Conversions and Regular Expressions

CS154

Chris Pollett

Feb 14, 2007.

Outline

- Regular Expressions

Regular Expressions

- In arithmetic, we can use the operations + and * to build up expressions such as:
  $(5 + 3) * 4$.
- Similarly we can use the regular operations to build up expressions describing regular languages.
- For instance, $0(0 \cup 1)^*$ (We use juxtaposition to abbreviate concatenation: $0 \circ (0 \cup 1)^*$).
- This means the language which results from concatenating the language containing 0 with the language of $(0 \cup 1)^*$. This in turn is the star of the union of the two languages one containing just 0; the other containing just 1.
- These kind of expressions are used in many modern programming languages: Perl, PHP, Java, AWK, GREP

Formal Definition of a Regular Expression

- We say that R is a regular expression if R is
  1. *a* for some symbol *a* in the alphabet Σ,
  2. ε (notice we are using ε but the book uses λ)
  3. $\varnothing$
  4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions. The book writes $(R_1 + R_2)$ for this case, + is also used by JFLAP.
  5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
  6. $(R_1)^*$ where R_1 is a regular expression
- We write R^+ as a shorthand for RR^*.
- We write L(R) for the language given by the regular expression

Examples of the Definition

- $0^*1\ 0^* = \{w \mid w \text{ contains a single } 1\}$
- $(01 \cup 10) = \{01, 10\}$
- $(\Sigma\Sigma)^* = \{w \mid w \text{ is of even length}\}$
- $(\varepsilon \cup 0)(\varepsilon \cup 1) = \{\varepsilon, 0, 1, 01\}$
- $1^* \varnothing = \varnothing$
- $\varnothing^* = \{\varepsilon\}$

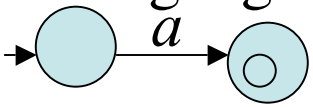
Equivalence with Finite Automata

- We want to show that a language is regular if and only if some regular expression describes it.
- We will do this in two steps:
  - Prove if a language is described by a regular expression, then it is regular
  - Prove if a language is regular, then it is described by a regular expression.

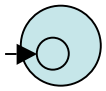
Proof that regular expression implies regular

- The proof is by induction on the complexity (number of uses of union, *, or concatenation) of the regular expression. In the base case, we have regular expressions which make no use of union, *, or concatenation.

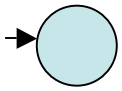
1. Let R= *a* for some *a* in ∑. Then the following NFA recognizes the languages contain only *a*.



2. Let R= ε. Then the following NFA recognizes it:



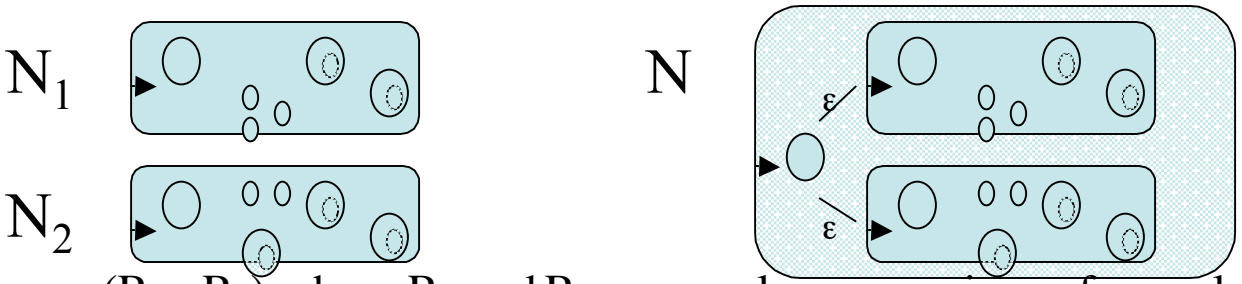
3. Let R= ∅. Then the following NFA recognizes it:



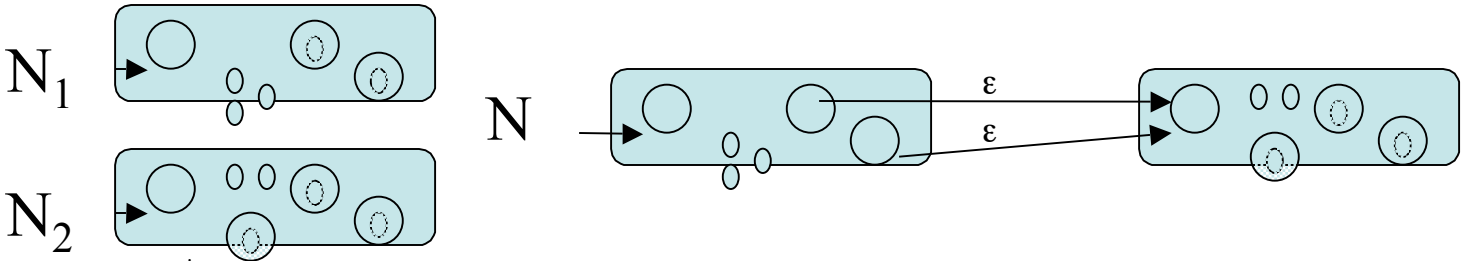
Proof cont'd

- Assume now the result holds for languages for which the total number of uses of union, *, or concatenation is at most n. Consider R a regular language of complexity n+1. There are three cases to consider:

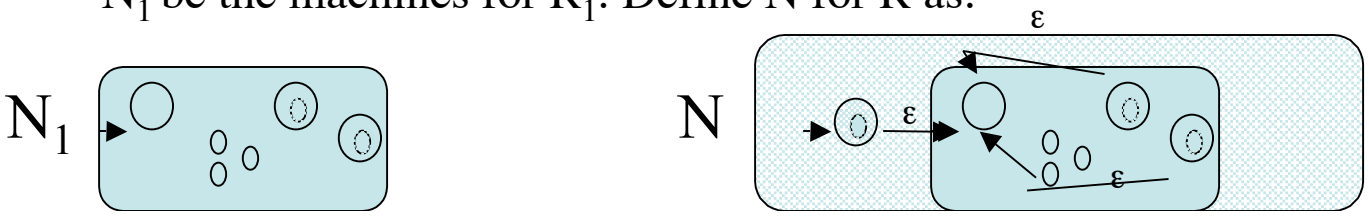
1. R is of the form $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions of complexity ≤n. By induction let N_1 and N_2 be the machines for R_1 and R_2. Define N for R as:



2. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions of complexity ≤n. By induction let N_1 and N_2 be the machines for R_1 and R_2. Define N for R as:

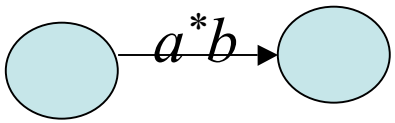


3. $(R_1)^*$ where R_1 is a regular expression of complexity ≤n. By induction let N_1 be the machines for R_1. Define N for R as:



Proof that regular implies the language of some regular expression

- We will split the proof into two parts:
  - We first define a new kind of finite automata called a generalized nondeterministic finite automata (GNFA) and show how to convert any DFA into a GNFA.
  - Then we show how to convert any GNFA into a regular expression.
- To begin we define a GNFA to be an NFA where we allow transition arrows to have any regular expression as labels:



Converting DFAs to GFNA

- We will be interested in GNFAs that have the following special form:
  - The start state has transition arrows to every other state but no arrows coming in from other states.
  - There is a single accept state, and it has arrows coming in from every other state but no arrows going to any other state.
  - Except for the start and accepts state, one arrow goes from from every state to every other state and also from each state to itself.
- To convert a DFA into a GNFA, we add a new start state with and ε arrow to the old start state and a new accept state with ε arrows from the old accept states.
- If any arrows have multiple labels (or if we have two or more arrows between the same two states) we replace each with a single label whose label is the union labels of the these arrows.
- Finally, we add arrows with labels $\varnothing$ between states which had no labels so as to satisfy the remaining conditions of our special form.

Converting GNFAs to Regular expressions

- Our conversion above gives a GNFA with k >= 2 states.
- If k > 2, we will construct an equivalent GNFA with k-1 states.
- To do this we pick some state q_{rip} other than the start or accept state, and we will rip it out of the machine.
- To compensate for the loss of this state, for any pair of states q_i, q_j. in this new machine we replace $\delta(q_i, q_j)$ with:

  $\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$

  where $\delta(q_i, q_{rip}) = R_1$; $\delta(q_{rip}, q_{rip}) = R_2$; $\delta(q_{rip}, q_j) = R_3$; $\delta(q_i, q_j) = R_4$
- This machine will be equivalent to the old machine.
- Further, by repeatedly ripping out states in this fashion we can get down to the 2-state machine with just a regular expression on the single transition between these two states.
- This regular expression will be equivalent to the original NFA.

A Corollary

The regular languages are closed under union, *, and concatenation.

Proof. The regular languages are precisely those recognized by DFAs. We have shown in turn that the languages recognized by DFAs are precisely those recognized by NFAs, and these in turn are precisely the languages recognized by regular expression. As the languages of regular expressions are trivially closed under these operations, we get the regular languages are closed under these operations.