Regular Expressions

CSCI 2210

2023-09-25 – 2023-10-02

A Theory of Languages

Over the last two weeks we have studied a class of languages that are decidable by a range of simple memory-free machine models, the **regular languages**.

Now we will describe a mathematical theory for these languages.

As we do this we will recall a familiar analog in a theory of arithmetic.

Many constructions will translate directly, but the theories will not be the same because numbers and languages have different properties from each other.

Expression Languages

A (single-sorted) **expression language** is a formal system for representing tree-structured syntax. It consists of:

variables: which we typically write as x, y, z, etc.,

symbols: each with a natural number **arity** specifying the number of required arguments,

parentheses: for grouping, so that we can write the tree structure linearly.

A syntactically valid construction made up of these is an **expression**.

We will disambiguate an "*expression language*" from a "*string language*" if necessary.

Arithmetic Expressions

You are probably familiar with the following expression language of **arithmetic** with symbols:

- $\underline{0}$ and $\underline{1}$, each of arity 0,
- $-\underline{+}-$ and $-\underline{\times}-$, each of arity 2.

For convenience we allow the abbreviation $\underline{n} := \underline{\overbrace{(1 + \cdots) + 1}^{n \text{ times}}}$.

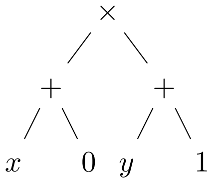
These *symbols* don't *mean* anything, they're pure syntax.

Sometimes we highlight expressions with underlining, bold font, etc. to distinguish e.g. the symbol $\underline{0}$ from the natural number 0.

Arithmetic Expression Trees

Expressions in this language include, $(x + 0) \times (y + 1)$.

This is a linear representation of the tree structure:



We can **substitute** one expression for a variable in another expression by "plugging in" a copy of its tree at each occurrence of the given variable.

Regular Expressions

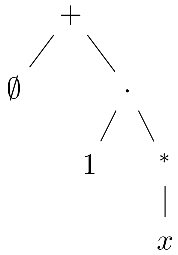
The expression language of **regular expressions** over an alphabet Σ has the symbols:

- for each $s \in \Sigma$, a symbol $\underline{s}$ of arity 0,
- $\underline{\emptyset}$ and $\underline{\varepsilon}$, each of arity 0,
- $-^{\underline{*}}$ of arity 1,
- $- \underline{+} -$ and $- \underline{\cdot} -$, each of arity 2.

Regular Expression Trees

If $\Sigma := \{0, 1\}$ then expressions in this language include, $\underline{\emptyset + (1 \cdot x^*)}$.

This is a linear representation of the tree structure:



It is customary to write $-\cdot-$ as *juxtaposition*, so $\underline{1 \cdot 0}$ becomes "$\underline{1\ 0}$".

Interpretation

An **interpretation** of an expression language into some mathematical structure is a function that assigns symbols of the language to operations of the structure in an arity-preserving way:

- symbols of arity 0 are mapped to *elements* of the structure,
- symbols of arity 1 are mapped to *endomorphisms* on the structure,
- symbols of arity 2 are mapped to *binary operations* on the structure,
- etc. (we don't need any higher arities today).

Variables that range over *expressions* are mapped to variables that range over *elements* of the structure.

It is customary to write interpretation functions as "$[\![-]\!]$" (with disambiguating annotations if needed).

Interpreting Arithmetic Expressions

An obvious interpretation of arithmetic expressions is into the structure of arithmetic, let's say, on the natural numbers $\mathbb{N}$:

- The nullary symbol $\underline{0}$ is interpreted as the *natural number* 0: $[\![\underline{0}]\!] = 0$.
- The nullary symbol $\underline{1}$ is interpreted as the *natural number* 1: $[\![\underline{1}]\!] = 1$.
- The binary symbol $\underline{+}$ is interpreted as the *addition operation*: $[\![\underline{+}]\!] = +$.
- The binary symbol $\underline{\times}$ is interpreted as the *multiplication operation*: $[\![\underline{\times}]\!] = \times$.

