Complexity of Conversions and Tests

MIEIC, 2nd Year

João M. P. Cardoso
Email: jmpc@acm.org

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO
DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Outline

- Regular Languages (RLs)
- Context-Free Languages (CFLs)

Regular Languages (RLs)

Converting among representations

- In the context of Regular Languages
    - i.e., conversions between DFAs, NFAs, ε-NFAs, and regular expressions

Converting among representations

- NFA (ε-NFA) → DFA
  - A function of the number of states *n* of the NFA
  - Calculate closure-ε: $O(n^3)$ [*n* states and a maximum of n^2 transitions per state]
  - Construction of subsets: $O(2^n)$ [2^n is the maximum number of DFA states]
  - Compute the transitions δ for each state: $O(n^3)$
  - We consider the alphabet fixed and thus it only influences the constant hiden in the O() ("big-oh") notation

  - **Complete conversion: $O(n^3 2^n)$**
    - As the number os states *s* (often close to *n*) of the DFA is frequently much lower than exponencial (**2^n**): **$O(n^3 s)$**

Converting among representations

- DFA → NFA
  - **O(n)** [just a copy with some modifications]
- NFA/DFA → RE
  - Using the path construction:
    - n^3 to compute the table [n^2 rows and n columns (steps)]
    - In each step we have n^2 expressions and each expression is built using 4 expressions of the previous step
    - 4^n as size of regular expression grows by 4 every time
    - **$O(n^3 4^n)$**
    - In practice it is close to $O(n^3)$
      - By simplifying the regular expression at every step and
      - Using judicious algorithm avoiding recomputation of $R_{kk}^{(k)}$
  - Using State Elimination:
    - State elimination technique: *n* steps
    - And what about the full conversion?
- NFA → RE
  - If we start by converting first the NFA to a DFA, we obtain a doubly exponencial algorithm!

Converting among representations

- RE → ε-NFA
  - Being *n* the length of the expression
  - Construct the expression tree in **O(n)**
  - Number of states and arcs are **O(n)**
  - Complete conversion: **O(n)** [proportional to n]
- RE → NFA
  - Being *n* the length of the expression, RE → ε-NFA: **O(n)**
  - Complete conversion: **O(n^3)** [being n the number of the states of the ε-NFA]
- RE → DFA
  - Being *n* the length of the expression: can take exponential [because of NFA (ε-NFA) → DFA]

Minimization of DFAs

- Table-filling algorithm to find if two states are equivalent or not:
  - There are *n(n-1)/2* pairs of states
  - A round takes $\mathbf{O(n^2)}$ and there will be no more than n^2 rounds
    - Thus: $\mathbf{O(n^4)}$ if not selecting a carefully algorithm
  - Can be $\mathbf{O(n^2)}$ if:
    - an initialization step stores the list of pairs dependent on each pair, and
    - for each distinguishable pair found, all the dependent pairs not already distinguishable are marked as distinguishable

Testing Emptiness of Regular Languages (RLs)

- Given an FA for the language, find if there is a path from the start to a final state: **$O(n^2)$** [n is the number of states of the FA]
- We can also test emptiness directly from a regular expression (RE):
  - Check for the existence of $\emptyset$ in RE and
    - if there is, check recursively for the conditions for emptiness for concatenation, union, kleen and parêntesis
    - If there is not, L(RE) is not empty

Testing Membership in a Regular Language (RL)

- Using a DFA:
  - String with length *n*
  - **O(n)**
- Using an NFA (ε-NFA):
  - String with length *n* and NFA with *s* states
  - **O(ns^2)**
- Using an RE of size s:
  - Convert to ε-NFA of at most *2s* states: O(s)
  - **O(ns^2)** [as previously]

Context-Free Languages (CFLs)

Complexity of the conversions

- Linear conversions in the length of the representation, **O(n)**
  - **CFG → PDA**
  - PDA with final state → PDA with empty stack
  - PDA with empty stack → PDA with final state
- Conversion $O(n^3)$ [see Hopcroft, Motwani, and Ullman book]
  - **PDA → CFG** (size of the CFG is also $\mathbf{O(n^3)}$)
  - *n* represents the number of transition rules
  - Each transition rule generates n^2 productions
- Conversion $\mathbf{O(n^2)}$ [see Hopcroft, Motwani, and Ullman book]
  - **CFG → CNF** (size of the CNF is also $\mathbf{O(n^2)}$)
  - Constructing unit pairs and eliminating unit productions takes $\mathbf{O(n^2)}$ while the other transformations can be done in **O(n)** [considering that the elimination of the ε-productions is done after the breaking of production bodies of length 3 or more to 2]

Testing Emptiness of CFL's

- Find if start symbol is generating, if it generates then the CFL is not empty
  - Being *n* the size of the CFG, there are at most *n* variables in the CFG
  - Complexity: $\mathbf{O(n^2)}$
- With an efficient data-structure it can take: **O(n)** [see Fig. 7.11 of Hopcroft, Motwani, and Ullman book, *Introduction to Automata Theory, Languages, and Computation*]

Testing Membership in a CFL: CYK Algorithm*

- Test if a string is in the language: **O(n^3)**
  - *n* is the length of the string
  - Table with *n(n+1)/2* cells
  - For each cell, a maximum of *n-1* pairs of cells to consider
  - Note that the grammar is fixed: each cell pair has a maximum fixed $|V|^2$ pairs of variables, with |V| the number of variables of the grammar (CNF)

* Based on the idea of “dynamic programming”, and also known as a “table-filling algorithm” or “tabulation”