T-61.182 Robustness in Language and Speech Processing

Chapter 6: Regular Approximation of Context-Free Grammars

Chapter 7: Weighted Grammar Tools: The GRM Library

Mehryar Mohri (and Mark-Jan Nederhof)

presented by Tapani Raiko

Mar 27, 2003

Contents

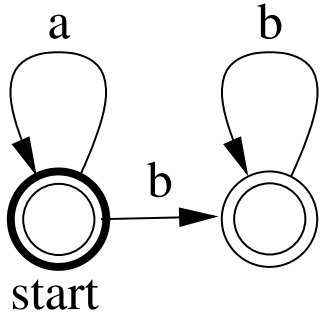
- Introduction
- Context free grammars
- Regular languages ($\sim$ finite automata)
- The general grammar (GRM) library
- Transforming strongly regular grammars into finite automata
- Approximating other grammars by strongly regular grammars
- Conclusions

Example

Context–free grammar

Terminals	{a,b}
Nonterminals	{S}
Rules	{S–> . , S–>aSb}
Start symbol	S
Language	{. , ab, aabb, ...}

Finite state machine



Language {. , a, b, aa, ab, bb, ...}

Introduction

- Natural languages are often modelled as context-free grammars (CFGs)
- Regular languages are computationally less demanding than CFGs
  - but still model syntactic and semantic properties better than n-grams
- Some CFGs can be transformed into regular languages
- The rest can be approximatively transformed

Context-Free Grammars

- **Terminals**: alphabet of a finite set of symbols
- **Nonterminals**
- **Rules** map nonterminals to strings of terminals and nonterminals
- **Start symbol** is one of the nonterminals
- Language defined by the grammar is the reflexive closure of the start symbol using the rules (a set of strings of terminals)
- Qubic time complexity for parsing
  $\rightarrow$ in most real-time applications too demanding

Regular Languages

- Language is a set of alphabet strings
- Regular grammars are those that generate regular languages
- Regular languages are those that are accepted by a finite automaton
- Linear time complexity for parsing
- Note that a mapping from a regular grammar into a corresponding finite automaton cannot be realized by any algorithm (Ullian 1967)
- Strongly regular grammars can be mapped to finite automata

Finite Automata

- A finite state machine (FSM) consists of
  - a finite set of states
  - an alphabet (a finite set of symbols)
  - (a finite set of) transitions from a state to a state with a corresponding symbol
  - start state
  - end states
- A language defined by a FSM corresponds to all the paths through the machine

General Grammar (GRM) Library

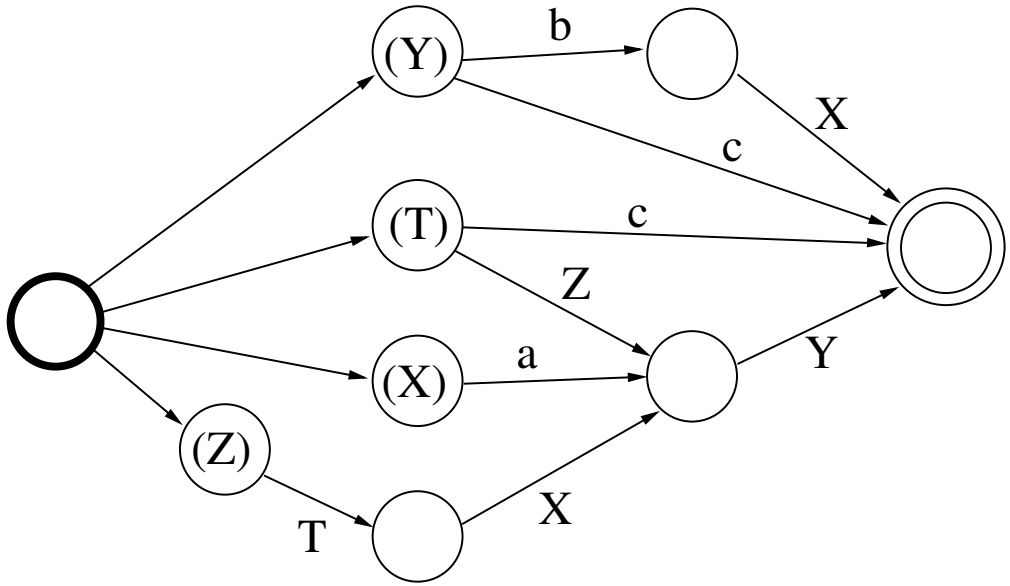
- Software library designed for use in spoken-dialogue systems, speech synthesis and other speech processing applications
- Based on the FSM library (by same authors)
- Implements weighted rules and grammars
- Generality: Implementation does not depend on the grammar
  $\longrightarrow$ grammar can be dynamically modified
- Efficiency: On-the-fly algorithms expand the automata for the specific input sequence
- Three access levels: command line, C library and source code
- Handles also context-dependent rules (skipped here)

Transforming Strongly Regular Grammars into Finite Automata

- Why?
  - CFGs are computationally too demanding in real-time applications
  - FSMs can be optimized by general algorithms (ϵ-removal, ϵ-normalization, determinization, and minimization of weighted transducers)
- Algorithm consists of 6 consequtive steps...

