Computer Science

Dr. Yihsiang Liow (July 1, 2024)

Contents

Chapter 11

Languages

11.1 Informal introduction to languages debug: informal.tex

I will now introduce some terms used in the study of Languages in CS. Most of it is just giving terms in set theory some new names.

Let $\Sigma = \{a, b, c\}$. The following is a **string** of length 5: string

$$x = bccab$$

A **word** is just a string. Two strings are **equal** if elements of Σ appears in the string in the same order. You can **concatenate** strings. For instance if y is $aaab$, equal concatenate

$$x \cdot y = bccab \cdot aaab = bccabaaab$$

Instead of writing $x \cdot y$, I will also write xy. I also want to include an **empty string** written ϵ. The following holds for ϵ: empty string

$$\epsilon bccab = bcca\epsilon\epsilon b = bccab\epsilon\epsilon\epsilon$$

Given a string x we can define $|x|$, the **length** of x. For instance: length

$$|\epsilon| = 0, \quad |ab| = 2, \quad |a\epsilon\epsilon b| = 2$$

If L is a set of strings (not necessarily all the strings) is a **language** over Σ. Σ is said to be the **alphabet** of L. Elements of L are called **words**. If a word has length k, it's also called a k–**word**. language alphabet k–word

For instance take your favorite programming language, say C++. A string of the form:

```
if ( x == 0 ) x = 1;
```

is a string in the C++ language. You can already see words such as

```
if
```

and

```
==
```

and even

```
(
```

In this case the alphabet is pretty much all the visible character on your keyboard.

The language therefore specifies all possible "words" in C++. See how important the concept is now?

The above tells us that language theory (and automata theory ... later), although it seems abstract, can be very practical and useful.

In the chapter on sets, I mentioned that you might want to solve the following problem:

TelephoneNumberProblem(s) : Is the string s a valid telephone number?

One thing you can do is to construct a set to solve this problem. In this context, you have a language for the TelephoneNumberProblem:

$$\text{TelephoneNumberLanguage} = \{s \mid s \text{ is a valid telephone number}\}$$

Then answering the question TelephoneNumberProblem(1112223333) is just a membership check:

$$1112223333 \in \text{TelephoneNumberLanguage}$$

If you don't care about efficiency, of course this problem can be solved since you can store all telephone numbers in TelephoneNumberLanguage. This is possible because the TelephoneNumberLanguage is finite! So it's no big deal.

But in fact the above is way more important than for the sake of a telephone number validation program.

The above example converts a question (that requires a YES or NO answer) into a set membership problem. This abstraction is a very important step to the heart of **TCS** (**Theoretical Computer Science**).

TCS

Theoretical Computer Science

By the way a question that asks for a YES or NO is called a **decision problem**. We'll come back this concept during the later half of this course.

decision problem

The the above problem hides something more complex.

Consider the following problem. Suppose we consider the symbols Σ so be the set

$$\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{+}, \texttt{=}\}$$

Right now just think of `0`,`1`,`2`,...,`9` as symbols. Forget about the fact that they look like numbers. Also forget the fact that + looks like addition: it's just a symbol. Same for =: it's just a symbol. Now consider the following problem:

$$\text{AdditionProblem} : \text{Is } x + y = z?$$

where x, y, z are integers. I can convert this to a language:

$$\text{AdditionLanguage} = \{x\texttt{+}y\texttt{=}z \mid x, y, z \text{ are numeric strings and when viewed as integers, } x + y = z\}$$

For instance the word

$$\texttt{1+1=2}$$

is a word over Σ. And when we re-think this word (or string) as representing the *mathematical* fact

$$1 + 1 = 2$$

we see that the mathematical fact is true. Hence

$$\texttt{1+1=2} \in \text{AdditionLanguage}$$

On the other hand although

$$\texttt{2+3=7}$$

is a word over Σ, but it definitely does not represent a valid mathematical fact. Hence

$$\texttt{2+3=7} \notin \text{AdditionLanguage}$$

And of course the word

$$\texttt{2+52=} \notin \text{AdditionLanguage}$$

But there's fundamentally something very different between AdditionLanguage and TelephoneNumberLanguage. AdditionLanguage is infinite. You cannot possible store an infinite collection of string from the set AdditionLanguage in a database then query it!!!

