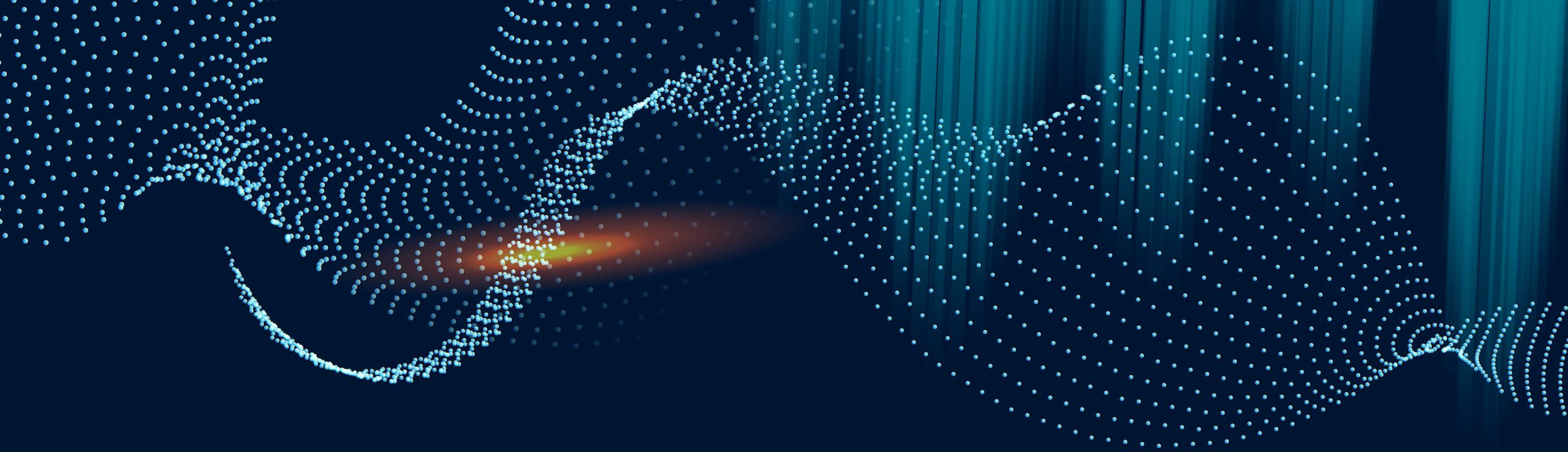
LANGUAGES

A general definition

Gilberto Echeverria
02 / 2021



02

Definition of languages

Based on mathematical formulations

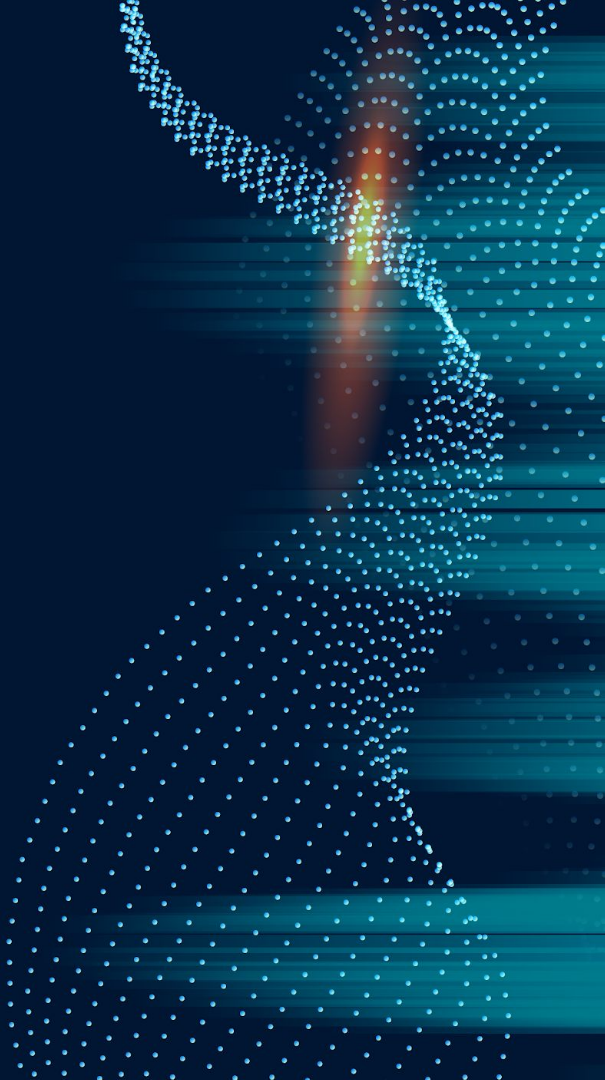
General concepts

Many kinds of languages

- Natural Languages
- Mathematical
- Computer

A general definition: a set of strings over an alphabet

- Alphabet are the symbols in the language
- String is a finite sequence of symbols in a language



General concepts

Examples:

English language:

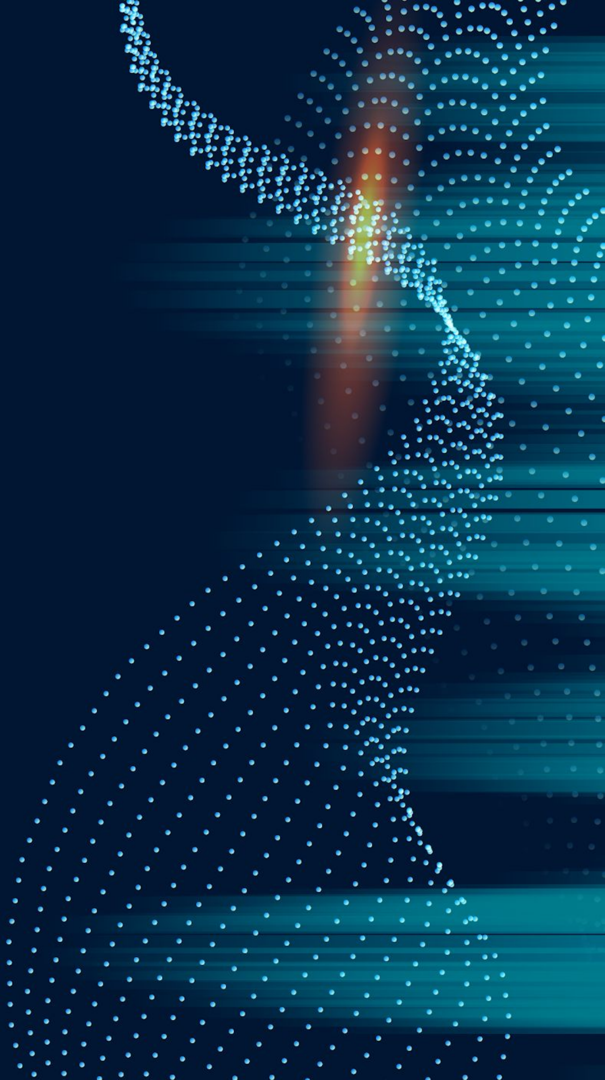
This is a nice day

Mathematical language:

$X = (5 * y - 8) / r$

Computer language:

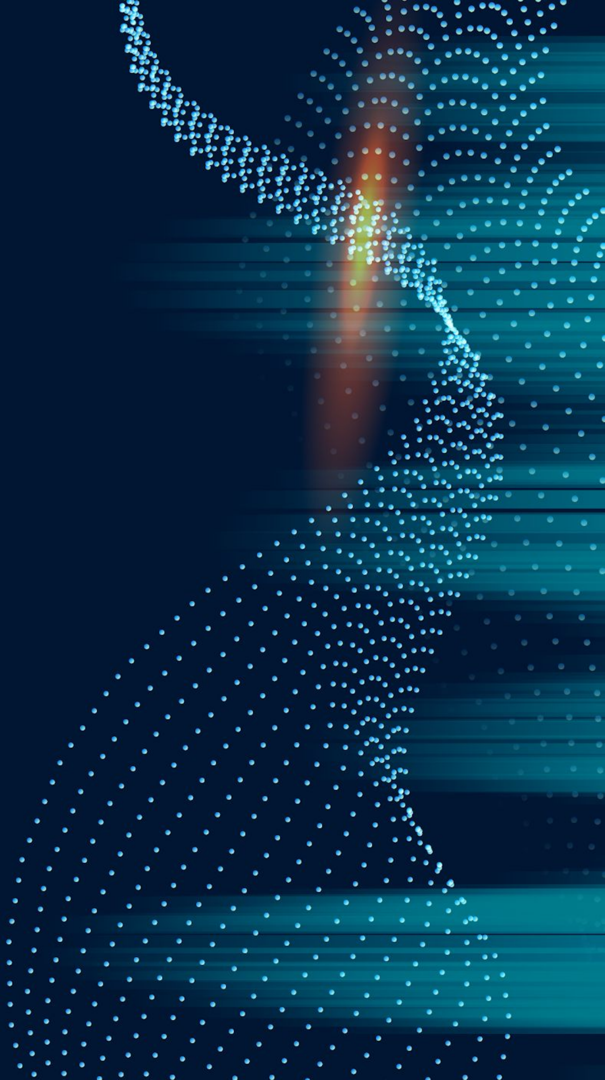
```
printf(“The result is: %5.2f\n”, cos(angle) / m);
```



Languages

Topics to be covered

1. Strings and Languages
2. Finite Specifications of Languages
3. Regular Sets and Expressions
4. Regular Expressions and Text Searching