Step 1: Compact transducer representation

T –> ZY
T –> c
Z –> TXY
X –> aY
Y –> bX
Y –> c



Step 2: Dependency graph
Step 3: Strongly connected components

T –> ZY

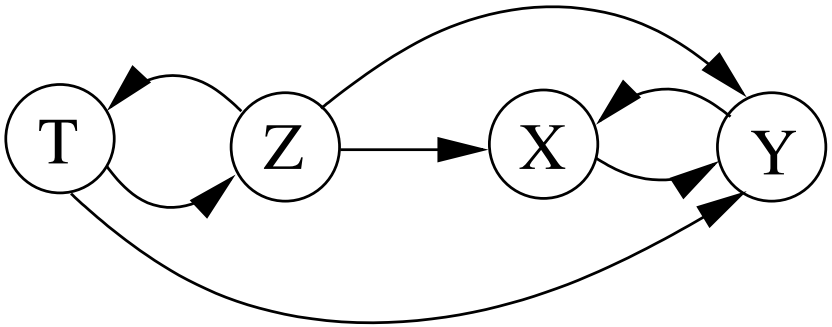
T –> c

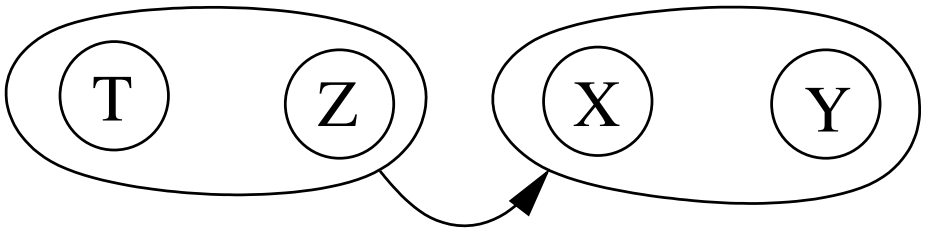
Z –> TXY

X –> aY

Y –> bX

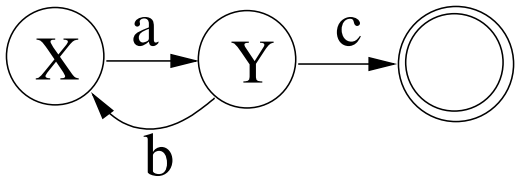
Y –> c