So then how do we answer the membership problem

$$w \in \text{AdditionLanguage}$$

when AdditionLanguage?

It turns out that it can be done. But for sure the algorithm used has to be finite. This is achieved through the mathematical construction of computational models called **automata**. Automata are finite in nature in terms of the mathematical description of how they work. The description is provided by a finite directed graph where the edges are decorated in a certain way.

automata

Although automata were first discovered abstractly as mathematical objects, we now know that they can be implemented in software and hardware. In fact

the modern day computer is essentially the realization of Turing machines, which are a type of automatas.

You will see that there are many classes of automatas. For simple languages, we only need to define "simple" automatas. For complex languages, we will need "complex" automatas. We will study the following classes of automatas:

1. Finite state automatas
2. Pushdown automatas
3. Deterministic pushdown automatas (optional)
4. Turing machines

These computational models have given rise to the study of computational complexity which studies how to classify the complexity of a problem in terms of resource usage such as time and space. Probably the most important problem is TCS and in math is the so–called "P = NP" or "P vs NP" problem. This is one of the seven Millennium Problem. You can read about it at https://www.claymath.org/millennium-problems/p-vs-np-problem.

The classification of problems into complexity classes gives us a catalog of problems that we know either cannot be solved or can only be solved with different resources (polynomial runtime, exponential runtime, etc.) Automata theory provides methods to compare problems and help predict if a problem is easy or hard by place the problem into certain complexity classes. For instance a problem is in P is the problem can be solved in polynomial time. This is obviously very helpful: If you know your theory of automata well, you can tell your boss that a problem he wants you to solve either cannot be solved or is know to have an exponential runtime.

11.2 Formal introduction to languages (uh-oh)

debug: formal.tex

Through this section, Σ is a finite nonempty set. (What's the point of an empty set?? The only language you get out of it is the empty language or the language with only the empty string!)

empty string

Definition 11.2.1. ϵ is the **empty string**. The standard symbol for this empty string is ϵ. Obviously we assume that ϵ is not a symbol in Σ! (Some books use λ and some authors call it the null string). For convenience we write Σ_ϵ for $\Sigma \cup \{\epsilon\}$.

string
word

Definition 11.2.2. A **string** or **word** (defined) over Σ is an expression of the form

$$x_0 x_1 x_2 \dots x_{n-1}$$

where $x_i \in \Sigma_\epsilon$ (i.e., x_i is either ϵ or is in Σ.) The set of all strings over Σ is denote by Σ^*.

Definition 11.2.3. We also define

$$\Sigma^+ \stackrel{\text{def}}{=} \Sigma^* - \{\epsilon\}$$

11.3 The equals relation

debug: equals.tex

Most authors define $=$ using intuition: $x = y$ if after omitting ϵ from x and y, the alphabets in x and y occur in the same order. Here's a more formal definition. First we will define $=$ using recursion. We will then note that $=$ is an equivalence relation. Also, we will then show that every string is equivalence to either ϵ or one without ϵ. So formally, you can think of ϵ and strings without ϵ as a complete set of representatves for Σ^*.

Definition 11.3.1. Given a string x over Σ. We define the relation $=$ on Σ^* inductively as follows.

(a) $\epsilon = \epsilon$
(b) $a = a$ for all $a \in \Sigma$
(c) If $x = y$ where $x, y \in \Sigma^*$, then $x = \epsilon y$
(d) If $x = y$ where $x, y \in \Sigma^*$, then $x = y\epsilon$
(e) If $x = y$ where $x, y \in \Sigma^*$, then $\epsilon x = y$
(f) If $x = y$ where $x, y \in \Sigma^*$, then $x\epsilon = y$
(g) If $x = y$ where $x, y \in \Sigma^*$, then $ax = ay$ for any $a \in \Sigma$.
(h) If $x = y$ where $x, y \in \Sigma^*$, then $xa = ya$ for any $a \in \Sigma$.

