1 Finite Representations of Languages

Languages may be infinite sets of strings. We need a finite notation for them.

There are at least four ways to do this:

1. Language generators. The language can be represented as a mathematical sequence $w_1, w_2, w_3, \ldots$ such that the language is equal to the set $\{w_1, w_2, w_3, \ldots\}$. Given an integer i, the generator will produce the string w_i.
2. Language acceptors. The language can be represented as a mathematical predicate, a membership tester. Given a string, this will tell if the string is in the language.
3. Mathematical descriptions, like $\{a^n b^n : n \geq 0\}$.
4. Explicit listings, like $\{0, 1, 00, 01\}$.

- Explicit listings work only for finite languages.
- Math descriptions are very general, but it may be hard to know if a string is in the language.
- Language acceptors have a hard time answering some questions, such as whether the language is empty.
- Language generators have a hard time testing if a string is in the language.

There are uncountably many languages over a nonempty set Σ but only countably many representations in a finite set of symbols. Therefore most languages will never have a finite representation.

1.1 Regular Expressions

Regular expressions are one way to represent languages. They are analogous to arithmetic expressions for representing quantities. This notation will turn out to be useful for describing programming languages and also for text searching applications.

There are rules of inference for constructing regular expressions over an alphabet Σ.

1. If $a \in \Sigma$ then a itself is a regular expression over Σ.
2. $\emptyset$ is a regular expression over Σ.
3. If E and F are regular expressions over Σ then so is (EF).
4. If E and F are regular expressions over Σ then so is $(E \cup F)$.
5. If E is a regular expression over Σ then so is (E^*).
6. Parentheses can often be omitted.

Example: Suppose $\Sigma = \{0, 1\}$.

Then 0 is a regular expression over $\{0, 1\}$ by 1.
So (0^*) is a regular expression over $\{0, 1\}$ by 5.
Also, 1 is a regular expression over $\{0, 1\}$ by 1.
So $1(0^*)$ is a regular expression over $\{0, 1\}$ by 3.
Also (1^*) is a regular expression over $\{0, 1\}$ by 5.
So $0(1^*)$ is a regular expression over $\{0, 1\}$ by 3.
Thus $1(0^*) \cup 0(1^*)$ is a regular expression over $\{0, 1\}$ by 4.

This regular expression represents the language $(\{1\}\{0\}^*) \cup (\{0\}\{1\}^*)$.

This language contains strings like $\{1, 10, 100, 1000, \ldots, 0, 01, 011, 0111, \ldots\}$.
Note that $\{0, 1\}^*$ is not a regular expression over the alphabet $\{0, 1\}$.

1.2 Language Represented by a Regular Expression

If E is a regular expression then let $\mathcal{L}(E)$ be the language it represents.

We have the following rules:

$$\text{If } a \in \Sigma \text{ then } \mathcal{L}(a) = \{a\}.$$

$$\mathcal{L}(\emptyset) = \emptyset$$

$$\mathcal{L}(EF) = \mathcal{L}(E) \circ \mathcal{L}(F)$$

$$\mathcal{L}(E \cup F) = \mathcal{L}(E) \cup \mathcal{L}(F)$$

$$\mathcal{L}(E^*) = \mathcal{L}(E)^*$$

Note that $\mathcal{L}(E) \circ \mathcal{L}(F)$ is the concatenation of two languages, $\mathcal{L}(E) \cup \mathcal{L}(F)$ is the union of two languages, and $\mathcal{L}(E)^*$ is the Kleene star of a language.

Thus for example

$$\mathcal{L}(1(0^*) \cup 0(1^*)) =$$

$$\mathcal{L}(1(0^*)) \cup \mathcal{L}(0(1^*)) =$$

$$(\mathcal{L}(1) \circ \mathcal{L}(0^*)) \cup (\mathcal{L}(0) \circ \mathcal{L}(1^*)) =$$

$$(\{1\} \circ \{0\}^*) \cup (\{0\} \circ \{1\}^*).$$

1.3 Regular Languages

A language L is said to be *regular* if there is a regular expression E such that $L = \mathcal{L}(E)$, that is, if L can be represented by a regular expression.