Strings and Languages

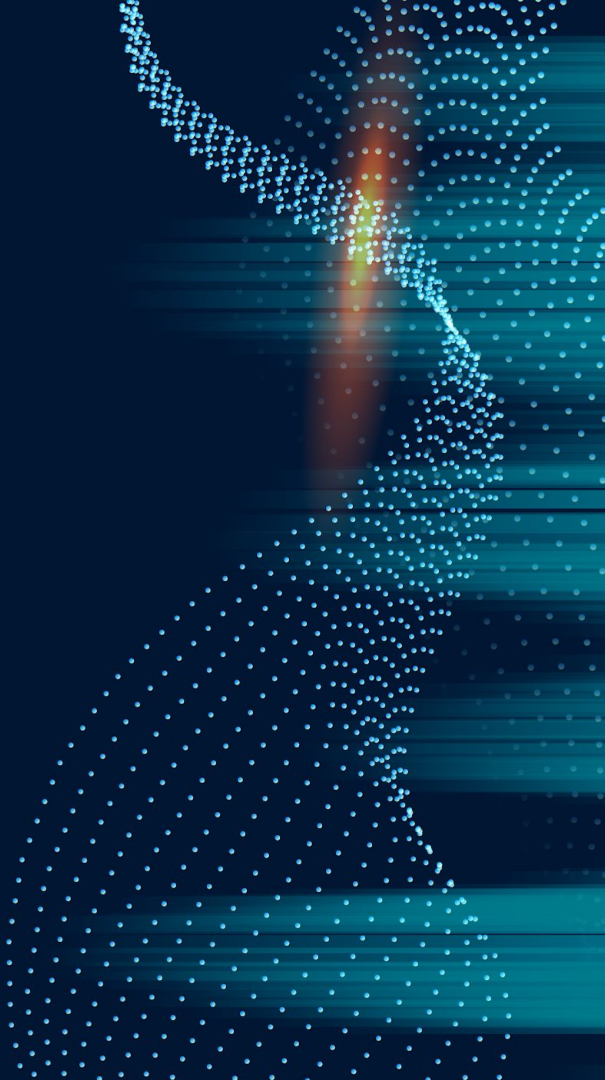
Strings and Languages

An alphabet is a list of indivisible objects. For human languages these are words and punctuation

A string is the sequence of words and punctuation.

Notation:

- Alphabet: **Σ**
- Elements in an alphabet: *a*, *b*, *c*, *d*
- Strings from an alphabet: *p*, *q*, *u*, *v*, *w*, *x*, *y*, *z*



Strings and Languages

Recursive definition of a language:

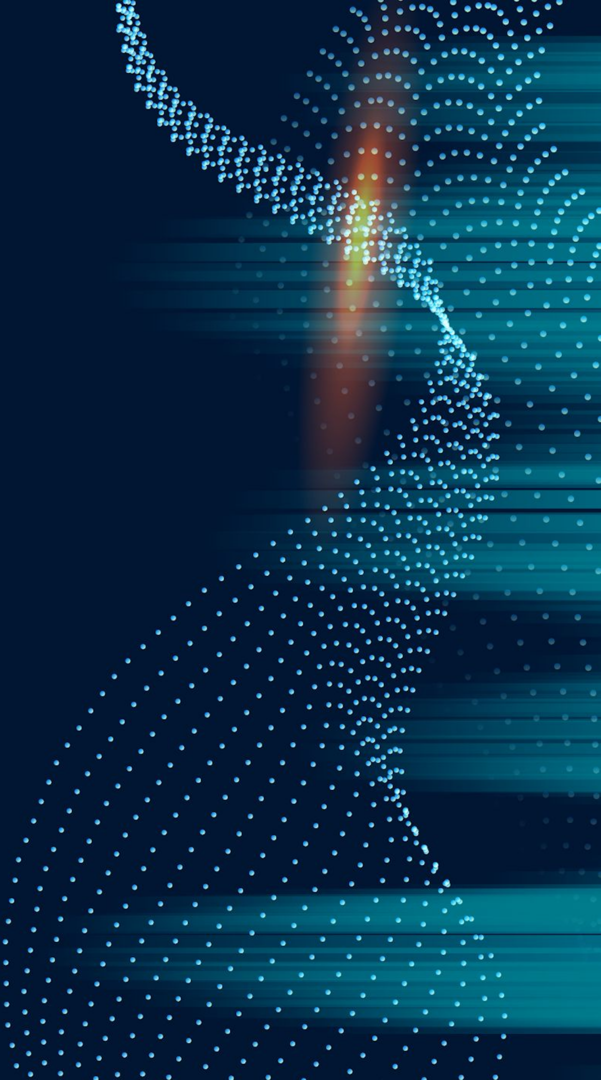
A string with nothing is called null string: **λ**

Let **Σ** be an alphabet. The set of strings over **Σ** is **Σ***

I. **Basis**: **λ** ∈ **Σ***

II. **Recursive step**: If *w* ∈ **Σ*** and *a* ∈ **Σ**, then *wa* ∈ **Σ***

III. **Closure**: *w* ∈ **Σ*** only if it can be obtained from **λ** from a finite number of applications of the recursive step

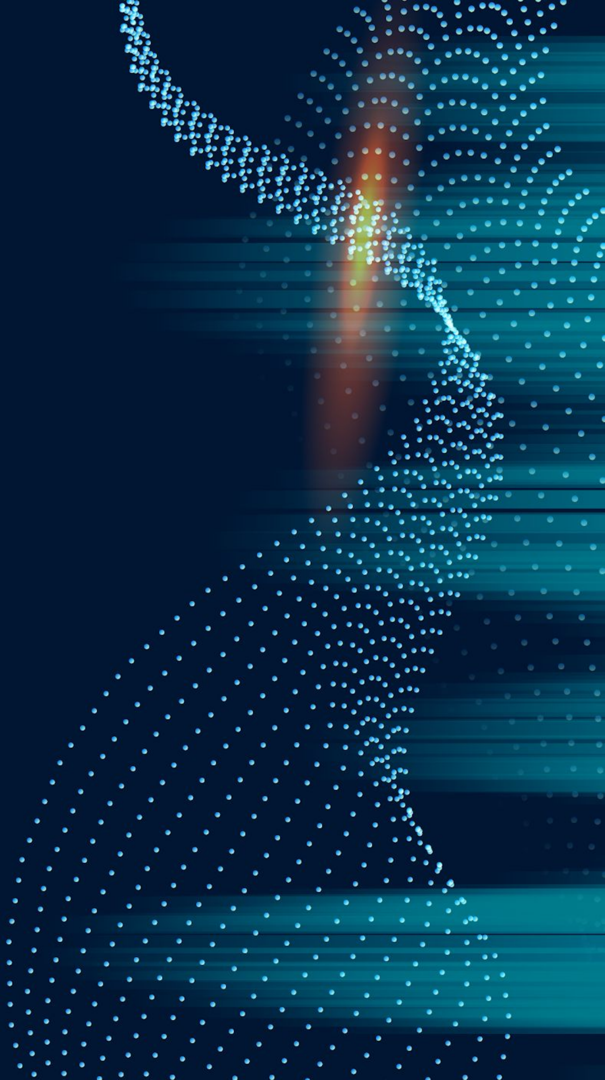


Strings and Languages

Length of a string w: length(w)

Corresponds to the number of applications of the recursive step

If $\mathbf{\Sigma}$ contains n elements, there are n^k strings of length k in $\mathbf{\Sigma^*}$



Examples:

For an alphabet $\mathbf{\Sigma} = \{a, b, c\}$

The elements in $\mathbf{\Sigma}^*$ include:

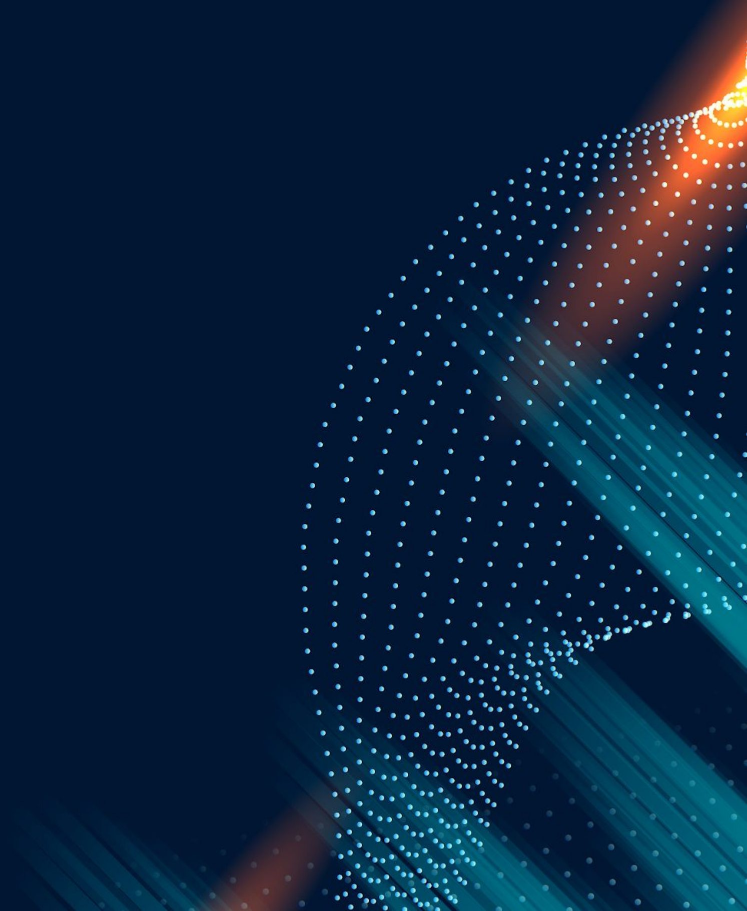
Length 0: $\boldsymbol{\lambda}$

Examples:

For an alphabet **Σ** = {*a*, *b*, *c*}

The elements in **Σ*** include:

Length 0:	**λ**		
Length 1:	*a*	*b*	*c*



Examples:

For an alphabet **Σ** = {*a*, *b*, *c*}

The elements in **Σ*** include:

Length 0:	**λ**								
Length 1:	*a*	*b*	*c*						
Length 2:	*aa*	*ab*	*ac*	*ba*	*bb*	*bc*	*ca*	*cb*	*cc*

Examples:

For an alphabet **Σ** = {*a*, *b*, *c*}

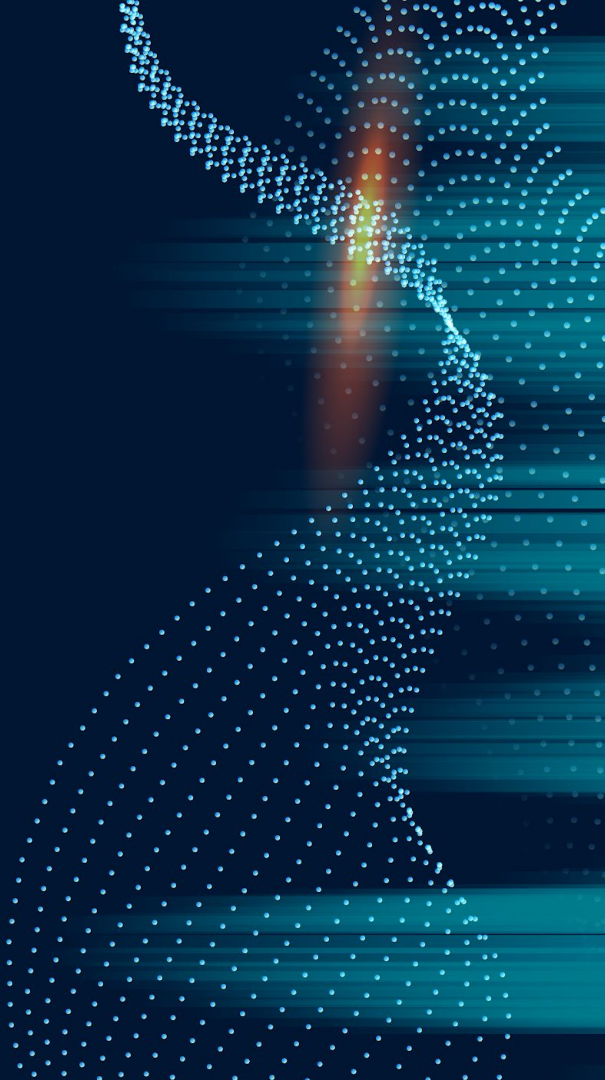
The elements in **Σ*** include:

Length 0:	**λ**								
Length 1:	*a*	*b*	*c*						
Length 2:	*aa*	*ab*	*ac*	*ba*	*bb*	*bc*	*ca*	*cb*	*cc*
Length 3:	*aaa*	*aab*	*aac*	*aba*	*abb*	*abc*	*aca*	*acb*	*acc*
	baa	*bab*	*bac*	*bba*	*bbb*	*bbc*	*bca*	*bcb*	*bcc*
	caa	*cab*	*cac*	*cba*	*cbb*	*cbc*	*cca*	*ccb*	*ccc*

Strings and Languages

The number of possible strings is infinite, but not all strings are valid. The rules to determine a correct string are the syntax of a language.

A language over alphabet **Σ** is a subset of the set of all possible strings over the alphabet **Σ***



Strings and Languages

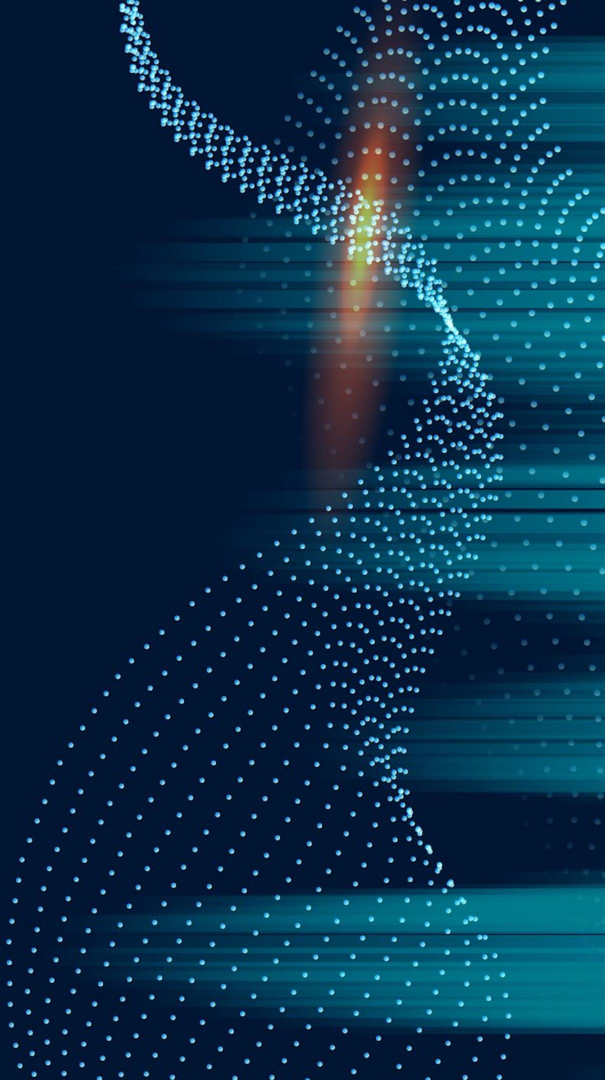
To generate sentences we use the concatenation operation on two elements of the alphabet

Let u, $v \in \mathbf{\Sigma^*}$, the concatenation uv is defined:

I. **Basis**: if length(v) = 0, then $v = \boldsymbol{\lambda}$ and $uv = u$

II. **Recursive step**: Let v be a string with length(v) = n > 0. Then $v = wa$, for some string w with length n-1 and $a \in \mathbf{\Sigma}$, and $uv = (uw)a$

III. **Closure**: uv is a concatenation on $\mathbf{\Sigma^*}$ only if it can be obtained from the basis by a finite number of applications of the recursive step



Strings and Languages

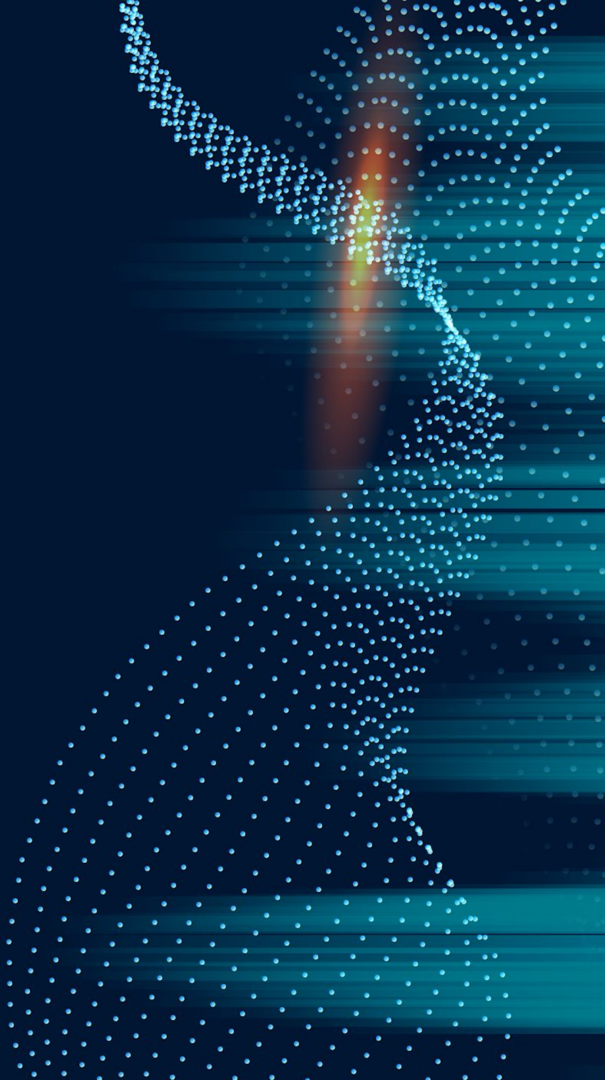
The concatenation operation is associative. The order is not important:

Let $u, v, w \in \mathbf{\Sigma^*}$. Then $(uv)w = u(vw)$

Example:

Let $u = ab$, $v = ca$, $w = bb$. Then

$uv = abca$
$vw = cabb$
$(uv)w = abcabb$
$u(vw) = abcabb$



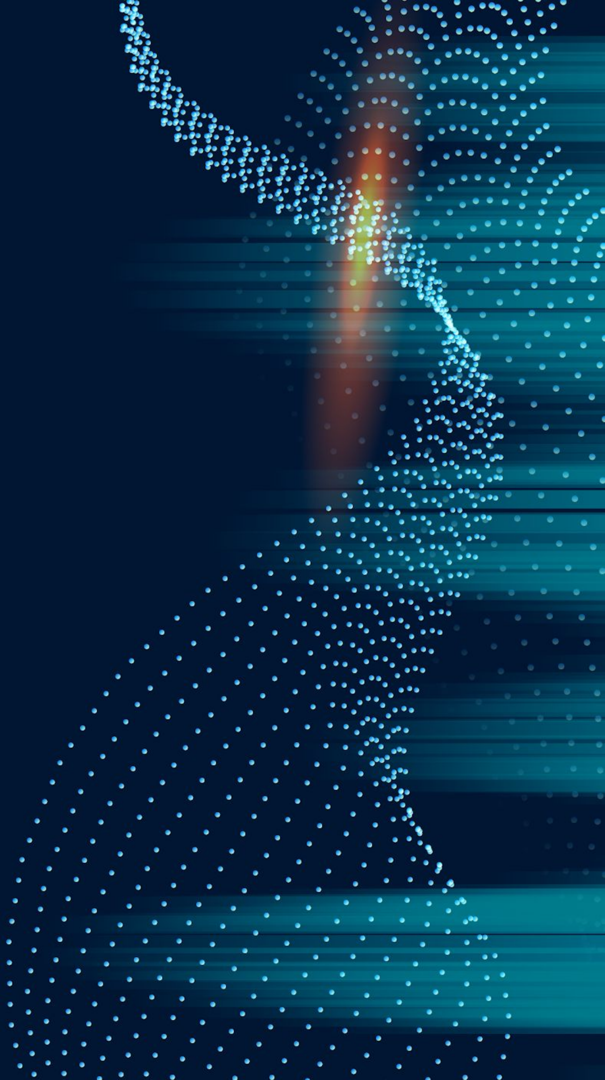
Strings and Languages

Demonstration by mathematical induction, using the recursive definition of concatenation and the length property over w:

Basis: $\text{length}(w) = 0$, then $(uv)w = uv$ and $u(vw) = uv$

Inductive Hypothesis: Assume that $(uv)w = u(vw)$ for all strings w of length n

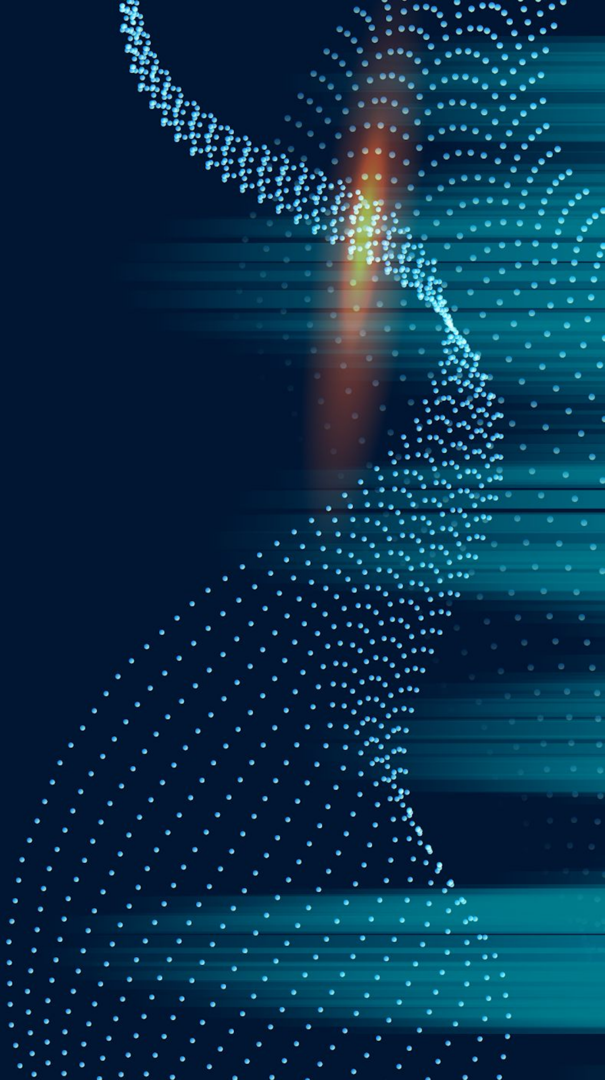
Inductive Step: Demonstrate that the same holds for strings of length n+1



Strings and Languages

Inductive Step: If w has length n+1, then $w = xa$ where x has length n and $a \in \Sigma$

$$
\begin{aligned}
(uv)w &= (uv)(xa) && \text{// Substitute w for xa} \\
&= ((uv)x)a && \text{// Definition of concatenation} \\
&= (u(vx))a && \text{// Inductive hypothesis} \\
&= u((vx)a) && \text{// Definition of concatenation} \\
&= u(v(xa)) && \text{// Definition of concatenation} \\
&= u(vw) && \text{// Substitute xa for w}
\end{aligned}
$$



Strings and Languages

We can also define the reversal of a string operation:

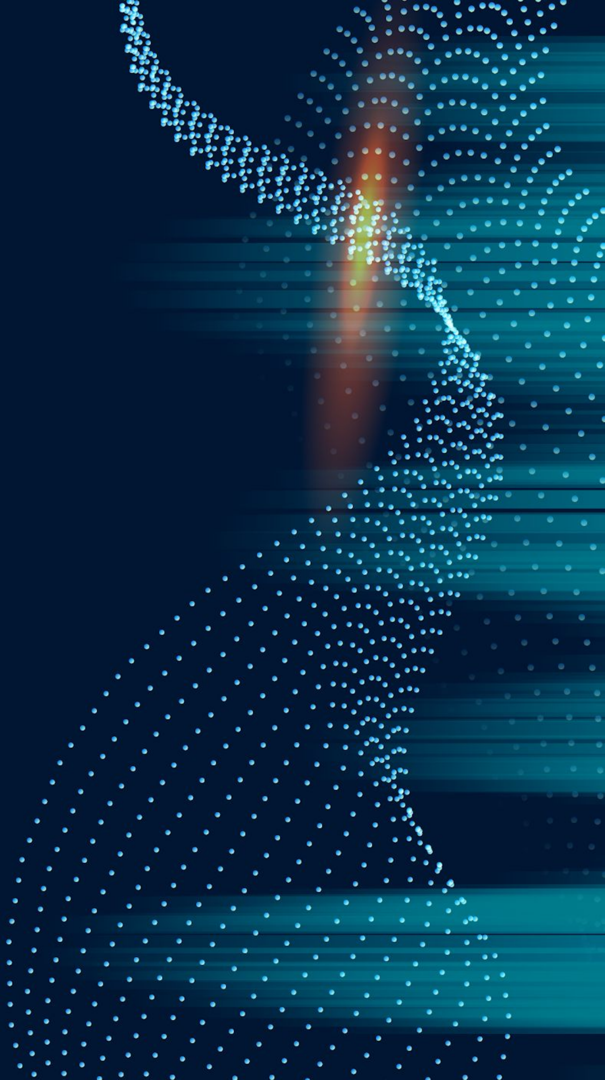
Let $u \in \mathbf{\Sigma^*}$, the reversal u^R is defined:

I. **Basis**: if length(u) = 0, then $u = \boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^R = \boldsymbol{\lambda}$

II. **Recursive step**: if length(u) = n > 0. Then $u = wa$, for some string w with length n-1 and $a \in \mathbf{\Sigma}$, and $u^R = aw^R$

We can also establish that:

Let $u, v \in \mathbf{\Sigma^*}$. Then $(uv)^R = v^R u^R$



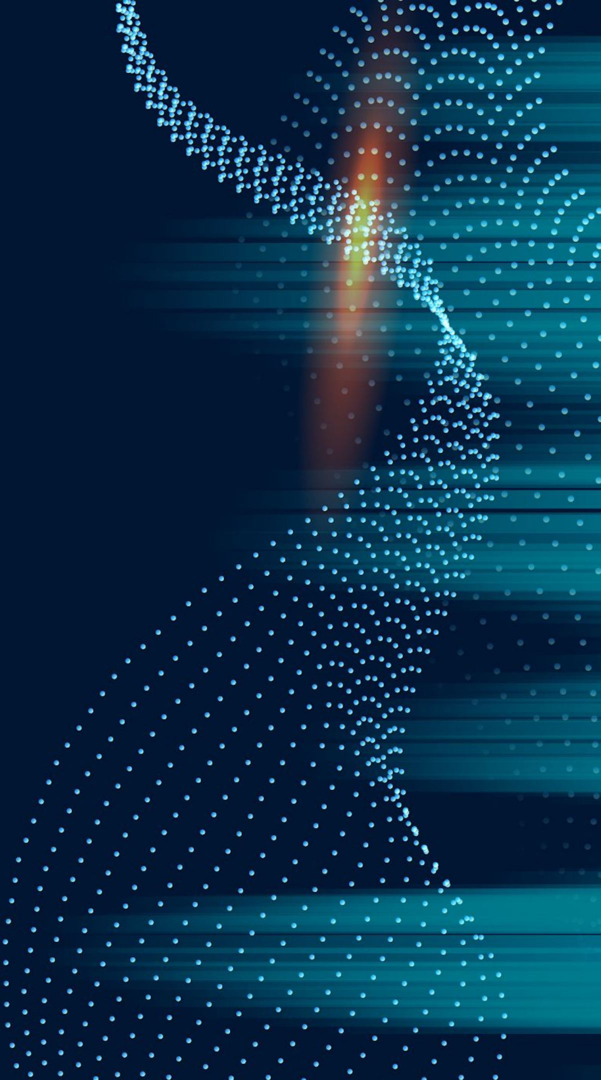
Strings and Languages

Demonstration by induction on the length of string *v*

Basis: length(*v*) = 0, then $(uv)^R = u^R$
Or $(uv)^R = \boldsymbol{\lambda}^R u^R = u^R$