It may seem like we're just repeating ourselves, but remember that until we provide an interpretation the symbols of an expression language don't mean anything, they're just syntax.

With this interpretation:

$$[\![\underline{(x + 0) \times (y + 1)}]\!] = (x + 0) \times (y + 1)$$

Alternative Interpretation of Arithmetic Expressions

To stress this point, here is a *different* interpretation of arithmetic expressions, this time into the logical structure of the booleans $\mathbb{B}$:

- The nullary symbol $\underline{0}$ is interpreted as boolean *false*: $[\![\underline{0}]\!] = \bot$.
- The nullary symbol $\underline{1}$ is interpreted as the boolean *true*: $[\![\underline{1}]\!] = \top$.
- The binary symbol $\underline{+}$ is interpreted as the *disjunction* operation: $[\![\underline{+}]\!] = \vee$.
- The binary symbol $\underline{\times}$ is interpreted as the *conjunction* operation: $[\![\underline{\times}]\!] = \wedge$.

With this interpretation:

$$[\![\underline{(x + 0) \times (y + 1)}]\!] = (x \vee \bot) \wedge (y \vee \top)$$

Interpreting Regular Expressions

The intended **interpretation of regular expressions** over an alphabet Σ is into an algebraic structure on *string languages over* Σ (i.e. subsets of Σ^*).

- for $s \in \Sigma$, the nullary symbol $\underline{s}$ is interpreted as the *singleton language* of the length-one string s: $[\![\underline{s}]\!] = \mathrm{L}_{\{s\}}$,
- the nullary symbol $\underline{\emptyset}$ is interpreted as the *empty language*: $[\![\underline{\emptyset}]\!] = \mathrm{L}_{\emptyset}$,
- the nullary symbol $\underline{\varepsilon}$ is interpreted as the *singleton language* of the empty string: $[\![\underline{\varepsilon}]\!] = \mathrm{L}_{\{\varepsilon\}}$,
- the unary symbol $\underline{*}$ is interpreted as the *iteration operation* on languages: $[\![\underline{*}]\!] = {}^*$,
- the binary symbol $\underline{+}$ is interpreted as the *union operation* on languages: $[\![\underline{+}]\!] = \cup$,
- the binary symbol $\underline{\cdot}$ is interpreted as the *concatenation operation* on languages: $[\![\underline{\cdot}]\!] = +\!\!\!+$.

Interpreting Regular Expressions ctd.

So for our example regular expression $\underline{\emptyset + (1 \cdot x^*)}$ we have interpretation

$$[\![\underline{\emptyset + (1 \cdot x^*)}]\!] \;=\; \mathrm{L}_{\emptyset} \cup (\mathrm{L}_{\{1\}} +\!\!\!+\, x^*)$$

where the variable x now ranges over string languages.

This is like an endomorphism of strings languages.

If we make a substitution for the variable then we get a string language.

For example, if $x := \mathrm{L}_{\{0\}}$ then we get the language $\mathrm{L}_{\emptyset} \cup (\mathrm{L}_{\{1\}} +\!\!\!+\, \mathrm{L}_{\{0\}}{}^*)$, which contains strings beginning with a 1 and followed by any number of 0s.

Equational Theories

A **syntactic equation** for an expression language is a pair of its expressions, which we write suggestively as:

$$\underline{\mathrm{lexp}} \;=\; \underline{\mathrm{rexp}}$$

An **equational theory** for an expression language is a set of syntactic equations.

An interpretation $[\![-]\!]$ into a mathematical structure on the set A **satisfies** a syntactic equation $\underline{\mathrm{lexp}} \;=\; \underline{\mathrm{rexp}}$ involving variables $x, \cdots, z$ if the following proposition is true:

$$\forall\, x, \cdots, z \in \mathrm{A}\ .\ [\![\underline{\mathrm{lexp}}]\!] = [\![\underline{\mathrm{rexp}}]\!]$$

An interpretation satisfies an equational theory if it satisfies all its equations.

