Chapter Seven:
Regular Expressions

Regular Expressions

- We have seen that DFAs and NFAs have equal definitional power.
- It turns out that *regular expressions* also have exactly that same definitional power:
  - they can be used to define all the regular languages, and *only* the regular languages.

Outline

- 7.1 Regular Expressions, Formally Defined
- 7.2 Regular Expression Examples
- 7.3 For Every Regular Expression, a Regular Language
- 7.4 Regular Expressions and Structural Induction
- 7.5 For Every Regular Language, a Regular Expression

Regular Expression

- In order to define regular expressions we need to additional operators on languages:
  - Concatenation
  - Kleene closure

Concatenation of Languages

- The concatenation of two languages L_1 and L_2 is $L_1L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$
- The set of all strings that can be constructed by concatenating a string from the first language with a string from the second
- For example, if $L_1 = \{a, b\}$ and $L_2 = \{c, d\}$ then $L_1L_2 = \{ac, ad, bc, bd\}$

Kleene Closure of a Language

- The Kleene closure of a language L is $L^* = \{x_1x_2 \ldots x_n \mid n \geq 0, \text{ with all } x_i \in L\}$
- The set of strings that can be formed by concatenating any number of strings, each of which is an element of L
- In L^*, each x_i may be a different element of L
- For example, $\{ab, cd\}^* = \{\varepsilon, ab, cd, abab, abcd, cdab, cdcd, ababab, \ldots\}$
- For all L, $\varepsilon \in L^*$
- For all L containing at least one string other than ε, L^* is infinite

Note: this is very similar to the set of all strings Σ^* over alphabet Σ. In fact, sometimes we talk about the Kleene closure of the alphabet.

Regular Expressions

- A regular expression is a string *r* that denotes a language *L*(*r*) over some alphabet Σ
- Regular expressions make special use of the symbols ε, $\emptyset$, +, *, and parentheses
- We will assume that these special symbols are not included in Σ
- There are six kinds of regular expressions…

The Six Regular Expressions

- The six kinds of regular expressions, and the languages they denote, are:
  - Three kinds of *atomic* regular expressions:
    - Any symbol $a \in \Sigma$, with $L(a) = \{a\}$
    - The special symbol ε, with $L(\varepsilon) = \{\varepsilon\}$
    - The special symbol $\emptyset$, with $L(\emptyset) = \{\}$
  - Three kinds of *compound* regular expressions built from smaller regular expressions, here called r, r_1, and r_2:
    - $(r_1 + r_2)$, with $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
    - (r_1r_2), with $L(r_1r_2) = L(r_1)L(r_2)$
    - $(r)^*$, with $L((r)^*) = (L(r))^*$
- The parentheses may be omitted, in which case * has highest precedence and + has lowest

Outline

ab

- Denotes the language $\{ab\}$
- Our formal definition permits this because
  - *a* is an atomic regular expression denoting $\{a\}$
  - *b* is an atomic regular expression denoting $\{b\}$
  - Their concatenation (ab) is a compound
  - Unnecessary parentheses can be omitted
- Thus any string x in Σ^* can be used by itself as a regular expression, denoting $\{x\}$

ab+c

- Denotes the language {*ab*,*c*}
- We omitted parentheses from the fully parenthesized form ((*ab*)+*c*)
- The inner pair is unnecessary because + has lower precedence than concatenation
- Thus any finite language can be defined using a regular expression
- Just list the strings, separated by +

Hint: when you see a “+’ in a RE just think “or”

*ba**

- Denotes the language $\{ba^n | n \geq 0\}$: the set of strings consisting of *b* followed by zero or more *a*s
- Not the same as (*ba*)*, which denotes $\{(ba)^n | n \geq 0\}$
- * has higher precedence than concatenation
- The Kleene star is the only way to define an infinite language using regular expressions

$(a+b)^*$

- Denotes $\{a,b\}^*$: the whole language of strings over the alphabet $\{a,b\}$
- The parentheses are necessary here, because * has higher precedence than +
- Kleene closure does not distribute, that is,
  - $(a+b)^* \neq a^*+b^*$
  - a^*+b^* denotes $\{a\}^* \cup \{b\}^*$

$\varnothing$

- Denotes {}
- There is no other way to denote the empty set with regular expressions
- That's all you should ever use $\varnothing$ for
- It is not useful in compounds:
  - $L(r\varnothing) = L(\varnothing r) = \{\}$
  - $L(r+\varnothing) = L(\varnothing+r) = L(r)$
  - $L(\varnothing^*) = \{\varepsilon\}$

From Languages to RE

- Give the regular expressions for the following languages:
  - {x | x is a string that starts with three 0s followed by arbitrary 0s and 1s and then ends with three 0s}
  - {x | x is a string that starts with a 0 followed by an arbitrary number of 1s and ends with a 0 OR x is a string that starts with a 1 followed by an arbitrary number of 0s and ends with a 1}
  - { x^n | x is either the string ab or the string c and n >= 0}