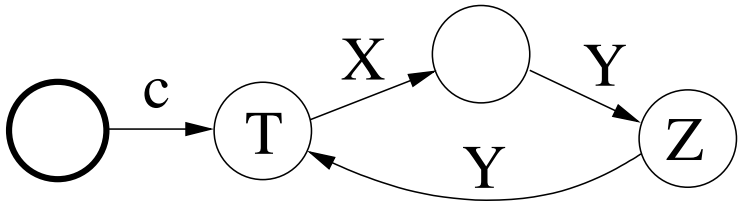




Step 4: Automaton for each component

T –> ZY
T –> c
Z –> TXY
X –> aY
Y –> bX
Y –> c

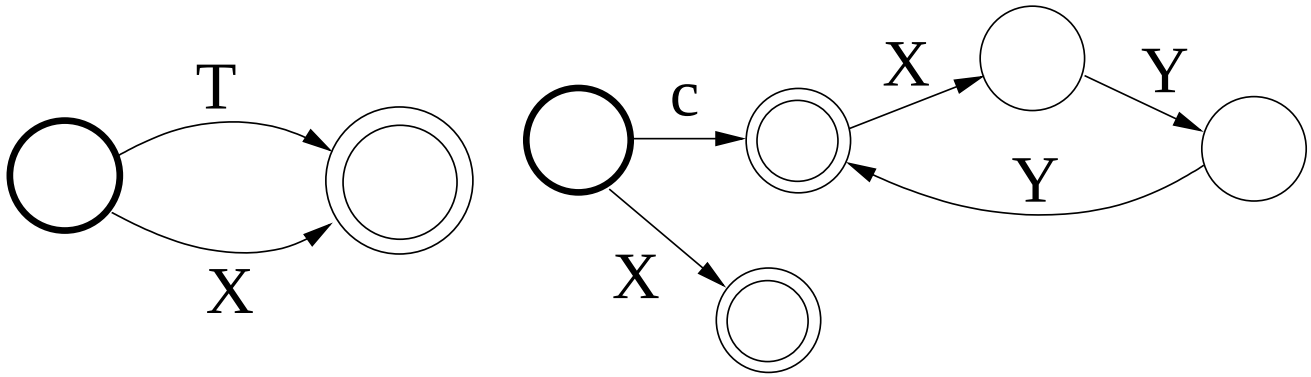




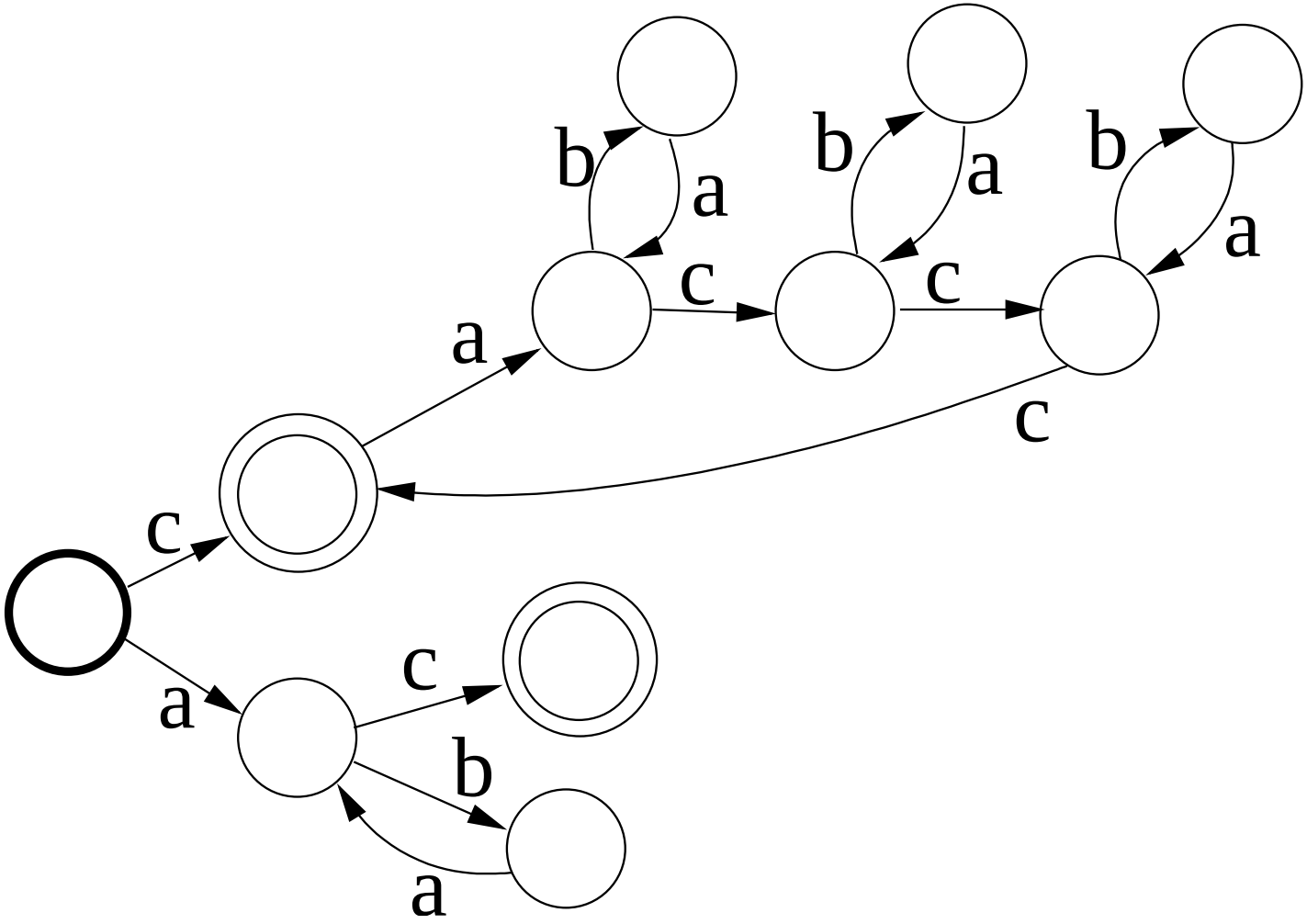
- Classical construction (Aho and Ullman 1973)
- Note: each component must contain either right-linear or left-linear rules (explained after a few slides)