Equational Theory of Arithmetic

Here is an equational theory for our expression language of arithmentic:

$$\underline{0 + n} = \underline{n}, \qquad \underline{n + 0} = \underline{n},$$

$$\underline{1 \times n} = \underline{n}, \qquad \underline{n \times 1} = \underline{n},$$

$$\underline{0 \times n} = \underline{0}, \qquad \underline{n \times 0} = \underline{0},$$

$$\underline{(l + m) + n} = \underline{l + (m + n)}, \qquad \underline{m + n} = \underline{n + m},$$

$$\underline{(l \times m) \times n} = \underline{l \times (m \times n)}, \qquad \underline{m \times n} = \underline{n \times m},$$

$$\underline{l \times (m + n)} = \underline{(l \times m) + (l \times n)}, \qquad \underline{(l + m) \times n} = \underline{(l \times n) + (l \times n)},$$

Satisfying Arithmetic Equations

Consider the first syntactic equation,

$$\underline{0 + n} = \underline{n}$$

The natural numbers interpretation of our language of arithmetic satisfies this equation because,

$$\forall\, n \in \mathbb{N} \;.\; 0 + n = n.$$

The boolean interpretation of our language of arithmetic also satisfies this equation because,

$$\forall\, b \in \mathbb{B} \;.\; \bot \vee b = b.$$

Indeed, both interpretations satisfy all of the listed equations.

Equational Theory of Regular Expressions

Because of the properties of the operations of union, concatenation, and iteration for string language, many equations between regular expressions are satisfied in the string language interpretation.

For example:

Concatenation Semigroup

The following *associative law* for regular expressions:

$$\underline{(x \cdot y) \cdot z} \;=\; \underline{x \cdot (y \cdot z)}$$

is satisfied in the string language interpretation, because:

$$\forall\, \mathrm{L}_0, \mathrm{L}_1, \mathrm{L}_2 \subseteq \Sigma^* \;.\; (\mathrm{L}_0 \mathbin{+\!\!\!+} \mathrm{L}_1) \mathbin{+\!\!\!+} \mathrm{L}_2 = \mathrm{L}_0 \mathbin{+\!\!\!+} (\mathrm{L}_1 \mathbin{+\!\!\!+} \mathrm{L}_2)$$

which, in turn is true because concatenation of strings is associative:

$$\forall\, w_0, w_1, w_2 \in \Sigma^* \;.\; (w_0 \mathbin{+\!\!\!+} w_1) \mathbin{+\!\!\!+} w_2 = w_0 \mathbin{+\!\!\!+} (w_1 \mathbin{+\!\!\!+} w_2)$$

We use this associativity to write regular expressions involving $\cdot$ (a.k.a. juxtaposition) without explicit bracketing.

Concatenation Neutral and Absorbing Elements

The expression $\underline{\varepsilon}$ is a *neutral element* for $\underline{\cdot}$:

$$\underline{\varepsilon \cdot x} \;=\; \underline{x} \;=\; \underline{x \cdot \varepsilon}$$

because the singleton language of the empty string $\mathrm{L}_{\{\varepsilon\}}$ is neutral for language concatenation, in turn because ε is neutral for string concatenation:

$$\forall\, w \in \Sigma^* \;.\; \varepsilon +\!\!\!+ w = w = w +\!\!\!+ \varepsilon$$

The expression $\underline{\emptyset}$ is an *absorbing element* for $\underline{\cdot}$:

$$\underline{\emptyset \cdot x} \;=\; \underline{\emptyset} \;=\; \underline{x \cdot \emptyset}$$

because there are no strings in the empty language so:

$$\forall\, \mathrm{L} \subseteq \Sigma^* \;.\; \mathrm{L}_\emptyset +\!\!\!+ \mathrm{L} = \mathrm{L}_\emptyset = \mathrm{L} +\!\!\!+ \mathrm{L}_\emptyset$$

