# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

$\Lambda$ is a special string, called the empty string. It exists, regardless of the alphabet being used.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

$\Lambda$ is a special string, called the empty string. It exists, regardless of the alphabet being used.

**Concatenation:** The primary operator we use on strings is concatenation. This takes two strings as input and outputs a new string. Because we use it so often, we don't bother with a symbol: the concatenation of strings $x$ and $y$ is written $xy$.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

$\Lambda$ is a special string, called the empty string. It exists, regardless of the alphabet being used.

**Concatenation:** The primary operator we use on strings is concatenation. This takes two strings as input and outputs a new string. Because we use it so often, we don't bother with a symbol: the concatenation of strings $x$ and $y$ is written $xy$.
Example: $dog$ concatenated with $doogle$ is $dogdoogle$.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

$\Lambda$ is a special string, called the empty string. It exists, regardless of the alphabet being used.

**Concatenation:** The primary operator we use on strings is concatenation. This takes two strings as input and outputs a new string. Because we use it so often, we don't bother with a symbol: the concatenation of strings $x$ and $y$ is written $xy$.
Example: $dog$ concatenated with $doogle$ is $dogdoogle$.
We use $x^2 = xx$, $x^k = xx^{k-1}$, $x^0 = \Lambda$

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

$\Lambda$ is a special string, called the empty string. It exists, regardless of the alphabet being used.

**Concatenation:** The primary operator we use on strings is concatenation. This takes two strings as input and outputs a new string. Because we use it so often, we don't bother with a symbol: the concatenation of strings $x$ and $y$ is written $xy$.
Example: $dog$ concatenated with $doogle$ is $dogdoogle$.
We use $x^2 = xx$, $x^k = xx^{k-1}$, $x^0 = \Lambda$

**Length:** The length of a string is the number characters in the string.
Example: $|doogle| = 6$.

# Strings

**Alphabet:** An alphabet is a set of symbols. E.g.
$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, or $\Sigma_2 = \{0, 1\}$.

**String:** A string is a *finite* sequence of characters.
A string over some alphabet $\Sigma$ is a finite sequence of characters from that alphabet.
Example: $dog$ is a string over $\Sigma_1$. So is $doogle$. $001010$ is a string over $\Sigma_2$.

$\Lambda$ is a special string, called the empty string. It exists, regardless of the alphabet being used.

**Concatenation:** The primary operator we use on strings is concatenation. This takes two strings as input and outputs a new string. Because we use it so often, we don't bother with a symbol: the concatenation of strings $x$ and $y$ is written $xy$.
Example: $dog$ concatenated with $doogle$ is $dogdoogle$.
We use $x^2 = xx$, $x^k = xx^{k-1}$, $x^0 = \Lambda$

**Length:** The length of a string is the number characters in the string.
Example: $|doogle| = 6$. Example: $|\Lambda| = 0$

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.

## Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0 + 0, 0 + 1, 1 + 0, 1 + 1, 0 + 0 + 0, 0 + 0 + 1, \ldots\}$

## Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0 + 0, 0 + 1, 1 + 0, 1 + 1, 0 + 0 + 0, 0 + 0 + 1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

## Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0 + 0, 0 + 1, 1 + 0, 1 + 1, 0 + 0 + 0, 0 + 0 + 1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |

## Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0 + 0, 0 + 1, 1 + 0, 1 + 1, 0 + 0 + 0, 0 + 0 + 1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |

## Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |
| $\bar{L} = \Sigma^* \setminus L$ | $\bar{L}_3 = \{b, ab, ba, bb, aab, aba, abb, baa, bab, bba, bbb \ldots\}$ |

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0 + 0, 0 + 1, 1 + 0, 1 + 1, 0 + 0 + 0, 0 + 0 + 1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |
| $\bar{L} = \Sigma^* \setminus L$ | $\bar{L}_3 = \{b, ab, ba, bb, aab, aba, abb, baa, bab, bba, bbb \ldots\}$ |
| $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$ | $\{a, aa, aaa, aba, aaaa, aaba\}$ |

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |
| $\bar{L} = \Sigma^* \setminus L$ | $\bar{L}_3 = \{b, ab, ba, bb, aab, aba, abb, baa, bab, bba, bbb \ldots\}$ |
| $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$ | $\{a, aa, aaa, aba, aaaa, aaba\}$ |
| $L_2 L_1$ | $\{a, aa, aaa, aaaa, baa, baaa\}$ |

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |
| $\bar{L} = \Sigma^* \setminus L$ | $\bar{L}_3 = \{b, ab, ba, bb, aab, aba, abb, baa, bab, bba, bbb \ldots\}$ |
| $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$ | $\{a, aa, aaa, aba, aaaa, aaba\}$ |
| $L_2 L_1$ | $\{a, aa, aaa, aaaa, baa, baaa\}$ |