Step 5: Simple automaton accepting the set of active nonterminals

Step 6: Expand automaton (on the fly)



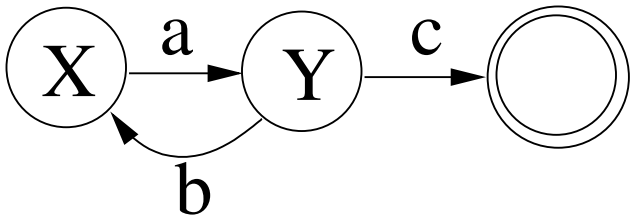
Result: Fully expanded automaton



Right-Linear Rules

- A right-linear rule is of the form
  nonterminal $\longrightarrow$ terminals [nonterminal]
- Straightforward transformation to
  state $\longrightarrow$ output symbols, state (or end state)
- Entry point depends and exit points are known

$$X \text{ -> } aY$$
$$Y \text{ -> } bX$$
$$Y \text{ -> } c$$

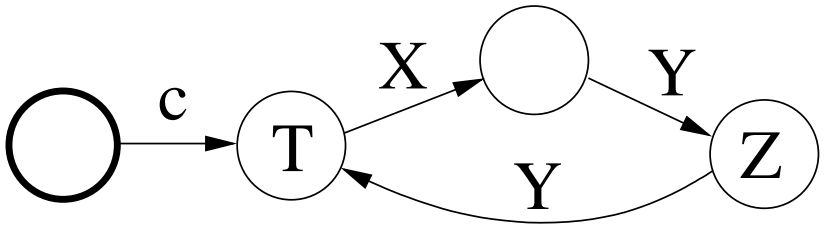


- Note: Nonterminals outside of the strongly connected component can be considered terminals

Left-Linear Rules

- A left-linear rule is of the form
  nonterminal $\longrightarrow$ [nonterminal] terminals
- Construct as before but read the right side of the rules backwards and reverse the arrows
- Entry points are known and exit point depends

T –> ZY
T –> c
Z –> TXY



- Note: Nonterminals X and Y are considered terminals here

Approximating Grammars by Strongly Regular Grammars

- Any grammar can be transformed into a strongly regular grammar approximatively
- Rules are repliced by right-linear rules
- The language will be a superset of the original
- Size of the resulting grammar is at most twice that of the input grammar

Transformation

For each nonterminal A in the strongly connected component:

Introduce a new nonterminal A' (interpreted as "after A").

Add the rule $A' \to \epsilon$. (ϵ is the empty string)

Replace each rule of the form

$A \to \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \ldots B_m \alpha_m$,

where α are terminals and "outsider" nonterminals (as before),

by the following set of rules:

$A \to \alpha_0 B_1$

$B'_1 \to \alpha_1 B_2$

$B'_2 \to \alpha_2 B_3$

$\ldots$

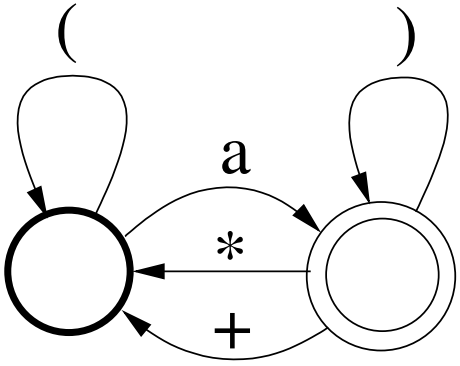
$B'_m \to \alpha_m A'$

Example: Arighmetic Expressions

E–>E+T
E–>T
T–>T*F
T–>F
F–>(E)
F–>a

E'–>.
T'–>.
F'–>.
E–>E
E'–>+T
T'–>E'
E–>T
T'–>E'

T–>T
T'–>*F
F'–>T'
T–>F
F'–>T'
F–>(E
E'–>)F'
F–>aF'



Handling Weights

- In speech recognition, grammar weights are combined with acoustic weights to rank hypotheses
- A robust grammar admits any hypothesis but with different weights
- Weights can be interpreted as probabilities $\Rightarrow$ the finite automaton becomes a hidden Markov model $\Rightarrow$ weights can be learned from the corpus
- Determining weights for the finite automaton based on the weights of the grammar is not trivial

Conclusions

- In most real-time applications, general context-free grammars are computationally too demanding
- Automata transformed from grammars are practical
- Some context-free languages need to be approximated with regular languages for the transformation
- The GRM library implements all that and it is used in many projects