Alternation Commutative Monoid

The following *associative law* for regular expressions:

$$\underline{(x+y)+z} = \underline{x+(y+z)}$$

is satisfied in the string language interpretation, because:

$$\forall \mathrm{L}_0, \mathrm{L}_1, \mathrm{L}_2 \subseteq \Sigma^* \,.\, (\mathrm{L}_0 \cup \mathrm{L}_1) \cup \mathrm{L}_2 = \mathrm{L}_0 \cup (\mathrm{L}_1 \cup \mathrm{L}_2)$$

which, in turn is true because the union of sets is associative.
So we can unambiguously write multiple alternatives without brackets as well.

The union of sets is also commutative, so the *commutative law* for regular expressions is satisfied:

$$\underline{x+y} = \underline{y+x}$$

Finally, the empty set is a neutral element for set union, satisfying the equation:

$$\underline{\emptyset+x} = \underline{x} = \underline{x+\emptyset}$$

More Equations

You can read more about the algebraic theory of string languages in either Savage §4.3 or Hopcroft §3.4.

For example, concatenation **distributes** over alternation:

$$\underline{x \cdot (y + z)} = \underline{(x \cdot y) + (x \cdot z)} \qquad \text{and} \qquad \underline{(x + y) \cdot z} = \underline{(x \cdot z) + (y \cdot z)},$$

iteration is **idempotent**:

$$\underline{(x^*)^*} = \underline{x^*},$$

etc.

Abbreviations

For each word in a string language $w := w_0 w_1 \cdots w_n \in \Sigma^*$ we can define a regular expression:

$$\underline{w} \;:=\; \underline{w_0 \cdot w_1 \cdot \ldots \cdot w_n}$$

Then we have

$$[\![\underline{w}]\!] \;=\; \mathrm{L}_{\{w_0\}} +\!\!\!+\, \mathrm{L}_{\{w_1\}} +\!\!\!+ \cdots +\!\!\!+\, \mathrm{L}_{\{w_n\}} \;=\; \mathrm{L}_{\{w\}}$$

giving us singleton languages for any string.

For any subset of the alphabet $\mathrm{S} := \{s_0, s_1, \cdots, s_n\} \subseteq \Sigma$ we can define a regular expression:

$$\underline{\mathrm{S}} \;:=\; \underline{s_0 + s_1 + \cdots + s_n}$$

Then we have

$$[\![\underline{\mathrm{S}}]\!] \;=\; \mathrm{L}_{\{s_0\}} \cup \mathrm{L}_{\{s_1\}} \cup \cdots \cup \mathrm{L}_{\{s_n\}}$$

letting us define *character classes* such as *digits*, *letters*, *punctuation*, etc..
In programming it’s common to use “.” for “any character”.

Regular Operations Precedence

By convention the precedence order for regular expression operators is:

- * binds most tightly,
- followed by $\cdot$ (a.k.a. juxtaposition),
- followed by $+$.

So "$00 + 1^*$" means $(00) + (1^*)$.

Of course, you can always use explicit parentheses to to indicate a different order of operations; e.g. $0(0 + 1)^*$.

Regular Expressions for Languages

Write a regular expression for each of the following languages over the alphabet $\{0, 1\}$:

- words that start with 0 or end with 1
- words containing exactly two 0s
- words containing an even number of 0s

Languages of Regular Expressions

Give a brief English description of the language corresponding to each of the following regular expressions over the alphabet $\{0, 1\}$:

- $0 + (0 \,.^*\, 0)$
- $(\ldots)^*$
- $(0\,0 + 1\,1)\,.^*\,(0\,1 + 1\,0)$

Simplifying Regular Expressions

Use the fact that concatenation distributes over alternation to simplify the following regular expression: $0\,0\,1 + 0\,1\,1$

Regular Expressions Denote Regular Languages

Theorem

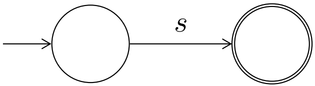
If r is a *closed regular expression* (i.e. one with no variables) over alphabet Σ then its string language interpretation $[\![r]\!]$ is a *regular language* over Σ.

