עיבודشفות טביעה
שולי רייטנר
Implementing morphology and phonology
Implementing morphology and phonology

We begin with a simple problem: a lexicon of some natural language is given as a list of words. Suggest a data structure that will provide insertion and retrieval of data. As a first solution, we are looking for time efficiency rather than space efficiency.
Implementing morphology and phonology

We begin with a simple problem: a lexicon of some natural language is given as a list of words. Suggest a data structure that will provide insertion and retrieval of data. As a first solution, we are looking for time efficiency rather than space efficiency.

The solution: trie (word tree).
Implementing morphology and phonology

We begin with a simple problem: a lexicon of some natural language is given as a list of words. Suggest a data structure that will provide insertion and retrieval of data. As a first solution, we are looking for time efficiency rather than space efficiency.

The solution: trie (word tree).

Access time: $O(|w|)$. Space requirement: $O(\sum_w |w|)$. 
Implementing morphology and phonology

We begin with a simple problem: a lexicon of some natural language is given as a list of words. Suggest a data structure that will provide insertion and retrieval of data. As a first solution, we are looking for time efficiency rather than space efficiency.

The solution: trie (word tree).

Access time: $O(|w|)$. Space requirement: $O(\sum w |w|)$.

A trie can be augmented to store also a morphological dictionary specifying concatenative affixes, especially suffixes. In this case it is better to turn the tree into a graph.
Implementing morphology and phonology

We begin with a simple problem: a lexicon of some natural language is given as a list of words. Suggest a data structure that will provide insertion and retrieval of data. As a first solution, we are looking for time efficiency rather than space efficiency.

The solution: *trie* (word tree).

Access time: $O(|w|)$. Space requirement: $O(\sum_w |w|)$.

A trie can be augmented to store also a morphological dictionary specifying concatenative affixes, especially suffixes. In this case it is better to turn the tree into a graph.

The obtained model is that of *finite-state automata*. 
Finite-state technology

Finite-state automata are not only a good model for representing the lexicon, they are also perfectly adequate for representing dictionaries (lexicons + additional information), describing morphological processes that involve concatenation etc.

A natural extension of finite-state automata – finite-state transducers – is a perfect model for most processes known in morphology and phonology, including non-segmental ones.
Formal language theory – definitions
Formal language theory – definitions

Formal languages are defined with respect to a given *alphabet*, which is a finite set of symbols, each of which is called a *letter*.

A finite sequence of letters is called a *string*.
Formal language theory – definitions

Formal languages are defined with respect to a given *alphabet*, which is a finite set of symbols, each of which is called a *letter*.

A finite sequence of letters is called a *string*.

Example: Strings
Let $\Sigma = \{0, 1\}$ be an alphabet. Then all binary numbers are strings over $\Sigma$.

If $\Sigma = \{a, b, c, d, \ldots, y, z\}$ is an alphabet then *cat*, *incredulous* and *supercalifragilisticexpialidocious* are strings, as are *tac*, *qqq* and *kjshdflkwjehr*.
Formal language theory – definitions

The *length* of a string $w$, denoted $|w|$, is the number of letters in $w$. The unique string of length 0 is called the *empty string* and is denoted $\epsilon$. 
Formal language theory – definitions

The *length* of a string \( w \), denoted \(|w|\), is the number of letters in \( w \). The unique string of length 0 is called the *empty string* and is denoted \( \epsilon \).

If \( w_1 = \langle x_1, \ldots, x_n \rangle \) and \( w_2 = \langle y_1, \ldots, y_m \rangle \), the *concatenation* of \( w_1 \) and \( w_2 \), denoted \( w_1 \cdot w_2 \), is the string \( \langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle \). \(|w_1 \cdot w_2| = |w_1| + |w_2|\).

For every string \( w \), \( w \cdot \epsilon = \epsilon \cdot w = w \).
Formal language theory – definitions

Example: Concatenation

Let $\Sigma = \{a, b, c, d, \ldots, y, z\}$ be an alphabet. Then

$\text{master} \cdot \text{mind} = \text{mastermind}$, $\text{mind} \cdot \text{master} = \text{mindmaster}$

and $\text{master} \cdot \text{master} = \text{mastermaster}$. Similarly, $\text{learn} \cdot \text{s} = \text{learns}$, $\text{learn} \cdot \text{ed} = \text{learned}$ and $\text{learn} \cdot \text{ing} = \text{learning}$. 
Formal language theory – definitions

An exponent operator over strings is defined in the following way: for every string \( w \), \( w^0 = \epsilon \). Then, for \( n > 0 \), \( w^n = w^{n-1} \cdot w \).