We say x **equals** y if $x = y$.

equals

Exercise 11.3.1. Let $\Sigma = \{0, 1\}$. Verify that $\epsilon 011\epsilon 1 = 0\epsilon\epsilon 111\epsilon$ using the above definition of $=$. (Go to solution, page 10008) □

debug: exercises/equals0/question.tex

Exercise 11.3.2. Prove or disprove: Let $x, y \in \Sigma^*$. Then $xy = yx$. (Go to solution, page 10009) □

debug: exercises/equals1/question.tex

Proposition 11.3.1. *$=$ is an equivalence relation, i.e.,*

(a) Reflexive: $x = x$
(b) Symmetric: *If $x = y$, then $y = x$.*
(c) Transitive: *If $x = y$ and $y = z$, then $x = z$.*

□

Exercise 11.3.3. You know this is coming: Prove the above statement. (Go to solution, page 10010) □

debug: exercises/equals2/question.tex

Definition 11.3.2. A **language** L over Σ is just a subset of Σ^*. Σ is the **alphabet** of L. Elements of L are called **words** in L.

language
alphabet
words

Solutions

Solution to Exercise 11.3.1.

debug: exercises/equals0/answer.tex

$$
\begin{aligned}
1 &= 1 && \\
\therefore\ 1 &= 1\epsilon && \text{by } \_\_\_\_\_ \\
\therefore\ \epsilon 1 &= 1\epsilon && \text{by } \_\_\_\_\_ \\
\therefore\ 1\epsilon 1 &= 11\epsilon && \text{by } \_\_\_\_\_ \\
\therefore\ 11\epsilon 1 &= 111\epsilon && \\
\therefore\ 11\epsilon 1 &= \epsilon 111\epsilon && \\
\therefore\ 11\epsilon 1 &= \epsilon\epsilon 111\epsilon && \\
\therefore\ 011\epsilon 1 &= 0\epsilon\epsilon 111\epsilon && \\
\therefore\ \epsilon 011\epsilon 1 &= 0\epsilon\epsilon 111\epsilon &&
\end{aligned}
$$

Hence $\epsilon 011\epsilon 1 = 0\epsilon\epsilon 111\epsilon$.

Solution to Exercise 11.3.2.

debug: exercises/equals1/answer.tex

Solution not provided.

Solution to Exercise 11.3.3.

debug: exercises/equals2/answer.tex

Solution not provided.

11.4 Concatenation operation debug: concat.tex

Definition 11.4.1. Define the **concatenation** operation as following. If $x_0 \cdots x_{m-1}$ and $y_0 \cdots y_{n-1}$ are strings over Σ where $x_i, y_j \in \Sigma_\epsilon$, then concatenation

$$(x_0 \cdots x_{m-1}) \cdot (y_0 \cdots y_{n-1}) \stackrel{\text{def}}{=} x_0 \cdots x_{m-1} y_0 \cdots y_{n-1}$$

We will also write xy instead of $x \cdot y$.

(Some authors write $x||y$ instead of xy, but I don't like it. This is very common in cryptography.)

Sanity check: Are you really sure that all the above are consistent? No circular definitions? Sure?

Let's check the definition very carefully. First of all, of course intuitively we know that we want $(x_0 \cdots x_{m-1}) \cdot (y_0 \cdots y_{n-1})$ to be a string. We then need to check the formal definition of a string. We have accepted the definition of a string (check the definition on previous pages) to be a sequence

$$x_0 x_1 \cdots x_{m-1}$$

where each $x_i \in \Sigma_\epsilon$. And $x_0 \cdots x_{m-1} y_0 \cdots y_{n-1}$is clearly a sequence where each x_i, y_j are in Σ_ϵ. To make it even more explicit, I can define it as:

Definition 11.4.2. Define the **concatenation** operation as following. If $x_0 \cdots x_{m-1}$ and $y_0 \cdots y_{n-1}$ are strings over Σ where $x_i, y_j \in \Sigma_\epsilon$, then concatenation

$$(x_0 \cdots x_{m-1}) \cdot (y_0 \cdots y_{n-1}) \stackrel{\text{def}}{=} z_0 z_1 z_2 \cdots z_{m+n-1}$$

where

$$z_i = \begin{cases} x_i & \text{if } 0 \leq i \leq m-1 \\ y_{i-m} & \text{if } m \leq j \leq m+n-1 \end{cases}$$

We will also write xy instead of $x \cdot y$.

Compare the two definitions and make sure you can see the difference. In this case, the earlier definition is actually correct. The second definition is equivalent to the first. But the second is a little bit more explicit.

Proposition 11.4.1. *Let $x, y, z \in \Sigma^*$. Then*

$$(xy)z = x(yz) \quad (\textit{Associativity})$$

This means that we can be lazy (or efficient) and write xyz.

