1 introduction

On the boundary of regular and context-free languages
Overview of the following lectures:

- CFGs and FAs, definitions
- their role in NLP
- CFGs versus FAs
- Approximation of CFGs by FAs
- weighted and probabilistic (string) languages
- Approximation of PCFGs by PFAs
- KL divergence between probabilistic languages

2 definitions

Context-free grammars
Context-free grammar (CFG) consists of:

- (finite) set $\Sigma$ of terminals
- (finite) set of $N$ nonterminals
- start symbol $S \in N$
• (finite) set \( R \subset N \times (\Sigma \cup N)^* \) of rules

Rule notation: \( A \to \alpha \) where \( A \in N, \alpha \in (\Sigma \cup N)^* \)

'derives' relation \( \Rightarrow \)

'derives in zero or more steps' relation \( \Rightarrow^* \)

**Generated language** is set of all strings \( w \in \Sigma^* \) s.t. \( S \Rightarrow^* w \)

**Finite automata**

Finite automaton (FA) consists of:

• (finite) set \( Q \) of states
• (finite) set \( \Sigma \) of input symbols
• initial state \( q_0 \in Q \)
• set \( F \subseteq Q \) of final states
• set \( \Delta \) of transitions

Transitions are of forms \((q, a, q')\) or \((q, \varepsilon, q')\)

\((q, \varepsilon, q')\) is epsilon transition (\( \varepsilon \) is empty string)

FA defines (string) language (accepted language)

**Unique finite states**

For some applications we may assume start-of-string \( \uparrow \) and end-of-string \( \downarrow \) markers

So every string has the form:

\[ \uparrow \, a_1 \cdots a_n \downarrow \]

where \( a_i \in \Sigma \setminus \{\uparrow, \downarrow\} \), all \( i \)

With end-of-string markers we may assume without loss of generality that \( F = \{q_f\} \), where \( q_f \) is unique final state

**Deterministic FAs**

FA is deterministic if

• there are no epsilon transitions, and
• for each \( q \) and \( a \) there is at most one \( q' \) such that \((q, a, q') \in \Delta\)

For every FA there is deterministic FA (DFA) accepting same language
But size of DFA can be much larger (exponentially)
DFAs allow fast processing (each input symbol requires one step)

3 recognition

Recognition problem for general FAs
Is string \( a_1 \cdots a_n \) accepted by FA \( \mathcal{M} \)?
For now assume no epsilon transitions
Let \( T_i \) be set of states reachable after processing \( a_1 \cdots a_i \)
Let \( T_0 = \{q_0\} \)
From \( T_i \) we can compute \( T_{i+1} \) in a number of steps linear in size of FA
Acceptance if \( T_n \cap F \neq \emptyset \)
Hence recognition problem linear-time in size of FA and in length of input

Recognition problem for CFGs
There are CFGs allowing linear-time recognition (e.g. LR and LL grammars)
For general CFGs time complexity \( O(|R| \cdot n^3) \)
Intuition: make table of triples \((i, A, j)\) such that \( A \Rightarrow^* a_{i+1} \cdots a_j \)
From rule \( A \rightarrow a_j \) derive \((j-1, A, j)\)
From \((i, A, j)\) and \((j, B, k)\) and rule \( C \rightarrow A B \) derive \((i, C, k)\)
There are \( O(|R| \cdot n^3) \) such steps to consider
CKY algorithm (Cocke-Kasami-Younger)
What if rules not of form \( C \rightarrow AB \)? Then transform grammar

Generalising to input lattice
Instead of input string we can also consider input lattice
Such lattice is acyclic FA
With FA \( \mathcal{M}_1 \) and input lattice \( \mathcal{M}_2 \) recognition means "\( L(\mathcal{M}_1) \cap L(\mathcal{M}_2) \neq \emptyset \)?"
Similarly with CFG $\mathcal{G}$ and input lattice $\mathcal{M}$
Procedures work much as before
FA-recognition of lattice linear-time in size $n$ of lattice
CFG-recognition of lattice in $O(n^3)$ time
With cyclic input some extra bookkeeping is needed to avoid non-termination but same time complexity

4 natural language

Applications of FAs in NLP
FAs are widely used in natural language processing (NLP)
Useful properties:

- Favourable time complexity (linear)
- Closed under intersection
- (and under composition for finite-state transducers)

Example applications:

- tokenisation
- morphological parsing

Tokenization
Divide text into words, punctuation, numbers, layout characters, etc.
Is often first phase in analysis of written text
Many difficulties, e.g.

- period ‘.’ can be end of sentence, or part of abbreviation, or decimal point in number, . . .
- apostrophe can be quotative marker, or marker for contraction (e.g. we’ve)
Morphological parsing
Divide word into morphemes
unbelievably → un- believe -able -ly
Often also in combination with grammatical properties of morphemes
bebo → beber +V +PInd +1P +Sg
This requires finite-state transduction

Applications of CFGs in NLP

Syntax: the structure of utterances
How words are combined to make sentences
Nonterminals are categories
E.g. noun phrase (NP), verb phrase (VP)
Terminal strings derived from categories are constituents

Example CFG

```
S → NP VP                           I want a morning flight
NP → Pronoun                        I
    | ProperNoun                      Boston
    | Det Nominal                     a flight
Nominal → Nominal Noun              morning flight
    | Noun                            flights
VP → Verb                           do
    | Verb NP                         want a flight
    | Verb NP PP                      leave Boston at night
    | Verb PP                         leaving on Thursday
PP → Preposition NP                 from Boston
```

Example CFG (cont.)
Lexical rules:
Are CFGs adequate?