Outline

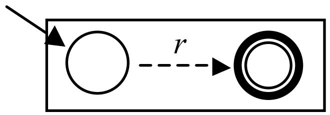
- 7.1 Regular Expressions, Formally Defined
- 7.2 Regular Expression Examples
- 7.3 For Every Regular Expression, a Regular Language
- 7.4 Regular Expressions and Structural Induction
- 7.5 For Every Regular Language, a Regular Expression

Regular Expression to NFA

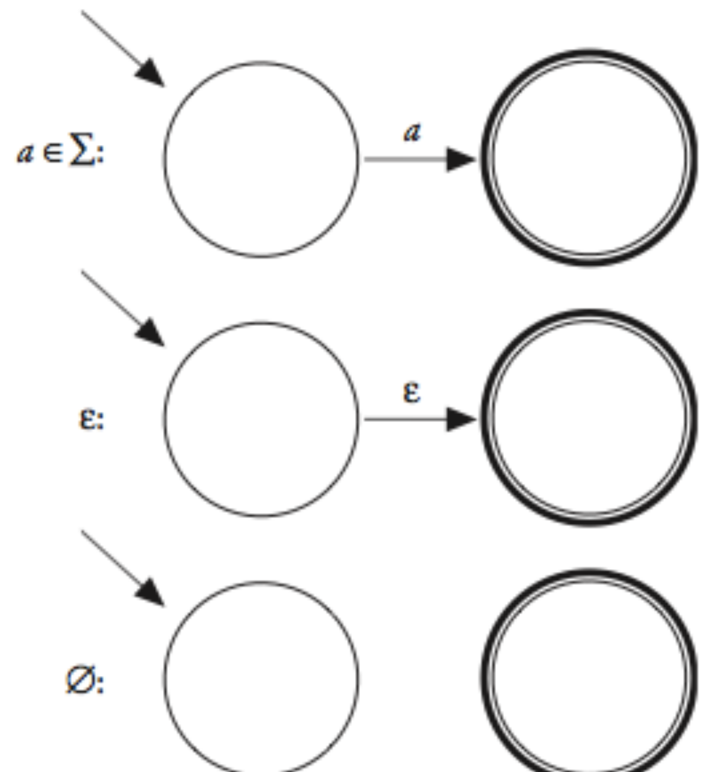
- Approach: convert any regular expression to an NFA for the same language

Standard Form

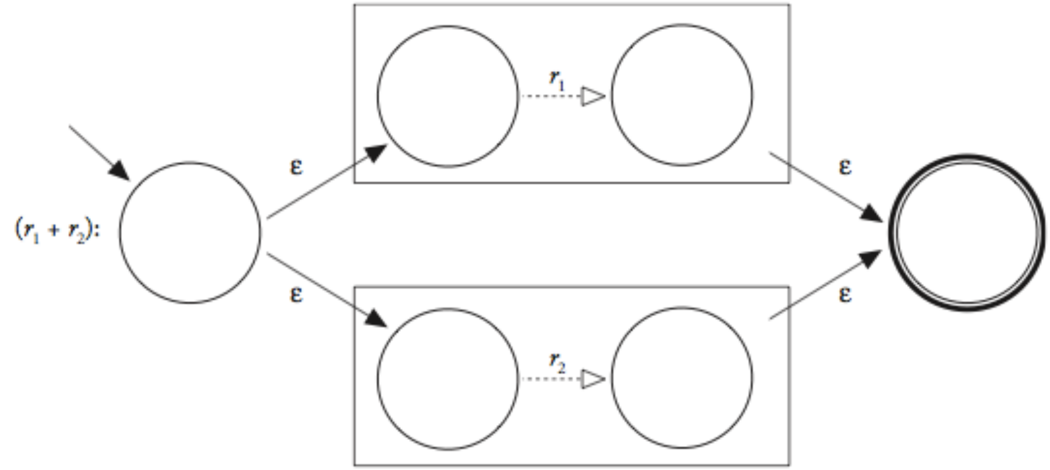
- To make them easier to compose, our NFAs will all have the same standard form:
  - Exactly one accepting state, not the start state
- That is, for any regular expression *r*, we will show how to construct an NFA *N* with *L*(*N*) = *L*(*r*), pictured like this:

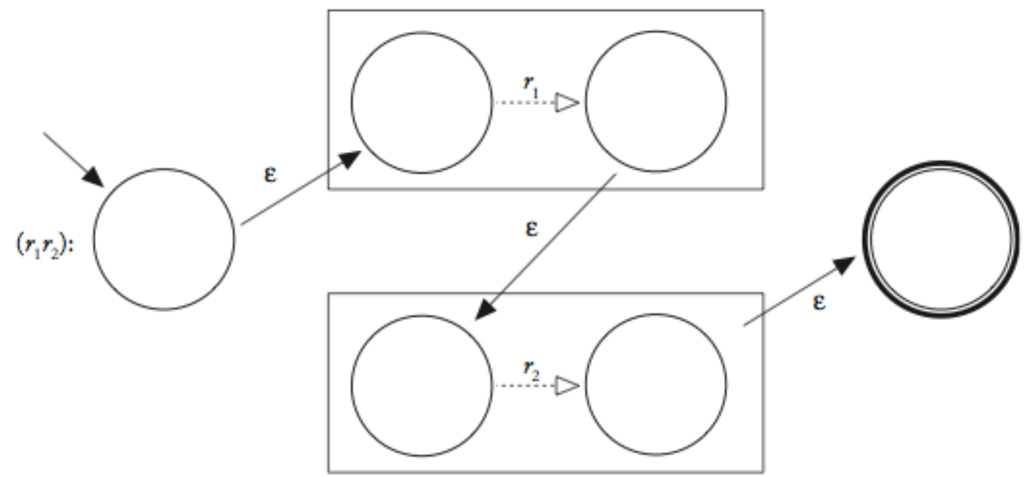


Atomic REs

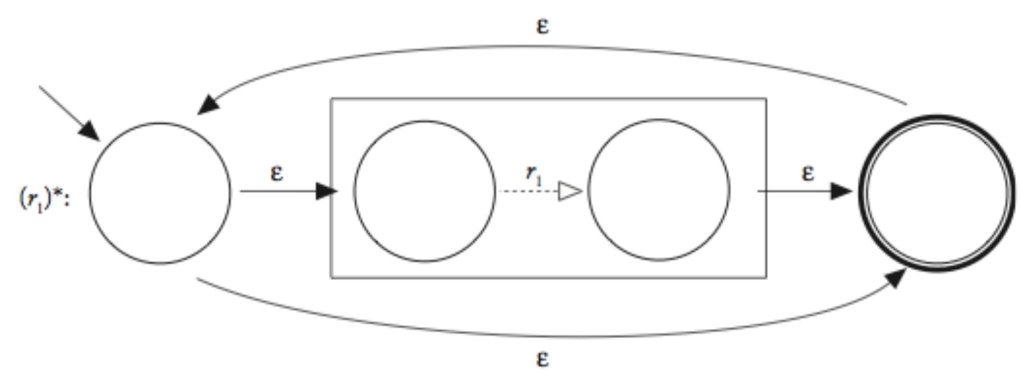


Compound REs





where r_1 and r_2 are REs



Outline

- 7.1 Regular Expressions, Formally Defined
- 7.2 Regular Expression Examples
- 7.3 For Every Regular Expression, a Regular Language
- 7.4 Regular Expressions and Structural Induction
- 7.5 For Every Regular Language, a Regular Expression

NFA to Regular Expression

- There is a way to take any NFA and construct a regular expression for the same language
- This gives us our next lemma:
- Lemma: if N is any NFA, there is some regular expression r with $L(r) = L(N)$
- A tricky construction, covered in Appendix A (very difficult to follow), the hand-out has a more intuitive proof.

Theorem (Kleene's Theorem)

A language is regular if and only if it is $L(r)$ for some regular expression r.

- Proof: follows from previous two lemmas.

Defining Regular Languages

- We can define the regular languages:
  - By DFA
  - By NFA
  - By regular expression
- These three have equal power for defining languages

Alphabets

- An *alphabet* is any finite set of symbols
  - {0,1}: binary alphabet
  - {0,1,2,3,4,5,6,7,8,9}: decimal alphabet
  - ASCII, Unicode: machine-text alphabets
  - Or just {a,b}: enough for many examples
  - {}: a legal but not usually interesting alphabet
- We will usually use Σ as the name of the alphabet we're considering, as in $\Sigma = \{a,b\}$

Strings

- A *string* is a finite sequence of zero or more symbols
- Length of a string: $|abbb| = 4$
- *A string over the alphabet* Σ means *a string all of whose symbols are in* Σ
  - The set of all strings of length 2 over the alphabet $\{a,b\}$ is $\{aa, ab, ba, bb\}$

Languages

- A *language* is a set of strings over some fixed alphabet
- *Not* restricted to finite sets: in fact, finite sets are not usually interesting languages
- All our alphabets are finite, and all our strings are finite, but most of the languages we're interested in are infinite

The Quest

- Using set formers to describe complex languages is challenging
- They can often be vague, ambiguous, or self-contradictory
- A big part of our quest in the study of formal language is to develop better tools for defining and classifying languages

The Quest

- We went from this:
  - {x | x is a string that starts with three 0s followed by arbitrary 0s and 1s and then ends with three 0s}
- to this:
  - 000(0+1)*000

The Quest

- We just defined a major class of languages:
  - the regular languages
- The hallmark of these languages is that their structure is such that simple computational models (DFA/NFA) can recognize them and that they can be defined using regular expressions.

The Quest

- The idea that the structure of languages is connected to computational models is important.
- Later on we see that the structure of languages is tightly coupled with idea of algorithms and classes of computational problems.

Assignment #4

- See website