Exercise 11.4.1. Prove

debug: exercises/concat0/question.tex

$$xyz = x'yz' \iff xz = x'z'$$

[Hint: Induction on $|y|$.] (Go to solution, page 10013) □

Exercise 11.4.2. Given any string $x = y_0 \dots y_m$, either

debug: exercises/concat1/question.tex

- $x = \epsilon$, or
- $x = x_0 \dots x_n$ where $x_i \in \Sigma$. Specifically, there exists integers $0 \leq i_0 < i_1 < \dots < i_n \leq m$ such that

$$y_j = \epsilon \iff j \in \{0, \dots, m\} - \{i_0, \dots, \}$$

and

$$x = y_{i_0} y_{i_1} \dots y_{i_n}$$

We will call this the simplified expression for x. Furthermore, the simplified expression for x is unique in the sense that if $x = z_0 \dots z_p = z'_0 \dots z'_q$, then $p = q$, and $z_i = z'_i$ for $0 \leq i \leq p$.

(Go to solution, page 10014) □

Definition 11.4.3. Let $x, y \in \Sigma^*$. Then $x \neq y$ iff their simplied expression are not the same, i.e., if their simplified expressions are $x = x_0 \dots x_m$ and $y = y_0 \dots y_n$, then either $n \neq m$, or there is some i such that $x_i \neq y_i$.

(For you math experts out there, Σ^* is a free semigroup on Σ where concatenation is the semigroup operation.)

Solutions

Solution to Exercise 11.4.1.

debug: exercises/concat0/answer.tex

Solution not provided.

Solution to Exercise 11.4.2.

debug: exercises/concat1/answer.tex

Solution not provided.

11.5 Exponentiation

debug: exponentiation.tex

Because we are such advanced life forms obviously we don't want to write $xxxxxxxxxx$. So ...

Definition 11.5.1. The definition of x^k for $n \geq 0$ is obvious. Here's the formal definition using recursion:

1. $x^0 = \epsilon$
2. $x^{n+1} = x^n x$ if $n \geq 0$. (NOTE: ORDER CHANGED.)

□

Exercise 11.5.1. Prove or disprove the following: Let $m, n \geq 0$ be integers.

debug: exercises/exponentiation0/question.tex

(a) $x^{n+1} = xx^n$
(b) $(x^m)(x^n) = x^{m+n}$
(c) $(x^m)(x^n) = (x^n)(x^m)$
(d) $(x^m)^n = x^{mn}$

(Go to solution, page 10016)

□

Solutions

Solution to Exercise 11.5.1.

debug: exercises/exponentiation0/answer.tex

Solution not provided.

11.6 Concatenation of languages and Kleene star

debug: kleenestar.tex

I already talked about exponentiation of a word, i.e., x^n where x is a string. Now I want to talk about the exponentiation of a set of words (i.e. exponentiation of languages).

Definition 11.6.1. Let L, L' be languages over Σ. The **concatenation** of L and L' is defined to be concatenation

$$L \cdot L' = \{x \cdot x' \mid x \in L,\, x' \in L'\}$$

We will usually write

$$LL'$$

instead of

$$L \cdot L'$$

Let's do a quick example: Suppose $L_1 = \{a, ba, bab, baba\}$ and $L_2 = \{b, ab, aab, aaab\}$. The product L_1L_2 is

$$\begin{aligned} L_1L_2 = \{&a \cdot b, a \cdot ab, a \cdot aab, a \cdot aaab,\\ &ba \cdot b, ba \cdot ab, ba \cdot aab, ba \cdot aaab,\end{aligned}$$

$$\begin{aligned} = \{&ab, aab, aaab, aaaab,\\ &bab, baab, baaab, baaaab,\end{aligned}$$

$$\begin{aligned} = \{&ab, a^2b, a^3b, a^4b,\\ &bab, ba^2b, ba^3b, ba^4b,\end{aligned}$$

(finish it!) Now check which of the following strings are in L_1L_2:

1. ϵ
2. a
3. b
4. aa
5. ab
6. ba
7. bb
8. aaa
9. aab
10. aba
11. baa
12. abb
13. bab
14. bba
15. bbb