Example: Exponent
If \( w = go \), then \( w^0 = \epsilon, w^1 = w = go, w^2 = w^1 \cdot w = w \cdot w = gogo, w^3 = gogogo \) and so on.
Formal language theory – definitions

The reversal of a string $w$ is denoted $w^R$ and is obtained by writing $w$ in the reverse order. Thus, if $w = \langle x_1, x_2, \ldots, x_n \rangle$, $w^R = \langle x_n, x_{n-1}, \ldots, x_1 \rangle$. 
Formal language theory – definitions

The reversal of a string $w$ is denoted $w^R$ and is obtained by writing $w$ in the reverse order. Thus, if $w = \langle x_1, x_2, \ldots, x_n \rangle$, $w^R = \langle x_n, x_{n-1}, \ldots, x_1 \rangle$.

Given a string $w$, a substring of $w$ is a sequence formed by taking contiguous symbols of $w$ in the order in which they occur in $w$. If $w = \langle x_1, \ldots, x_n \rangle$ then for any $i, j$ such that $1 \leq i \leq j \leq n$, $\langle x_i, \ldots x_j \rangle$ is a substring of $w$. 
Formal language theory – definitions

The reversal of a string $w$ is denoted $w^R$ and is obtained by writing $w$ in the reverse order. Thus, if $w = \langle x_1, x_2, \ldots, x_n \rangle$, $w^R = \langle x_n, x_{n-1}, \ldots, x_1 \rangle$.

Given a string $w$, a substring of $w$ is a sequence formed by taking contiguous symbols of $w$ in the order in which they occur in $w$. If $w = \langle x_1, \ldots, x_n \rangle$ then for any $i, j$ such that $1 \leq i \leq j \leq n$, $\langle x_i, \ldots x_j \rangle$ is a substring of $w$.

Two special cases of substrings are prefix and suffix: if $w = w_l \cdot w_c \cdot w_r$ then $w_l$ is a prefix of $w$ and $w_r$ is a suffix of $w$. 

Formal language theory – definitions

Example: Substrings
Let $\Sigma = \{a, b, c, d, \ldots, y, z\}$ be an alphabet and $w =$ indistinguishable a string over $\Sigma$. Then $\epsilon$, in, indis, indistinguish and indistinguishable are prefixes of $w$, while $\epsilon$, e, able, distinguishable and indistinguishable are suffixes of $w$. Substrings that are neither prefixes nor suffixes include distinguish, gui and is.
Formal language theory – definitions

Given an alphabet $\Sigma$, the set of all strings over $\Sigma$ is denoted by $\Sigma^*$. 
Formal language theory – definitions

Given an alphabet $\Sigma$, the set of all strings over $\Sigma$ is denoted by $\Sigma^*$.

A *formal language* over an alphabet $\Sigma$ is a subset of $\Sigma^*$. 
Formal language theory – definitions

Example: Languages
Let $\Sigma = \{a, b, c, \ldots, y, z\}$. Then $\Sigma^*$ is the set of all strings over the Latin alphabet. Any subset of this set is a language. In particular, the following are formal languages:
Formal language theory – definitions

• $\Sigma^*$;
Formal language theory – definitions

• $\Sigma^*$;
• the set of strings consisting of consonants only;
Formal language theory – definitions

• $\Sigma^*$;

• the set of strings consisting of consonants only;

• the set of strings consisting of vowels only;
Formal language theory – definitions

• $\sum^*$;
• the set of strings consisting of consonants only;
• the set of strings consisting of vowels only;
• the set of strings each of which contains at least one vowel and at least one consonant;
Formal language theory – definitions

- $\Sigma^*$;
- the set of strings consisting of consonants only;
- the set of strings consisting of vowels only;
- the set of strings each of which contains at least one vowel and at least one consonant;
- the set of palindromes;
Formal language theory – definitions