Inductive Hypothesis: Assume that $(uv)^R = v^R u^R$ for all strings *v* of length *n*

Inductive Step: Demonstrate that the same holds for strings of length n+1



Strings and Languages

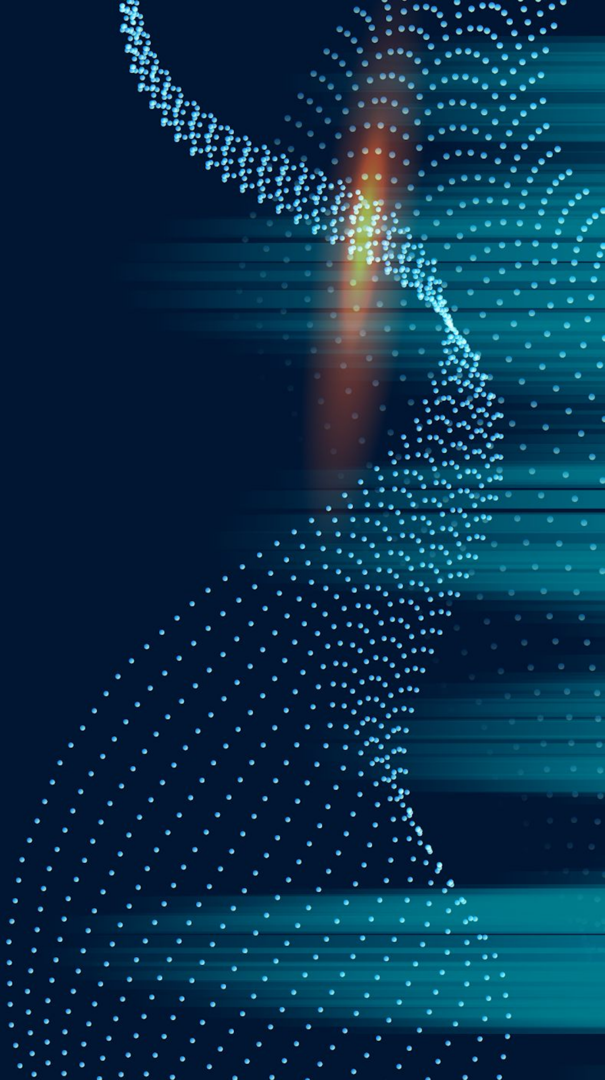
Inductive Step: If v has length n+1, then $v = xa$ where x has length n and $a \in \Sigma$

$(uv)^R = (u\ (x\ a))^R$	
$= ((u\ x)\ a)^R$	// Associativity of concatenation
$= a\ (u\ x)^R$	// Definition of reversal
$= a\ (x^R\ u^R)$	// Inductive hypothesis
$= (a\ x^R)\ u^R$	// Associativity of concatenation
$= (x\ a)^R\ u^R$	// Definition of reversal
$= v^R\ u^R$	// Substitute xa for v

Finite Specification of Languages

Finite Specification of Languages

Languages normally follow syntactic restrictions for what is a valid string. This can be indicated explicitly, using recursion



Examples:

Language **L** of strings over {*a*, *b*} where strings begin with *a* and have even length

I. **Basis**: *aa* and *ab* ∈ **L**

II. **Recursive step**: if *u* ∈ **L** then *uaa*, *uab*, *uba*, *ubb* ∈ **L**

III. **Closure**: a string *u* ∈ **L** only if it can be obtained from the basis following a finite set of applications of the recursive step

Finite Specification of Languages

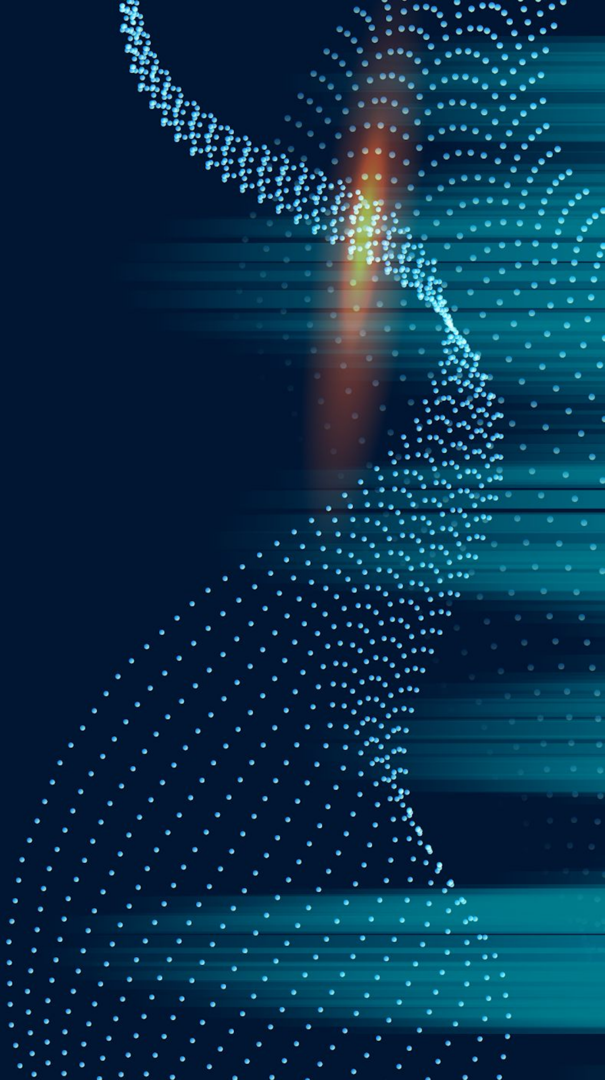
Concatenation of languages can also be defined as:

Let **X** and **Y** be languages, the concatenation **XY** is the language:

$$\mathbf{XY} = \{\, uv \mid u \in \mathbf{X} \text{ and } v \in \mathbf{Y} \,\}$$

The concatenation of **X** with itself n times can be represented as $\mathbf{X}^n$

$\mathbf{X}^0$ is defined as $\{\boldsymbol{\lambda}\}$



Examples:

For languages **X** = {*a*, *b*, *c*} and **Y** = {*abb*, *ba*}

XY = {*aabb*, *aba*, *babb*, *bba*, *cabb*, *cba*}

$\mathbf{X}^0$ = {**λ**}

$\mathbf{X}^1$ = **X** = {*a*, *b*, *c*}

$\mathbf{X}^2$ = {*aa*, *ab*, *ac*, *ba*, *bb*, *bc*, *ca*, *cb*, *cc*}

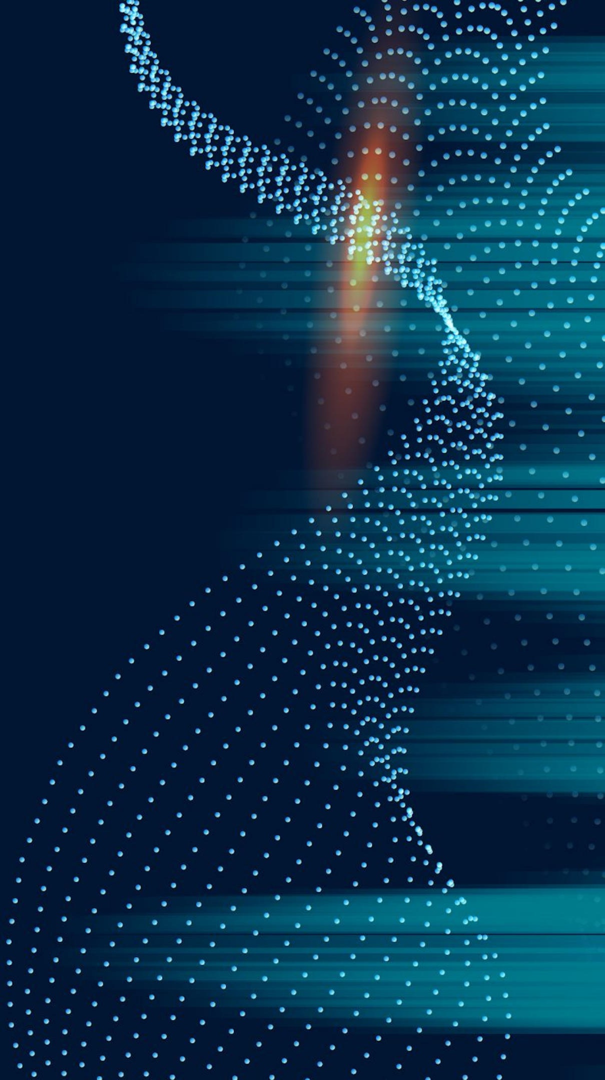
$\mathbf{X}^3$ = {*aaa*, *aab*, *aac*, *aba*, *abb*, *abc*, *aca*, *acb*, *acc*,
baa, *bab*, *bac*, *bba*, *bbb*, *bbc*, *bca*, *bcb*, *bcc*,
caa, *cab*, *cac*, *cba*, *cbb*, *cbc*, *cca*, *ccb*, *ccc*}

Finite Specification of Languages

For each *i*, the language $\mathbf{X}^{i}$ is the set of all strings of length *i* in $\mathbf{\Sigma}$

We can define sets of strings of different lengths as the Kleene star operation of set **X**: **X***
X+ is the set of all non null strings
X+ = **XX***

Using sets, concatenation and Kleene star we can describe languages with specific requirements



Examples:

- Language on {*a*, *b*} with strings that contain the substring *bb*:
- L = {*a*, *b*}*{*bb*}{*a*, *b*}*
- Language on {*a*, *b*} with strings that start with *aa* or end with *bb*:
- L = {*aa*}{*a*, *b*}* U {*a*, *b*}*{*bb*}
- Language on {*a*, *b*} with strings of an even length:
- L = {*aa*, *ab*, *ba*, *bb*}*
- Language on {*a*, *b*} with strings of an odd length:
- L = {*aa*, *ab*, *ba*, *bb*}*{*a*, *b*}
- Language on {*a*, *b*} with strings that start and end with *a*, and contain *b*:
- L = {*a*}{*a*, *b*}*{*b*}{*a*, *b*}*{*a*}

Regular Sets and Expressions

Regular Sets and Expressions

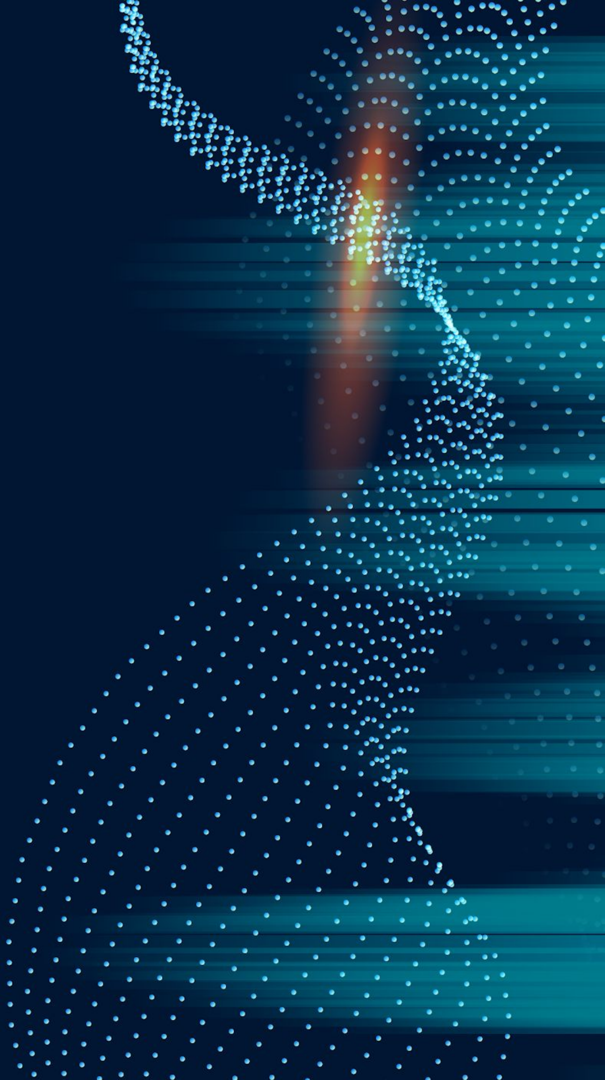
Recursive definition of regular sets:

Let **Σ** be an alphabet. The regular sets over **Σ** are defined as

I. **Basis**: ∅, {**λ**} and {*a*} for every *a* ∈ **Σ** are regular sets over **Σ**

II. **Recursive step**: Let **X** and **Y** be regular sets over **Σ**, then: **X** ∪ **Y**, **XY** and **X*** are regular sets over **Σ**

III. **Closure**: **X** is a regular set over **Σ** only if it can be obtained from the basis using a finite number of applications of the recursive step



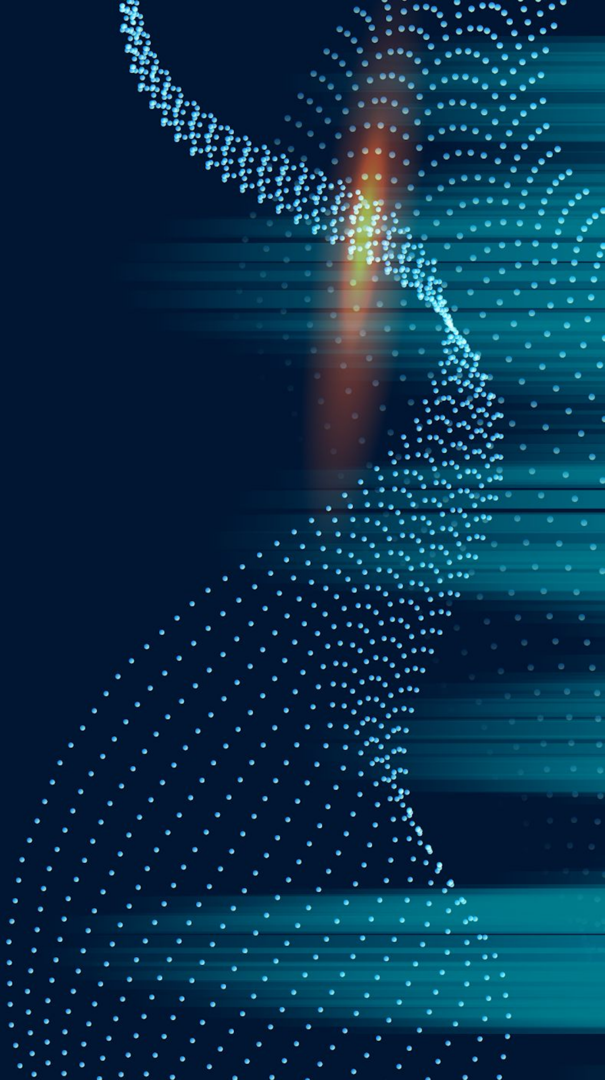
Regular Sets and Expressions

A regular language can be described by a regular set

These must be constructed from the empty set, a set with the null string or a set with a single element from **Σ**, using union, concatenation or Kleene star

The previous examples of languages are all regular

A regular expression is a form of abbreviation for a regular set, omitting the set brackets



Regular Sets and Expressions

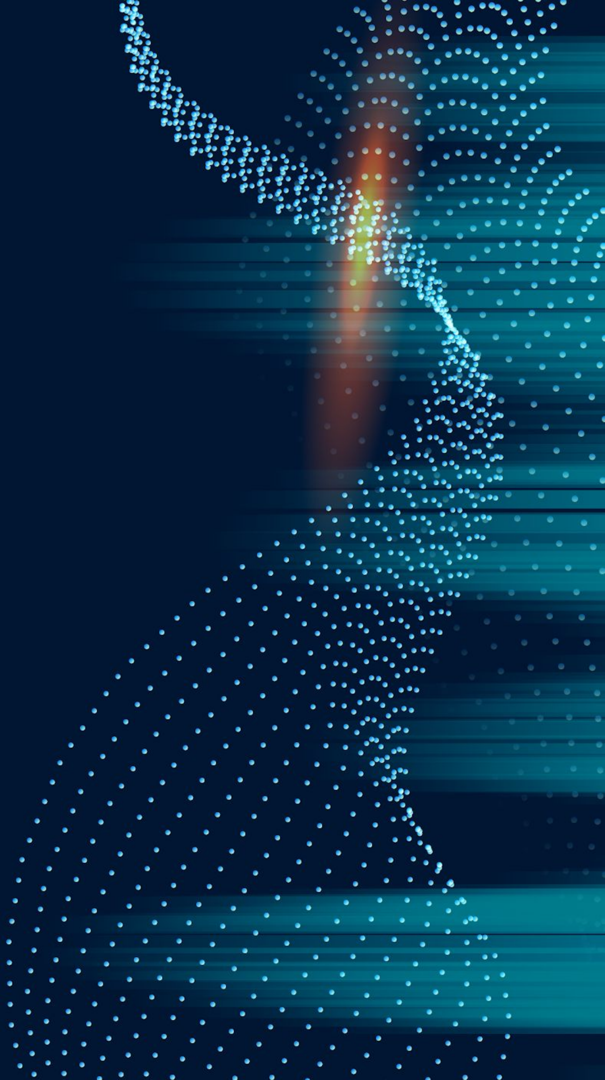
Recursive definition of regular expressions:

Let $\boldsymbol{\Sigma}$ be an alphabet. The regular expressions over $\boldsymbol{\Sigma}$ are defined as

I. **Basis**: $\varnothing$, λ and a for every $a \in \boldsymbol{\Sigma}$ are regular expressions over $\boldsymbol{\Sigma}$

II. **Recursive step**: Let u and v be regular expressions over $\boldsymbol{\Sigma}$, then: $(u \cup v)$, (uv) and (u^*) are regular expressions over $\boldsymbol{\Sigma}$

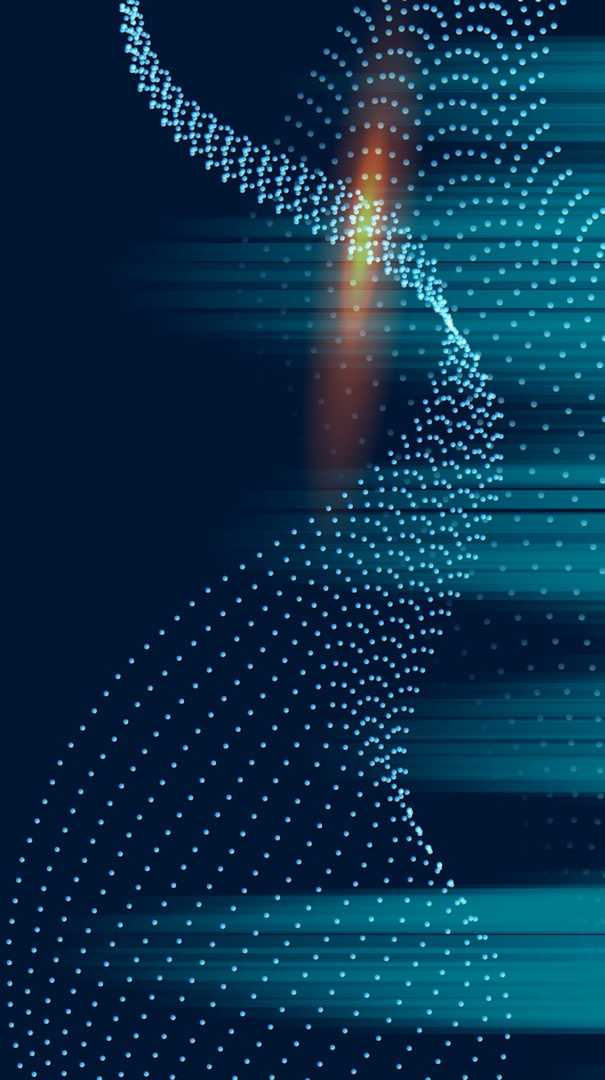
III. **Closure**: u is a regular expression over $\boldsymbol{\Sigma}$ only if it can be obtained from the basis using a finite number of applications of the recursive step