The strategy is to show that:

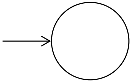
- for each symbol of arity 0 we can make an NFA to decide the language that is the symbol's interpretation,
- for each symbol of arity > 0 we can make an operation that transforms NFAs in the manner specified by the symbol's interpretation.

Nullary Symbols

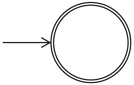
For nullary symbol $\underline{s} \in \Sigma$ the following NFA decides its interpretation, the language $\mathrm{L}_{\{s\}}$:



For nullary symbol $\underline{\emptyset}$ the following NFA decides its interpretation, the language $\mathrm{L}_{\emptyset}$:



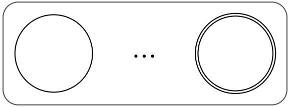
For nullary symbol $\underline{\varepsilon}$ the following NFA decides its interpretation, the language $\mathrm{L}_{\{\varepsilon\}}$:



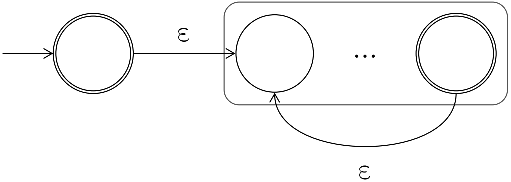
Iteration

The unary symbol $\underline{*}$ is interpreted as the iteration operation on languages *.

We saw previously how to transform an NFA that decides language L:



into an ε-NFA that decides language L^*:

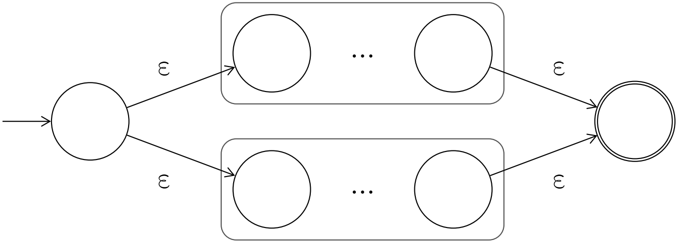


Union

The binary symbol $\underline{+}$ is interpreted as the union operation on languages $\cup$.

We saw previously how to transform NFAs that decide language L_0 and L_1:

into an ε-NFA that decides language $L_0 \cup L_1$:

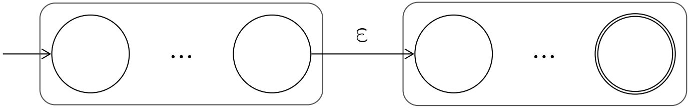


Concatenation

The binary symbol $\underline{\cdot}$ is interpreted as the concatenation operation on languages $+\!\!+$.

We saw previously how to transform NFAs that decide language L_0 and L_1:

into an ε-NFA that decides language $L_0 +\!\!+ L_1$:



Regular Languages are Described by Regular Expressions

Theorem

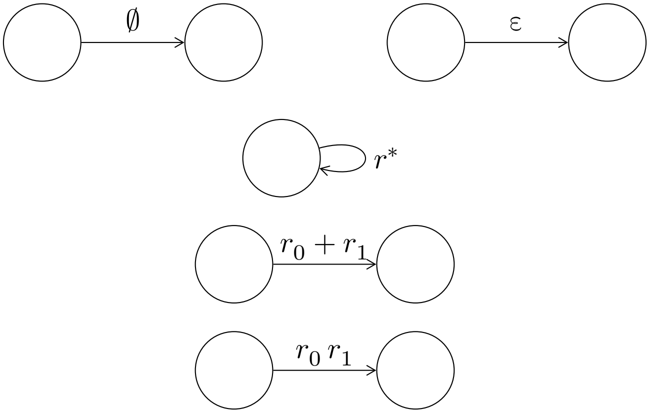
If L is a *regular language* over alphabet Σ then there is a closed *regular expression* over Σ whose string language interpretation is L.