- $\Sigma^*$;
- the set of strings consisting of consonants only;
- the set of strings consisting of vowels only;
- the set of strings each of which contains at least one vowel and at least one consonant;
- the set of palindromes;
- the set of strings whose length is less than 17 letters;
Formal language theory – definitions

• $\Sigma^*$;
• the set of strings consisting of consonants only;
• the set of strings consisting of vowels only;
• the set of strings each of which contains at least one vowel and at least one consonant;
• the set of palindromes;
• the set of strings whose length is less than 17 letters;
• the set of single-letter strings;
Formal language theory – definitions

• $\Sigma^*$;
• the set of strings consisting of consonants only;
• the set of strings consisting of vowels only;
• the set of strings each of which contains at least one vowel and at least one consonant;
• the set of palindromes;
• the set of strings whose length is less than 17 letters;
• the set of single-letter strings;
• the set $\{i, you, he, she, it, we, they\}$;
Formal language theory – definitions

- $\Sigma^*$;
- the set of strings consisting of consonants only;
- the set of strings consisting of vowels only;
- the set of strings each of which contains at least one vowel and at least one consonant;
- the set of palindromes;
- the set of strings whose length is less than 17 letters;
- the set of single-letter strings;
- the set \{i, you, he, she, it, we, they\};
- the set of words occurring in Joyce’s Ulysses;
Formal language theory – definitions

- $\Sigma^*$;
- the set of strings consisting of consonants only;
- the set of strings consisting of vowels only;
- the set of strings each of which contains at least one vowel and at least one consonant;
- the set of palindromes;
- the set of strings whose length is less than 17 letters;
- the set of single-letter strings;
- the set $\{i, you, he, she, it, we, they\}$;
- the set of words occurring in Joyce’s Ulysses;
- the empty set;
Formal language theory – definitions

- $\Sigma^*$;
- the set of strings consisting of consonants only;
- the set of strings consisting of vowels only;
- the set of strings each of which contains at least one vowel and at least one consonant;
- the set of palindromes;
- the set of strings whose length is less than 17 letters;
- the set of single-letter strings;
- the set $\{i, you, he, she, it, we, they\}$;
- the set of words occurring in Joyce’s Ulysses;
- the empty set;

Note that the first five languages are infinite while the last five are finite.
Formal language theory – definitions

The string operations can be lifted to languages.
Formal language theory – definitions

The string operations can be lifted to languages.

If $L$ is a language then the reversal of $L$, denoted $L^R$, is the language $\{w \mid w^R \in L\}$. 
Formal language theory – definitions

The string operations can be lifted to languages.

If $L$ is a language then the reversal of $L$, denoted $L^R$, is the language $\{w \mid w^R \in L\}$.

If $L_1$ and $L_2$ are languages, then $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$.
Formal language theory – definitions

The string operations can be lifted to languages.

If $L$ is a language then the reversal of $L$, denoted $L^R$, is the language $\{w \mid w^R \in L\}$.

If $L_1$ and $L_2$ are languages, then
$L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$.

Example: Language operations
$L_1 = \{i, you, he, she, it, we, they\}$, $L_2 = \{smile, sleep\}$.

Then $L_1^R = \{i, uoy, eh, ehs, ti, ew, yeht\}$ and $L_1 \cdot L_2 = \{ismile, yousmile, hesmile, shesmile, itsmile, wesmile, theysmile, isleep, yousleep, hesleep, shesleep, itsleep, wesleep, theysleep\}$.
Formal language theory – definitions

If $L$ is a language then $L^0 = \{\epsilon\}$.
Then, for $i > 0$, $L^i = L \cdot L^{i-1}$. 
Formal language theory – definitions

If $L$ is a language then $L^0 = \{\epsilon\}$.

Then, for $i > 0$, $L^i = L \cdot L^{i-1}$.