Exercise 11.6.1. debug: exercises/kleenestar0/question.tex

1. $\{a\} \cdot \{b\}$
2. $\{\epsilon\} \cdot \{b\}$
3. $\emptyset \cdot \{ab\}$
4. $\{a, aa\} \cdot \{b\}$
5. $\{a, aa, aaa\} \cdot \{b, ba, baa\}$

(Go to solution, page 10022) □

Exercise 11.6.2. Is it true that if L and L' are finite languages, then $|LL'|$ (the number of words in LL') must be $|L| \cdot |L'|$? (Go to solution, page 10023) debug: exercises/kleenestar1/question.tex □

Exercise 11.6.3. debug: exercises/kleenestar2/question.tex

1. Prove that if $L = \emptyset$ or $L' = \emptyset$, then
$$L \cdot L' = \emptyset$$
2. Is it true that if $LL' = \emptyset$, then either $L = \emptyset$ or $L' = \emptyset$?

By the way, are the following two languages the same:

$$L = \{\}, \qquad L' = \{\epsilon\}$$

(Go to solution, page 10024) □

Exercise 11.6.4. Let $L_1 = \{a^n \mid n \geq 0\}$ and $L_2 = \{ab^n a \mid n \geq 0\}$. Which of the following strings are in L_1L_2? (Note that in this case the languages are infinite.)

debug: exercises/kleenestar3/question.tex

1. ϵ
2. a
3. b
4. aa
5. ab
6. ba
7. bb
8. aaa
9. aab
10. aba
11. baa
12. abb
13. bab
14. bba
15. bbb

(Go to solution, page 10025) □

It's not shocking that if I have three languages L, L', and L'', then $LL'L''$ is defined as

$$LL'L'' = (LL')L''$$

This is just like for real value variables x, y, z, when I write $x + y + z$, I mean $(x + y) + z$. Because $(x + y) + z = x + (y + z)$, it doesn't really matter which $+$ you perform first which is why we omit parentheses from either $(x + y) + z$ or from $x + (y + z)$.

It's not too surprising that

Proposition 11.6.1. $(LL')L'' = L(L'L'')$.

Proof. DIY. [Hint: The proof depends on the fact that $(xy)z = x(yz)$ if x, y, z are words in L, L', L'' respectively.] □

Frequently, we want to concatenate the same language. So let's have a shorthand for that ...

We will write

$$L^2$$

instead of LL. In general we define

$$\begin{aligned} L^0 &= \{\epsilon\} \\ L^{n+1} &= LL^n \quad \text{for } n \geq 0 \end{aligned}$$

We also define

$$\begin{aligned} L^* &= \bigcup_{n=0}^{\infty} L^n = L^0 \cup L^1 \cup L^2 \cup \cdots \\ L^+ &= \bigcup_{n=1}^{\infty} L^n = L^1 \cup L^2 \cup \cdots \end{aligned}$$

Exercise 11.6.5. If $L = \{a\}$, what is L^*? What about L^+? (Go to solution, page 10026) □ debug: exercises/kleenestar4/question.tex

Exercise 11.6.6. If $L = \{ab\}$, what is L^*? What about L^+? (Go to solution, page 10027) □ debug: exercises/kleenestar5/question.tex

Exercise 11.6.7. If $L = \{a^n b \mid n \geq 0\}$, what is L^*? What about L^+? (Go to solution, page 10028) □ debug: exercises/kleenestar6/question.tex

Exercise 11.6.8. Is it true that for any language L, $L^2 \neq L$? (Go to solution, page 10029) □ debug: exercises/kleenestar7/question.tex

Solutions

Solution to Exercise 11.6.1.

debug: exercises/kleenestar0/answer.tex

Solution not provided.

Solution to Exercise 11.6.2.

debug: exercises/kleenestar1/answer.tex

No. $L = \{\epsilon, a\}$, $L' = L$

Solution to Exercise 11.6.3.

debug: exercises/kleenestar2/answer.tex

Solution not provided.

Solution to Exercise 11.6.4.

debug: exercises/kleenestar3/answer.tex

Solution not provided.

Solution to Exercise 11.6.5.

debug: exercises/kleenestar4/answer.tex

Solution not provided.

Solution to Exercise 11.6.6.

debug: exercises/kleenestar5/answer.tex

Solution not provided.

Solution to Exercise 11.6.7.

debug: exercises/kleenestar6/answer.tex

Solution not provided.