The strategy is to:

- define a generalization of ε-NFAs whose transitions are labeled by regular expressions,
- reduce such a machine to a single regular expression by recursively removing vertices.

Generalized NFAs

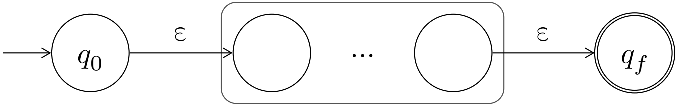
The state transition graph for a GNFA has edges labeled by regular expressions:



WLOG we can assume that there is at most one edge between each ordered pair of vertices and that every vertex has a loop. (Why?)

Recursive Reduction

We "sandwich" an NFA M with new start and accept states, q_0 and q_f:

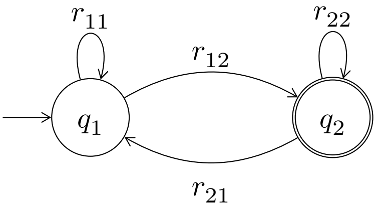


We choose any vertex $q_x \in \mathrm{Q_M}$ and *remove* it from the state set.

Then we *repair* the damage we caused by replacing each path we removed $[(q_i), r_0, (q_x), r_1, (q_x), r_2, (q_j)]$ with a new edge $r_0\, r_1{}^*\, r_2 : q_i \to q_j$.

We *repeat* this process until $\mathrm{Q} = \{q_0, q_f\}$, at which point the only remaining edge $r : q_0 \to q_f$ is labeled with a regular expression and $[\![r]\!] = \mathrm{L(M)}$.

Example: Recursive Reduction



Eliminating first q_1 then q_2 gives $r_{11}{}^* \, r_{12} \, (r_{21} \, r_{11}{}^* \, r_{12} + r_{22})^*$.

Eliminating first q_2 then q_1 gives $(r_{11} + r_{12} \, r_{22}{}^* \, r_{21})^* \, r_{12} \, r_{22}{}^*$.

How are these related?

Relating Regular Expressions

The following equations of regular expressions are true in the string language interpretation (see Savage §4.3):

alternative iteration: $(r+s)^* \;=\; (r^*\,s)^*\,r^* \;=\; s^*\,(r\,s^*)^*$

iteration rotation: $(r\,s)^*\,r \;=\; r\,(s\,r)^*$

With these we can calculate:

$$\begin{array}{cl} & r_{11}{}^*\,r_{12}\,(r_{21}\,r_{11}{}^*\,r_{12}+r_{22})^* \\ = & [a.i.] \\ & r_{11}{}^*\,r_{12}\,r_{22}{}^*\,(r_{21}\,r_{11}{}^*\,r_{12}\,r_{22}{}^*)^* \\ = & [i.r.] \\ & r_{11}{}^*\,r_{12}\,(r_{22}{}^*\,r_{21}\,r_{11}{}^*\,r_{12})^*\,r_{22}{}^* \end{array}$$

$$\begin{array}{cl} & (r_{11}+r_{12}\,r_{22}{}^*\,r_{21})^*\,r_{12}\,r_{22}{}^* \\ = & [a.i.] \\ & (r_{11}{}^*\,r_{12}\,r_{22}{}^*\,r_{21})^*\,r_{11}{}^*\,r_{12}\,r_{22}{}^* \\ = & [i.r.] \\ & r_{11}{}^*\,r_{12}\,(r_{22}{}^*\,r_{21}\,r_{11}{}^*\,r_{12})^*\,r_{22}{}^* \end{array}$$

Although the order we eliminate vertices can give us different *regular expressions*, they all represent the same *language*.