Regular Sets and Expressions

Further simplifications to regular expressions:

- Omit parentheses when only one operation is used
- Use precedence: Kleene > concatenation > union
- $u+$ represents uu^*
- u^2 represents uu



Examples:

Represent the following languages as regular expressions

- Language on {*a*, *b*} with strings that contain the substring *bb*:
  L = {*a*, *b*}*{*bb*}{*a*, *b*}*
- *(a U b)*bb(a U b)**
- Language on {*a*, *b*} with strings that start with *aa* or end with *bb*:
  L = {*aa*}{*a*, *b*}* U {*a*, *b*}*{*bb*}
- *aa(a U b)* U (a U b)*bb*
- Language on {*a*, *b*} with strings of an even length:
  L = {*aa*, *ab*, *ba*, *bb*}*
- *(aa U ab U ba U bb)**

Examples:

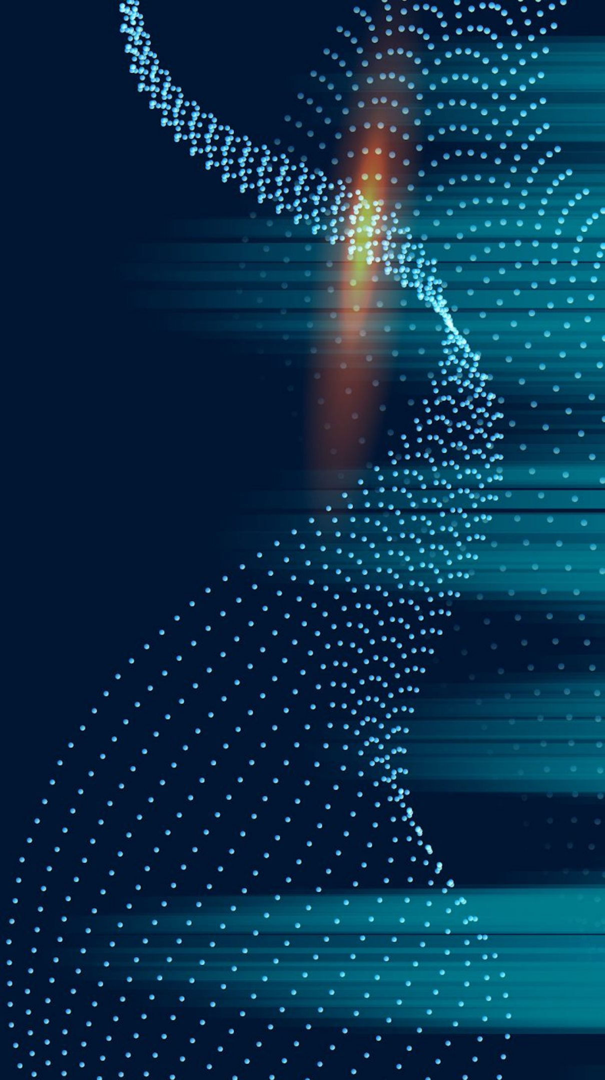
Represent the following languages as regular expressions

- Language on $\{a, b\}$ with strings of an odd length:
  $L = \{aa, ab, ba, bb\}^*\{a, b\}$
- $(aa \cup ab \cup ba \cup bb)^* (a \cup b)$
- Language on $\{a, b\}$ with strings that start and end with a, and contain b:
  $L = \{a\}\{a, b\}^*\{b\}\{a, b\}^*\{a\}$
- $a\,(a \cup b)^*\, b\,(a \cup b)^*\, a$

Regular Sets and Expressions

The operations on a regular expression represent:

- Concatenation: order
- Kleene star: repetition
- Union: selection

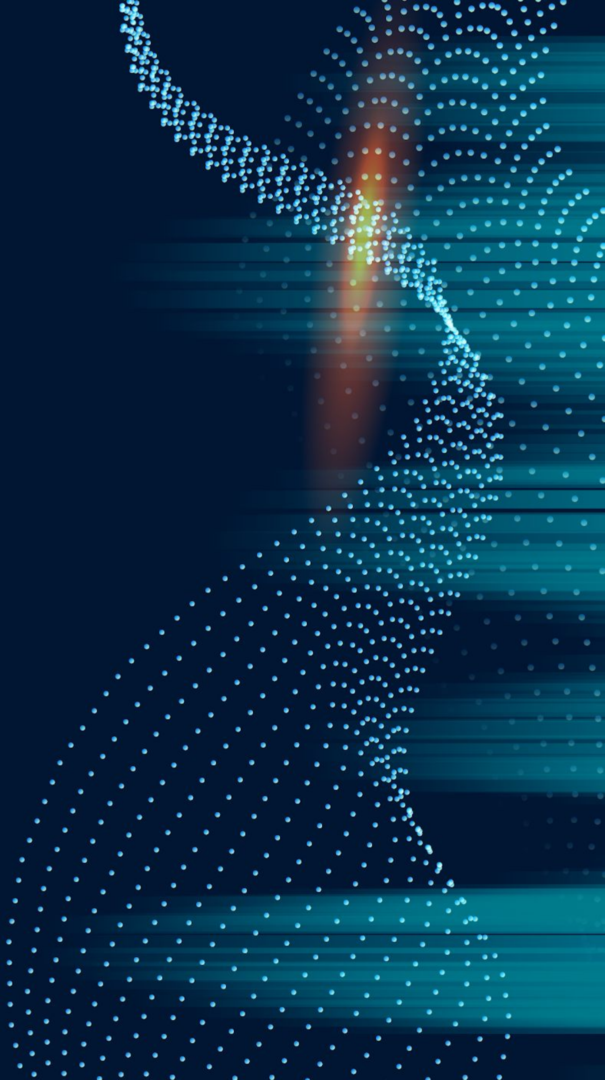


Regular Expressions and Text Searching

Regular Expressions and Text Searching

One of the most common uses for Regular Expressions is to search for string patterns

The syntax varies with implementations, but is generally based on sets, Kleene star and concatenation



Regular Expressions and Text Searching

Common symbols in search regular expressions:

Symbol	Example	Use
[]	[abcd] [a-z]	Selection from the objects in the set
^	[^abc]	Complement of the set
*	x*	Zero or more
+	[1-9]+	One or more
?	@?	One or none
|	a|b	Or (selection)

Regular Expressions and Text Searching

Common symbols in search regular expressions:

Symbol	Example	Use
^	^abc	The beginning of the string
$	xyz$	The end of the string
.	.*	Anything (greedy)
\d	\d\d	Digits. Equivalent to [0-9]
\s	a\sb	Whitespace
\w	\w+	Word characters (letters, numbers, _)

Examples:

- Language on {*a*, *b*} with strings that contain the substring *bb*:
- Regex = [*ab*]**bb*[*ab*]*
- Language on {*a*, *b*} with strings that start with *aa* or end with *bb*:
- Regex = (*aa*[*ab*]* | [*ab*]**bb*)
- Language on {*a*, *b*} with strings of an even length:
- Regex = (*aa* | *ab* | *ba* | *bb*)*
- Language on {*a*, *b*} with strings of an odd length:
- Regex = (*aa* | *ab* | *ba* | *bb*)*[*ab*]
- Language on {*a*, *b*} with strings that start and end with *a*, and contain *b*:
- Regex = *a*[*ab*]**b*[*ab*]**a*

Examples:

RE to search for a name (Name or initial, middle name or initial, Surname)

Name or initial: [A-Z][a-z]+|[A-Z][.]

Middle name or initial (optional): ([A-Z][a-z]+|[A-Z][.])?