Solution to Exercise 11.6.8.

debug: exercises/kleenestar7/answer.tex

Solution not provided.

11.7 Length debug: length.tex

Definition 11.7.1. Let $x \in \Sigma^*$. We want to define $|x|$, the **length** of x.

- If $x = \epsilon$, then
$$|x| = |\epsilon| = 0$$
- Suppose $x \neq \epsilon$. Recall from the above, if $x \neq \epsilon$, then $x = x_1 \dots x_n$ where $x_i \in \Sigma$. From the above proposition, this is always possible. In this case we define
$$|x| = n$$

Obviously $|\cdot|$ is a function from Σ^* to $\mathbb{N}$:

$$|\cdot| : \Sigma^* \to \mathbb{N}$$

A more recursive definition of length is this ... Let $x \in \Sigma^*$. If $x = \epsilon$,

$$|x| = 0$$

and if $x \neq \epsilon$, then $x = cx'$ where $c \in \Sigma$ and $x' \in \Sigma^*$. In this case

$$|x| = 1 + |x'|$$

Proposition 11.7.1. *Let $x, y \in \Sigma^*$. Then*

$$|xy| = |x| + |y|$$

Definition 11.7.2. Let $k \geq 0$. Define

$$\Sigma^k \stackrel{\text{def}}{=} \{\, x \in \Sigma^* \mid |x| = k \,\}$$

Proposition 11.7.2. *The following are pretty obvious*

- $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$
- $\Sigma^+ = \bigcup_{k=1}^{\infty} \Sigma^k$

Example 11.7.1. Let $\Sigma = \{a, b, c\}$.

- Let $x = aa\epsilon cba$. Then $|x| = 5$, i.e., $x \in \Sigma^5$.
- Let $y = ccc$. Then $xy = aacbaccc$.
- We have $\epsilon x = aacba$

Exercise 11.7.1. Let $\Sigma = \{\langle the\rangle, \langle boy\rangle, \langle girl\rangle, \langle kicks\rangle, \langle ball\rangle, \langle cat\rangle, \langle and\rangle\}$ debug: exercises/length0/question.tex

- What is Σ^0?
- What is Σ^1?
- What is Σ^2?
- What is the length of $\langle the\rangle\langle boy\rangle\langle kicks\rangle\langle the\rangle\langle ball\rangle$?
- What is the length of $\langle cat\rangle\langle the\rangle\langle ball\rangle\langle girl\rangle$?
- Let $x = \langle the\rangle\langle cat\rangle$, $y = \langle kicks\rangle\langle and\rangle$. What is $x\epsilon y^3\langle kicks\rangle x$?

(Go to solution, page 10034) □

Proposition 11.7.3. *Let $x \in \Sigma^*$ with $|x| = k > 0$.*

- *There is some $a \in \Sigma$ and some $y \in \Sigma^{k-1}$ such that $x = ay$*
- *There is some $b \in \Sigma$ and some $z \in \Sigma^{k-1}$ such that $x = zb$.*

The proof is trivial!

Solutions

Solution to Exercise 11.7.1.

debug: exercises/length0/answer.tex

Solution not provided.

11.8 Reversal function

debug: reverse.tex

Definition 11.8.1. Let x be a string. The **reversal** of x, denoted x^R, is just the same string with the letters reversed. Formally: reversal

- If $|x| = 0$, then $x^R = \epsilon^R = \epsilon$.
- If $|x| > 0$, then $x = ay$ where $a \in \Sigma$ and $x^R = y^R a$.

Exercise 11.8.1. Let $x, y \in \Sigma^*$. Prove that $(xy)^R = y^R x^R$. (Yes, I know it's obvious, but you still have to prove it.) [Hint: Induction.] (Go to solution, page 10036) □ debug: exercises/reverse0/question.tex

Now that I'm done defining the reversal of a string, I can define the reversal of a language.

Definition 11.8.2. Let L is a language then the reversal of a language is

$$L^R = \{x^R \mid x \in L\}$$

For instance if $L = \{\epsilon, a, ab, abaa, abaaa\}$, then

$$L^R = \{\epsilon, a, ba, aaba, aaaba\}$$

Solutions

Solution to Exercise 11.8.1.

debug: exercises/reverse0/answer.tex

Solution not provided.

Index