Example: Language exponentiation
Let $L$ be the set of words $\{bau, haus, hof, frau\}$. Then $L^0 = \{\epsilon\}$, $L^1 = L$ and $L^2 = \{baubau, bauhaus, bauhof, baufrau, hausbau, haushaus, haushof, hausfrau, hofbau, hofhaus, hofhof, hoffrau, fraubau, frauhaus, frauhof, fraufrau\}$. 
Formal language theory – definitions

The Kleene closure of $L$ and is denoted $L^*$ and is defined as $\bigcup_{i=0}^{\infty} L^i$. 
Formal language theory – definitions

The Kleene closure of $L$ and is denoted $L^*$ and is defined as $\bigcup_{i=0}^{\infty} L^i$.
$L^+ = \bigcup_{i=1}^{\infty} L^i$. 
Formal language theory – definitions

The Kleene closure of $L$ and is denoted $L^*$ and is defined as $\bigcup_{i=0}^{\infty} L^i$.

$L^+ = \bigcup_{i=1}^{\infty} L^i$.

Example: Kleene closure
Let $L = \{\text{dog, cat}\}$. Observe that $L^0 = \{\epsilon\}$, $L^1 = \{\text{dog, cat}\}$, $L^2 = \{\text{catcat, catdog, dogcat, dogdog}\}$, etc.
Thus $L^*$ contains, among its infinite set of strings, the strings $\epsilon, \text{cat, dog, catcat, catdog, dogcat, dogdog, catcatcat, catdogcat, dogcatcat, dogdogcat}$, etc.
Formal language theory – definitions

The Kleene closure of $L$ and is denoted $L^*$ and is defined as

$$
\bigcup_{i=0}^{\infty} L^i.
$$

$L^+ = \bigcup_{i=1}^{\infty} L^i$.

Example: Kleene closure

Let $L = \{\text{dog, cat}\}$. Observe that $L^0 = \{\epsilon\}$, $L^1 = \{\text{dog, cat}\}$, $L^2 = \{\text{catcat, catdog, dogcat, dogdog}\}$, etc.

Thus $L^*$ contains, among its infinite set of strings, the strings $\epsilon, \text{cat, dog, catcat, catdog, dogcat, dogdog, catcatcat, catdogcat, dogcatcat, dogdogcat}$, etc.

The notation for $\Sigma^*$ should now become clear: it is simply a special case of $L^*$, where $L = \Sigma$. 
Regular expressions
Regular expressions

Regular expressions are a formalism for defining (formal) languages. Their “syntax” is formally defined and is relatively simple. Their “semantics” is sets of strings: the denotation of a regular expression is a set of strings in some formal language.
Regular expressions

Regular expressions are defined recursively as follows:
Regular expressions

Regular expressions are defined recursively as follows:

- $\emptyset$ is a regular expression
Regular expressions

Regular expressions are defined recursively as follows:

- $\emptyset$ is a regular expression
- $\epsilon$ is a regular expression
Regular expressions

Regular expressions are defined recursively as follows:

• $\emptyset$ is a regular expression

• $\epsilon$ is a regular expression

• if $a \in \Sigma$ is a letter then $a$ is a regular expression
Regular expressions

Regular expressions are defined recursively as follows:

• $\emptyset$ is a regular expression

• $\epsilon$ is a regular expression

• If $a \in \Sigma$ is a letter then $a$ is a regular expression

• If $r_1$ and $r_2$ are regular expressions then so are $(r_1 + r_2)$ and $(r_1 \cdot r_2)$
Regular expressions

Regular expressions are defined recursively as follows:

• \( \emptyset \) is a regular expression

• \( \epsilon \) is a regular expression

• if \( a \in \Sigma \) is a letter then \( a \) is a regular expression

• if \( r_1 \) and \( r_2 \) are regular expressions then so are \( (r_1 + r_2) \) and \( (r_1 \cdot r_2) \)

• if \( r \) is a regular expression then so is \( (r)^* \)
Regular expressions

Regular expressions are defined recursively as follows:

• $\emptyset$ is a regular expression

• $\epsilon$ is a regular expression

• if $a \in \Sigma$ is a letter then $a$ is a regular expression

• if $r_1$ and $r_2$ are regular expressions then so are $(r_1 + r_2)$ and $(r_1 \cdot r_2)$

• if $r$ is a regular expression then so is $(r)^*$

• nothing else is a regular expression over $\Sigma$. 
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet $\{a, b, c, \ldots, y, z\}$. Some regular expressions over this alphabet are:
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet \{a, b, c, \ldots, y, z\}. Some regular expressions over this alphabet are:

- $\emptyset$
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet \{a, b, c, \ldots, y, z\}. Some regular expressions over this alphabet are:

- $\emptyset$
- $a$
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet \{a, b, c, . . . , y, z\}. Some regular expressions over this alphabet are:

- $\emptyset$
- $a$
- $((c \cdot a) \cdot t)$
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet \{a, b, c, \ldots, y, z\}. Some regular expressions over this alphabet are:

- $\emptyset$
- $a$
- $((c \cdot a) \cdot t)$
- $(((m \cdot e) \cdot (o)^\ast) \cdot w)$
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet $\{a, b, c, \ldots, y, z\}$. Some regular expressions over this alphabet are:

- $\emptyset$
- $a$
- $((c \cdot a) \cdot t)$
- $(((m \cdot e) \cdot (o)^*) \cdot w)$
- $(a + (e + (i + (o + u))))$
Regular expressions

Example: Regular expressions
Let $\Sigma$ be the alphabet $\{a, b, c, \ldots, y, z\}$. Some regular expressions over this alphabet are:

- $\emptyset$
- $a$
- $((c \cdot a) \cdot t)$
- $(((m \cdot e) \cdot (o)^*) \cdot w)$
- $(a + (e + (i + (o + u))))$
- $((a + (e + (i + (o + u))))^*)$
Regular expressions

For every regular expression $r$ its denotation, $\lbrack r \rbrack$, is a set of strings defined as follows:
Regular expressions

For every regular expression $r$ its denotation, $[r]$, is a set of strings defined as follows:

- $[\emptyset] = \emptyset$
Regular expressions

For every regular expression $r$ its denotation, $\llbracket r \rrbracket$, is a set of strings defined as follows:

- $\llbracket \emptyset \rrbracket = \emptyset$
- $\llbracket \epsilon \rrbracket = \{ \epsilon \}$
Regular expressions

For every regular expression $r$ its denotation, $[r]$, is a set of strings defined as follows:

- $[\emptyset] = \emptyset$
- $[\epsilon] = \{\epsilon\}$
- If $a \in \Sigma$ is a letter then $[a] = \{a\}$
Regular expressions

For every regular expression $r$ its denotation, $[r]$, is a set of strings defined as follows:

- $[\emptyset] = \emptyset$
- $[\epsilon] = \{\epsilon\}$
- if $a \in \Sigma$ is a letter then $[a] = \{a\}$
- if $r_1$ and $r_2$ are regular expressions whose denotations are $[r_1]$ and $[r_2]$, respectively, then $[(r_1 + r_2)] = [r_1] \cup [r_2]$, $[(r_1 \cdot r_2)] = [r_1] \cdot [r_2]$ and $[(r_1)^*] = [r_1]^*$
Regular expressions

Example: Regular expressions and their denotations
Regular expressions

Example: Regular expressions and their denotations

∅
Regular expressions

Example: Regular expressions and their denotations

\emptyset \quad \emptyset
Regular expressions

Example: Regular expressions and their denotations

∅  ∅
a
Regular expressions

Example: Regular expressions and their denotations

∅  ∅

a  {a}
Regular expressions

Example: Regular expressions and their denotations

\[ \emptyset \quad \emptyset \]
\[ a \quad \{a\} \]
\[ ((c \cdot a) \cdot t) \]
Regular expressions

Example: Regular expressions and their denotations

\[
\emptyset \\
a \\
((c \cdot a) \cdot t)
\]
Regular expressions

Example: Regular expressions and their denotations

\[ \emptyset \]
\[ a \]
\[ ((c \cdot a) \cdot t) \]
\[ (((m \cdot e) \cdot (o)^{*}) \cdot w) \]

\[ \emptyset \]
\[ \{a\} \]
\[ \{c \cdot a \cdot t\} \]
Regular expressions

Example: Regular expressions and their denotations

\[
\emptyset \\
a \\
((c \cdot a) \cdot t) \\
(((m \cdot e) \cdot (o)^*) \cdot w) \\
\{mew, meow, meoow, meooow, meoooow, \ldots\}
\]
Regular expressions