Surname: [A-Z][a-z]+

Full name:

```
[A-Z][a-z]+|[A-Z][.]\s([A-Z][a-z]+|[A-Z][.]\s)?[A-Z][a-z]+
```

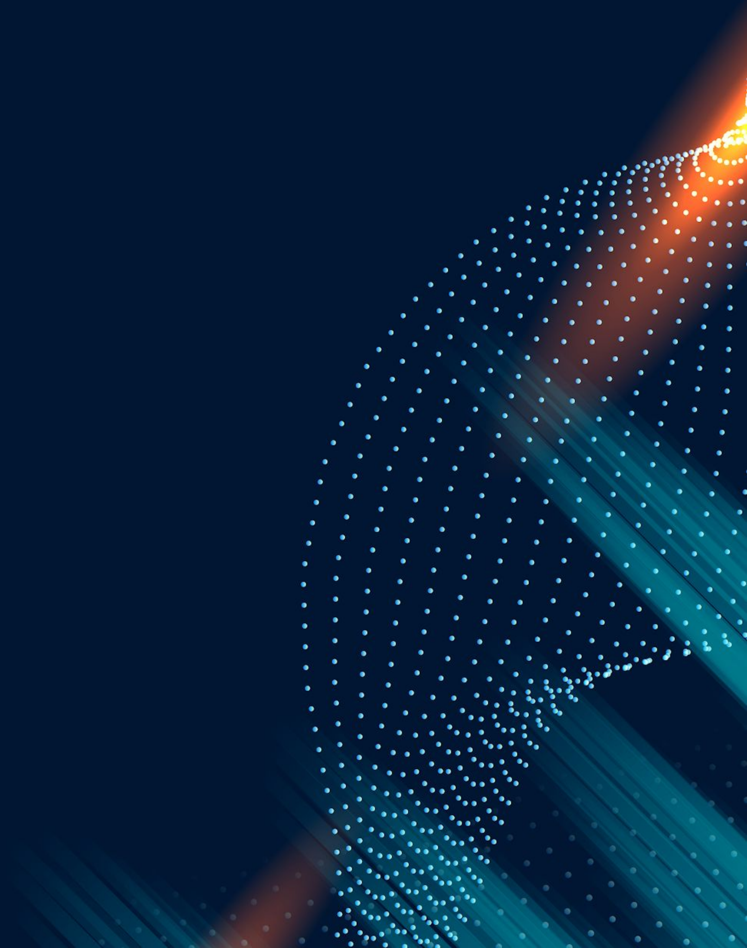
Examples:

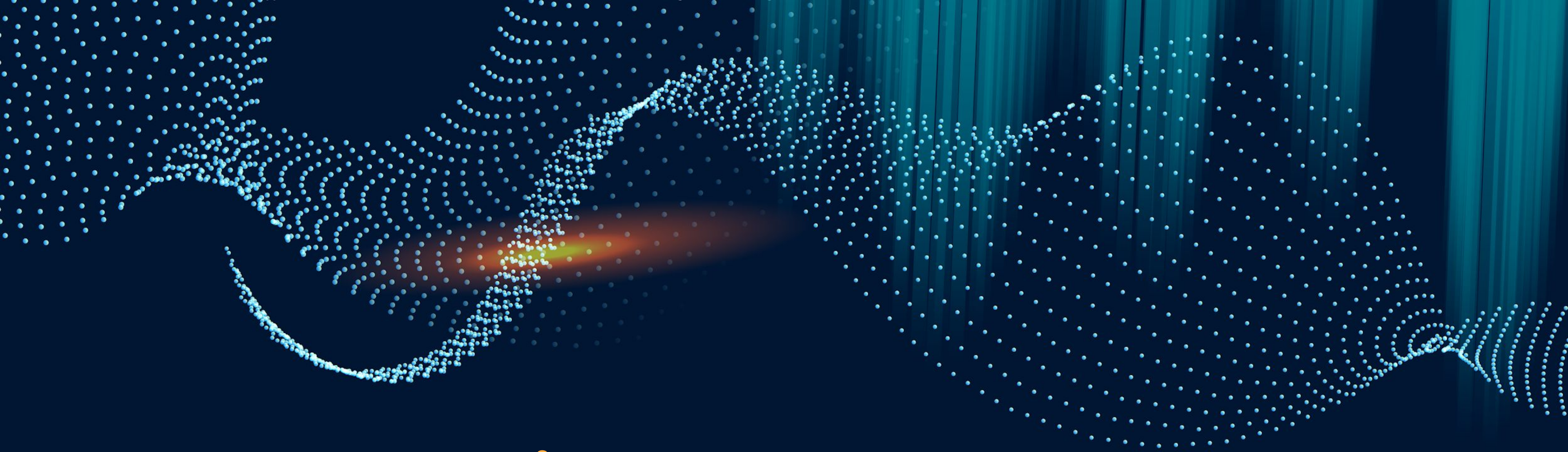
RE to search for a valid variable name:

```
[A-Za-z_][A-Za-z0-9_]*
```

RE to search for an email address:

```
\w+(\.\w+)*@\w+(\.\w+)+
```



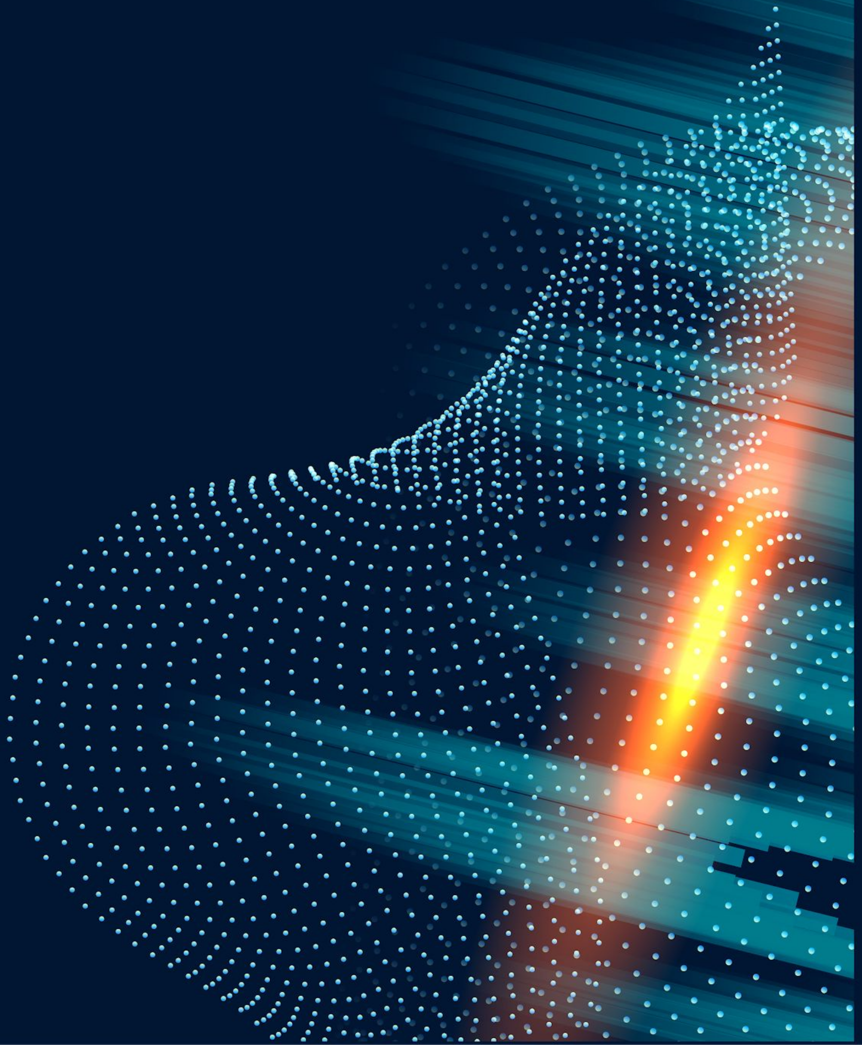


Simple but powerful notations to represent languages

Conclusion

THANKS!

Do you have any questions?
g.echeverria@tec.mx



References:

Nombre: Languages and machines: an introduction to the theory of computer science

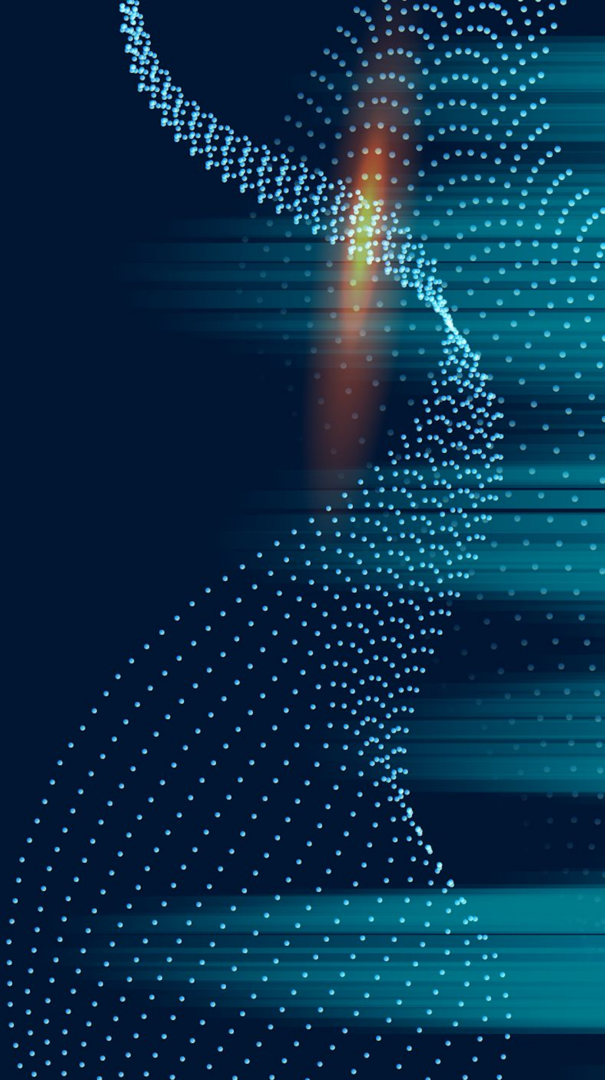
Autor: Thomas Sudkamp

Edición: 3rd

Año: 2016

Editorial: Addison-Wesley

ISBN: 9780321322210



References:

https://devopedia.org/chomsky-hierarchy

https://regex101.com/