: Let A be an alphabet. Then, any language L over A is a subset of A*

Proof: Let s be a string in L. Since L is a language over A, s is the result of a concatenation of symbols in A. But then, s is an element of A*

Encodings

Let A, B be two alphabets. An **encoding of** *A* **in** *B* is a **one-to-one** mapping

$$e : \mathrm{A} \rightarrow \mathrm{B}^*$$

satisfying the **additional condition** that the set

$$e(\mathrm{A}) = \{w : w \in \mathrm{B}^* \wedge (\exists x \in \mathrm{A}) \wedge e(x) = w\}$$

is also an **alphabet**.

Encodings

Let A, B be two alphabets. An **encoding of** *A* **in** *B* is a **one-to-one** mapping

$$e : \mathrm{A} \rightarrow \mathrm{B}^*$$

satisfying the **additional condition** that the set

$$e(\mathrm{A}) = \{w : w \in \mathrm{B}^* \wedge (\exists x \in \mathrm{A}) \wedge e(x) = w\}$$

is also an **alphabet**.

Example: Let $\mathrm{A} = \{0,1\}; \mathrm{B} = \{a, b\}$. Then,

$e(0) = a; e(1) = b$ is an encoding, but

$e(0) = ab; e(1) = abab$, is NOT

Popular examples of encodings

ASCII: The ASCII encoding specifies a one-to-one correspondence between an alphabet of letters and punctuation symbols of a written language and a set of binary strings (i.e. strings over {0, 1})

Some values are:

e(A)=1000001

e(B)=1000010

e(C)=1000011,…etc,

e(,)=01011100

e(.)= 01011110,…etc.

Popular examples of encodings

ASCII: The ASCII encoding specifies a one-to-one correspondence between an alphabet of letters and punctuation symbols of a written language and a set of binary strings (i.e. strings over {0, 1})

Some values are:

e(A)=1000001

e(B)=1000010

e(C)=1000011,…etc,

e(,)=01011100

e(.)= 01011110,…etc.

Hexadecimal numbers: the encoding specifies a one-to-one correspondence between the alphabet

A={0,1,2,…,9,A,B,C,D,E,F}

And strings over the alphabet

B={0,1,2,…,9}. The encoding is:

e(0)=0

e(1)=1,…,e(9)=9,

e(A)=10, e(B)=11, e(C)=12,

e(D)=13, e(E)=14, e(F)=15

Knowledge representation is independent of alphabets

Encodings are **alphabet translators**. Their existence ensures that **the ability to represent objects does not depend on the choice of an alphabet**

Knowledge representation is independent of alphabets

Encodings are **alphabet translators**. Their existence ensures that **the ability to represent objects does not depend on the choice of an alphabet**

Indeed, if A, B are two different alphabets, and $w = a_1 a_2 ... a_n$ is a string over A .Then, its "translation" as a string over B is

$$e(a_1)e(a_2)...e(a_n)$$

A simple illustration

The word in Spanish language:

CABA

is encoded by ASCII over {0, 1}* as

e(C)e(A)e(B)e(A)

= **1000011100000110000101000001**