$L^0 = \{\Lambda\}$ for any $L$

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |
| $\bar{L} = \Sigma^* \setminus L$ | $\bar{L}_3 = \{b, ab, ba, bb, aab, aba, abb, baa, bab, bba, bbb \ldots\}$ |
| $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$ | $\{a, aa, aaa, aba, aaaa, aaba\}$ |
| $L_2 L_1$ | $\{a, aa, aaa, aaaa, baa, baaa\}$ |

$L^0 = \{\Lambda\}$ for any $L$
$L^2 = LL$ and $L^k = LL^{k-1}$

# Languages

**Language:** A language is a set of strings. It can be finite or infinite.
Example: {ab, bab, bbaab} is a language of size 3.
Example: $L = \{\Lambda, a, b, aa, ab, ba, bb\}$
Example: $\Sigma = \{0, 1, +\}$, $L = \{0+0, 0+1, 1+0, 1+1, 0+0+0, 0+0+1, \ldots\}$
While a language may have infinite size, each string in the language has finite size.

$\Sigma^*$ is the language containing all possible strings over the alphabet $\Sigma$.
Example: $\Sigma = \{a, b\}$, $\Sigma^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$
We sometimes write this as $\{a, b\}^*$

$\Sigma = \{a, b\}$, $L_1 = \{a, aa\}$, $L_2 = \{\Lambda, aa, ba\}$, $L_3 = \{\Lambda, a, aa, aaa, \ldots\}$
Language operators:

| | |
|---|---|
| $L_1 \cup L_2$ | $\{a, aa, \Lambda, ba\}$ |
| $L_1 \cap L_2$ | $\{aa\}$ |
| $L_2 \setminus L_1$ | $\{\Lambda, ba\}$ |
| $L_1 \setminus L_2$ | $\{a\}$ |
| $\bar{L} = \Sigma^* \setminus L$ | $\bar{L}_3 = \{b, ab, ba, bb, aab, aba, abb, baa, bab, bba, bbb \ldots\}$ |
| $L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$ | $\{a, aa, aaa, aba, aaaa, aaba\}$ |
| $L_2 L_1$ | $\{a, aa, aaa, aaaa, baa, baaa\}$ |

$L^0 = \{\Lambda\}$ for any $L$
$L^2 = LL$ and $L^k = LL^{k-1}$
$L^* = \bigcup_{i=0}^{\infty} L^i$

# Languages

Example: $L = \{a, bb\}$

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$

## Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$
$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$

## Languages

Example: $L = \{a, bb\}$

$L^0 = \{\Lambda\}$

$L^1 = \{a, bb\}$

$L^2 = \{aa, abb, bba, bbbb\}$

$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$

$L^* = \{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbb, bbaa, \ldots\}$

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$
$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$
$L^* = \{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbb, bbaa, \ldots\}$

extensional form: enumerate the strings.

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$
$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$
$L^* = \{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbb, bbaa, \ldots\}$

extensional form: enumerate the strings. Only finite sets, or possibly use "..."

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$
$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$
$L^* = \{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbb, bbaa, \ldots\}$

extensional form: enumerate the strings. Only finite sets, or possibly use "…"
intensional form: specify the *properties* of the strings.

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$
$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$
$L^* = \{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbb, bbaa, \ldots\}$

extensional form: enumerate the strings. Only finite sets, or possibly use "$\ldots$"
intensional form: specify the *properties* of the strings.
Informally: $L = \{x \mid x \text{ contains an equal number of } a\text{s and } b\text{s } \}$

# Languages

Example: $L = \{a, bb\}$
$L^0 = \{\Lambda\}$
$L^1 = \{a, bb\}$
$L^2 = \{aa, abb, bba, bbbb\}$
$L^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$
$L^* = \{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbb, bbaa, \ldots\}$

extensional form: enumerate the strings. Only finite sets, or possibly use "$\ldots$"
intensional form: specify the *properties* of the strings.
Informally: $L = \{x \mid x \text{ contains an equal number of } a\text{s and } b\text{s } \}$
Formally: $L = \{x \mid x \in \{a, b\}^* \wedge N_a(x) = N_b(x)\}$, where $N_a(x)$ denotes the number of $a$s in string $x$.

# Now What?

Until now we've talked about *how* to prove things.

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$
- How do we recognize the strings of a language?

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$
- How do we recognize the strings of a language?
  - given some $x$ and some description of a language $L$, how do we decide whether $x \in L$?