Example: Regular expressions and their denotations

\[
\emptyset \\
a \\
(c \cdot a) \cdot t \\
((m \cdot e) \cdot (o)^*) \cdot w \\
\{ mew, meow, meoow, meooow, meooooow, \ldots \} \\
(a + (e + (i + (o + u)))) \\
\{ a, e, i, o, u \}
\]
Regular expressions

Example: Regular expressions and their denotations

∅

∅

a

{a}

((c \cdot a) \cdot t)

{c \cdot a \cdot t}

(((m \cdot e) \cdot (o)^*) \cdot w)

{mew, meow, meoow, meooow, meooow, \ldots}

(a + (e + (i + (o + u))))

{a, e, i, o, u}

((a + (e + (i + (o + u))))^*)
Regular expressions

Example: Regular expressions and their denotations

\[ \emptyset \]
\[ \{a\} \]
\[ (c \cdot a) \cdot t \]
\[ \{c \cdot a \cdot t\} \]
\[ (m \cdot e) \cdot (o)^* \cdot w \]
\[ \{mew, meow, meoow, meooow, meooooow, \ldots\} \]
\[ (a + (e + (i + (o + u)))) \]
\[ \{a, e, i, o, u\} \]
\[ ((a + (e + (i + (o + u))))^* \]

"the set containing all strings of 0 or more vowels"
Regular expressions

Example: Regular expressions
Given the alphabet of all English letters, \( \Sigma = \{a, b, c, \ldots, y, z\} \), the language \( \Sigma^* \) is denoted by the regular expression \( \Sigma^* \).
Regular expressions

Example: Regular expressions
Given the alphabet of all English letters, $\Sigma = \{a, b, c, \ldots, y, z\}$, the language $\Sigma^*$ is denoted by the regular expression $\Sigma^*$.

The set of all strings which contain a vowel is denoted by $\Sigma^* \cdot (a + e + i + o + u) \cdot \Sigma^*$. 
Regular expressions

Example: Regular expressions
Given the alphabet of all English letters, \( \Sigma = \{a, b, c, \ldots, y, z\} \), the language \( \Sigma^* \) is denoted by the regular expression \( \Sigma^* \).

The set of all strings which contain a vowel is denoted by \( \Sigma^* \cdot (a + e + i + o + u) \cdot \Sigma^* \).

The set of all strings that begin in “un” is denoted by \( (un) \Sigma^* \).
Regular expressions

Example: Regular expressions
Given the alphabet of all English letters, $\Sigma = \{a, b, c, \ldots, y, z\}$, the language $\Sigma^*$ is denoted by the regular expression $\Sigma^*$.

The set of all strings which contain a vowel is denoted by $\Sigma^* \cdot (a + e + i + o + u) \cdot \Sigma^*$.

The set of all strings that begin in “un” is denoted by $(un)\Sigma^*$.

The set of strings that end in either “tion” or “sion” is denoted by $\Sigma^* \cdot (s + t) \cdot (ion)$. 
Regular expressions

Example: Regular expressions
Given the alphabet of all English letters, \( \Sigma = \{a, b, c, \ldots, y, z\} \), the language \( \Sigma^* \) is denoted by the regular expression \( \Sigma^* \).

The set of all strings which contain a vowel is denoted by \( \Sigma^* \cdot (a + e + i + o + u) \cdot \Sigma^* \).

The set of all strings that begin in “un” is denoted by \((un)\Sigma^*\).

The set of strings that end in either “tion” or “sion” is denoted by \(\Sigma^* \cdot (s + t) \cdot (ion)\).

Note that all these languages are infinite.
Properties of regular languages
Properties of regular languages

Closure properties:

A class of languages $\mathcal{L}$ is said to be closed under some operation ‘$\bullet$’ if and only if whenever two languages $L_1, L_2$ are in the class ($L_1, L_2 \in \mathcal{L}$), also the result of performing the operation on the two languages is in this class: $L_1 \bullet L_2 \in \mathcal{L}$. 
Properties of regular languages

Regular languages are closed under:

- Union
- Intersection
- Complementation
- Difference
- Concatenation
- Kleene-star
- Substitution and homomorphism