Natural questions: Which languages can be represented by regular expressions? Is every language regular? Is $\{a^n b^n : n \geq 0\}$ regular? If L_1 and L_2 are regular, are $L_1 \cap L_2$, $L_1 - L_2$, $L_1 \cup L_2$, et cetera?

How can one generate a regular expression for a set S of strings? To do this, (a) split S into subsets that are easier to describe, (b) find a regular expression for each subset, then (c) take their union.

1.4 Equations Between Languages

Facts:

$$\{a, b\}^* \neq \{a\}^*\{b\}^*$$

$$\{a\}^*\{b\}^* \neq \{a\}^* \cup \{b\}^*$$

$$\mathcal{L}(\emptyset^*) = \{\epsilon\}$$

We write $E = F$ as regular expressions if $\mathcal{L}(E) = \mathcal{L}(F)$.

Facts:

$$ab\emptyset = \emptyset$$

$$ab(\emptyset^*) = ab$$

To simplify a regular expression E means to find a simpler regular expression F such that $E = F$.

In general how can one simplify a regular expression? To do this, (a) list some strings in the regular expression, (b) try to find a pattern in these strings, and (c) find a simpler regular expression for this pattern.

Note again that $\{0,1\}^*$ is not a regular expression over the alphabet $\{0,1\}$. Regular expressions do not contain any braces ($\{$, $\}$) or commas unless these symbols are in the alphabet.

1.5 Application

Consider a sequence of symbols $abacb$ as representing the trace of a computation where a, b, and c are events that occur in this order. Suppose one knows that in a given application, every a is followed by a b. This can be represented by the regular expression $(b \cup c)^*((ab)(b \cup c)^*)^*$. Call this $E1$. Suppose one also knows that every b is followed by a c two time steps later. This can be represented by the regular expression $E2$; this is a little bit complicated to write. One then wants to show that every a is followed by a c three time steps later. This can be represented by a regular expression $E3$ which is also complicated to write. Then if one shows that $\mathcal{L}(E1) \cap \mathcal{L}(E2) \subseteq \mathcal{L}(E3)$ this establishes the desired result. Thus regular expressions can be used to reason about program behaviors.

1.6 Problems

Give a regular expression for the set of even length binary strings.

Problem 1.8.1: What language is represented by the regular expression $(((a^*a)b) \cup b)$? Can you find a simpler expression for it?

Problem: Find a regular expression for the set of strings in $\{a,b\}^*$ that have exactly one a in them.

Problem: Find a regular expression for the set of strings in $\{a,b,c\}^*$ that have exactly one a or exactly one b in them.

Problem: Try to find a regular expression for the set of valid floating point numbers, things such as 0.326E+5. You can use D to represent the digits $\{0,1,2.3,4,5,6,7,8,9\}$.

1.7 Regular Expressions in Languages

Look at web links on regular expressions in various programming languages.

- Regular Expressions in Perl
- Unix Grep Utility
- Mastering Regular Expressions
- A Tao of Regular Expressions
- Wikipedia Article; Standards for Regular Expressions

Distinguish text searching from regular expressions
Searching for ca^* in $bbcaab$ will succeed but $bbcaab \notin \mathcal{L}(ca^*)$.
How to simulate ? with regular expressions
Protein Sequence Similarity – Explain BLAST

1.8 Finite Automata Introduction

- Fixed memory can be an advantage. Makes storage allocation and caching easier.
- A stack helps a little for memory allocation –can predict where accesses will be

Related Subjects

- Hidden Markov Model. Similar to finite automata but with probabilities attached to the transitions and also give outputs.
- Cellular Automata. Arrays of automata that interact with each other.
- Büchi Automata: Operate on infinite strings. Used for model checking. Accept if some accepting state is visited infinitely often.
- Timed automata