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$
- How do we recognize the strings of a language?
  - given some $x$ and some description of a language $L$, how do we decide whether $x \in L$?
  - An automaton for $L$ is a simple machine that allows us to decide, given some $x$, whether $x \in L$

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$
- How do we recognize the strings of a language?
  - given some $x$ and some description of a language $L$, how do we decide whether $x \in L$?
  - An automaton for $L$ is a simple machine that allows us to decide, given some $x$, whether $x \in L$

We will see that there are different *classes* of language: some are easier to generate / recognize than others.

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$
- How do we recognize the strings of a language?
  - given some $x$ and some description of a language $L$, how do we decide whether $x \in L$?
  - An automaton for $L$ is a simple machine that allows us to decide, given some $x$, whether $x \in L$

We will see that there are different *classes* of language: some are easier to generate / recognize than others.
$L_1 = \{x \mid x \in \{a\}^* \text{ and } x \text{ contains an even number of symbols}\}$
$L_2 = \{x \mid x \in \{a\}^* \text{ and } x \text{ contains a prime number of symbols}\}$

# Now What?

Until now we've talked about *how* to prove things.
Now we're going to start proving things about the nature of computation.

Central questions:

- How do we generate the strings of some language $L$?
  - Note that many languages have infinite size, so we need to take a finite representation of that language, and use it to construct all of the strings!
  - A *grammar* for $L$ gives a description of how to enumerate the strings of $L$
- How do we recognize the strings of a language?
  - given some $x$ and some description of a language $L$, how do we decide whether $x \in L$?
  - An automaton for $L$ is a simple machine that allows us to decide, given some $x$, whether $x \in L$

We will see that there are different *classes* of language: some are easier to generate / recognize than others.
$L_1 = \{x \mid x \in \{a\}^* \text{ and } x \text{ contains an even number of symbols}\}$
$L_2 = \{x \mid x \in \{a\}^* \text{ and } x \text{ contains a prime number of symbols}\}$

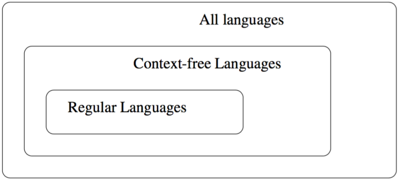


Figure 7.1: Containment of some language classes.

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

## Regular Languages

Let $\mathcal{R}$ be the set of all regular languages over symbol set $\Sigma$.

- $\emptyset \in \mathcal{R}$, $\{\Lambda\} \in \mathcal{R}$, $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{R}$

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

## Regular Languages

Let $\mathcal{R}$ be the set of all regular languages over symbol set $\Sigma$.

- $\emptyset \in \mathcal{R}$, $\{\Lambda\} \in \mathcal{R}$, $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{R}$
- If $L \in \mathcal{R}$, then $L^* \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 L_2 \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 \cup L_2 \in \mathcal{R}$

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

## Regular Languages

Let $\mathcal{R}$ be the set of all regular languages over symbol set $\Sigma$.

- $\emptyset \in \mathcal{R}$, $\{\Lambda\} \in \mathcal{R}$, $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{R}$
- If $L \in \mathcal{R}$, then $L^* \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 L_2 \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 \cup L_2 \in \mathcal{R}$
- There are no other regular languages over $\Sigma$.

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

## Regular Languages

Let $\mathcal{R}$ be the set of all regular languages over symbol set $\Sigma$.

- $\emptyset \in \mathcal{R}$, $\{\Lambda\} \in \mathcal{R}$, $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{R}$
- If $L \in \mathcal{R}$, then $L^* \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 L_2 \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 \cup L_2 \in \mathcal{R}$
- There are no other regular languages over $\Sigma$.

Example: $\Sigma = \{a, b, c\}$.
$L_1 = \{a\}, L_2 = \{b\}, L_3 = \{c\}$ are all regular

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

## Regular Languages

Let $\mathcal{R}$ be the set of all regular languages over symbol set $\Sigma$.

- $\emptyset \in \mathcal{R}$, $\{\Lambda\} \in \mathcal{R}$, $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{R}$
- If $L \in \mathcal{R}$, then $L^* \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 L_2 \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 \cup L_2 \in \mathcal{R}$
- There are no other regular languages over $\Sigma$.

Example: $\Sigma = \{a, b, c\}$.
$L_1 = \{a\}, L_2 = \{b\}, L_3 = \{c\}$ are all regular
$L = L_1 \cup L_2 L_3^*$:

# Regular Languages

The class of regular languages is the set of languages that can be built out of 3 simple operators: $\cup$, concatenation, and $*$

## Regular Languages

Let $\mathcal{R}$ be the set of all regular languages over symbol set $\Sigma$.

- $\emptyset \in \mathcal{R}$, $\{\Lambda\} \in \mathcal{R}$, $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{R}$
- If $L \in \mathcal{R}$, then $L^* \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 L_2 \in \mathcal{R}$
  If $L_1, L_2 \in \mathcal{R}$, then $L_1 \cup L_2 \in \mathcal{R}$
- There are no other regular languages over $\Sigma$.

Example: $\Sigma = \{a, b, c\}$.
$L_1 = \{a\}, L_2 = \{b\}, L_3 = \{c\}$ are all regular
$L = L_1 \cup L_2 L_3^*$: $L = \{a, b, bc, bcc, bccc, \ldots\}$

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | |
| $a(b + c)$ | |
| $(a + b)(a + c)(\Lambda + a)$ | |
| $a^*(b + cc)$ | |
| $a + bb^*$ | |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | |
| $a(b + c)$ | |
| $(a + b)(a + c)(\Lambda + a)$ | |
| $a^*(b + cc)$ | |
| $a + bb^*$ | |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | |
| $(a + b)(a + c)(\Lambda + a)$ | |
| $a^*(b + cc)$ | |
| $a + bb^*$ | |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | |
| $a^*(b + cc)$ | |
| $a + bb^*$ | |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE $(r)$ | Corresponding Language $(\mathcal{L}(r))$ |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | |
| $a + bb^*$ | |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | $\{x \mid x \in \{a, b\} \wedge \lvert x \rvert \text{ is even } \}$ |

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE $(r)$ | Corresponding Language $(\mathcal{L}(r))$ |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | $\{x \mid x \in \{a, b\} \wedge \lvert x \rvert \text{ is even } \}$ |

Some notational conveniences (that do not change the class of languages.):
Let $r$ be a RE. $r^0 = \Lambda$, and $\forall k \geq 0, r^{k+1} = rr^k$

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | $\{x \mid x \in \{a, b\} \wedge \lvert x\rvert \text{ is even } \}$ |

Some notational conveniences (that do not change the class of languages.):
Let $r$ be a RE. $r^0 = \Lambda$, and $\forall k \geq 0, r^{k+1} = rr^k$
$(a + b)^k$: all strings over $\{a, b\}$ of length exactly $k$

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | $\{x \mid x \in \{a, b\} \wedge \lvert x\rvert \text{ is even } \}$ |

Some notational conveniences (that do not change the class of languages.):
Let $r$ be a RE. $r^0 = \Lambda$, and $\forall k \geq 0, r^{k+1} = rr^k$
$(a + b)^k$: all strings over $\{a, b\}$ of length exactly $k$
$(a + b + \Lambda)^k$: all strings of length at most $k$.

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE $(r)$ | Corresponding Language $(\mathcal{L}(r))$ |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | $\{x \mid x \in \{a, b\} \wedge \lvert x\rvert \text{ is even }\}$ |

Some notational conveniences (that do not change the class of languages.):
Let $r$ be a RE. $r^0 = \Lambda$, and $\forall k \geq 0, r^{k+1} = rr^k$
$(a + b)^k$: all strings over $\{a, b\}$ of length exactly $k$
$(a + b + \Lambda)^k$: all strings of length at most $k$.

Positive closure: $r^+ = rr^*$

# Regular Expressions

We write $\cup$ with $+$, and we remove set notation.
Example: $(\{a\}\{b\}^* \cup \{c\}^*\{d\})^*\{e\}$ becomes $(ab^* + c^*d)^*e$.
Given some regular expression, $r$, we use $\mathcal{L}(r)$ to represent the language it denotes.

| RE ($r$) | Corresponding Language ($\mathcal{L}(r)$) |
|---|---|
| $a + bc$ | $\{a, bc\}$ |
| $a(b + c)$ | $\{ab, ac\}$ |
| $(a + b)(a + c)(\Lambda + a)$ | $\{aa, ac, ba, bc, aaa, aca, baa, bca\}$ |
| $a^*(b + cc)$ | $\{b, cc, ab, acc, aab, aacc, aaab, aaacc, \ldots\}$ |
| $a + bb^*$ | $\{a, b, bb, bbb, bbbb, \ldots\}$ |
| $(a + bb)^*$ | $\{\Lambda, a, bb, aa, abb, bba, bbbb, aaa, \ldots\}$ |
| $a^*b^*$ | $\{\Lambda, a, b, ab, aa, bb, aaa, aab, abb, bbb, \ldots\}$ |
| $((a + b)(a + b))^*$ | $\{x \mid x \in \{a, b\} \wedge \|x\| \text{ is even } \}$ |

Some notational conveniences (that do not change the class of languages.):
Let $r$ be a RE. $r^0 = \Lambda$, and $\forall k \geq 0, r^{k+1} = rr^k$
$(a + b)^k$: all strings over $\{a, b\}$ of length exactly $k$
$(a + b + \Lambda)^k$: all strings of length at most $k$.

Positive closure: $r^+ = rr^*$
(Alternatively, $r^* = \Lambda + r^+$)