CFGs often not convenient for describing syntax of natural languages

Especially with coarse categories like NP, VP, . . .

It seems most syntactic phenomena in most languages can be described in principle by formalisms equivalent to CFGs

But not cross-serial dependencies in e.g. Swiss German (and Dutch)

Cross-serial dependencies in Swiss German

With word-by-word translation with accusative ACC and dative DAT

…das mer em Hans es huus hälfed aastricche

…that we Hans/DAT the house helped paint

(…that we helped Hans paint the house)

Extendable to:

…das mer d’chind em Hans es huus lönd hälfe aastricche

…that we the children/ACC Hans/DAT the house let help paint

(…that we let the children help Hans paint the house)

Cross-serial dependencies (cont.)

The construction can be extended further to a larger scheme:

- by taking a sequence of accusative NPs and the same number of occurrences of verbs (e.g. lönd; in infinitive form laa below) that take accusative NPs

- by taking a sequence of dative NPs and the same number of occurrences of verbs that take dative NPs

Jan säit das mer (d’chind)$^m$ (em Hans)$^m$ es huus haend wele (laa)$^n$ (hälfe)$^m$
aastricche
Jan says that we have wanted to \((\text{let the children})^n\) \((\text{help Hans})^m\) paint the house

This cannot be generated by a CFG

Applications of CFGs: parsing

Parse tree (produced as side-effect of recognition, e.g. by CKY)

Applications of CFGs: recognition

Output of speech recognizer (lattice):

Task: select plausible paths in lattice from initial to final state

Do recognition with CFG (time complexity \(n^3\))
or with FA (time complexity \(n\))

5 FAs versus CFGs

Regular languages are CFLs

Each regular language can be described by (right-linear) CFG
Proof by constructing a CFG ’simulating’ given FA
Let start symbol be initial state \(q_0\)
For each transition \((q, a, q')\) add rule \(q \rightarrow a q'\)
For each \(q \in F\) add rule \(q \rightarrow \varepsilon\)

**Strict inclusion**
Not regular is palindrome language generated by CFG:

\[
A \rightarrow a A a | b A b | a | b | \varepsilon
\]

Intuition: FAs have no memory to store first half of string, and then match it to second half in mirror image

Formal such proofs are often by pumping lemma (see below)

For palindrome language, proof also uses closure under intersection of regular languages

**Restrictions of regular languages**

**Pumping lemma (regular languages)**

Let \(L\) be an infinite regular language. Then there are strings \(x, y\) and \(z\) such that \(y \neq \varepsilon\) and \(xy^m z \in L\) for any \(m \geq 0\)

**Proof**: for a long enough string, a path through a fixed FA must go through a cycle (i.e. one state is visited for a second time; by the pigeonhole principle)

Take \(y\) to be the string of symbols on the cycle

We can go through the cycle any number \(m\) times

**CFLs and regular languages**

Given CFG, can we decide whether the generated language is regular?
The palindrome language is obviously non-regular

But how about:

\[
A \rightarrow a A a | b A b | a A b | b A a | a | b | \varepsilon
\]

Language is in fact \(\{a, b\}^*\) so regular

The question in general is undecidable!
"(linear) CFL is regular?" is undecidable

The question remains undecidable if we restrict ourselves to linear CFGs with center-marker #

I.e. every rule is of one of the following forms:

\[ A \rightarrow B \ a \text{ where } a \neq \# \]
\[ A \rightarrow a \ B \text{ where } a \neq \# \]
\[ A \rightarrow B \]
\[ A \rightarrow \# \]

This we will prove next, using Turing machines

**Sketch of proof**

Configuration of Turing machine (TM) represented by tape preceding tape head, state, and tape following tape head

\[ L(M) \] is language accepted by TM \( M \)

Consider accepting computation of TM consisting of configurations \( w_0, \ldots, w_{2k} \)

Encode this as

\[ w_0 \tilde{w} w_2 \tilde{w} \ldots \tilde{w}_{2k} \tilde{w}_{2k+1} \tilde{w} \ldots \tilde{w}_3 \tilde{w} \]

where \( \tilde{w} \) is mirror image of \( w \)

Note that the input string is encoded in \( w_0 \)

**Sketch of proof (cont.)**

Let \( VALC(M) \) be set of valid encodings of accepting computations of TM \( M \)

We use:

**Lemma 1**

It is undecidable whether language accepted by TM is infinite

(Hopcroft and Ullman: Introduction to Automata Theory, Languages, and Computation, 1979, p. 189)

**Proof of lemma (sketch)**

**Lemma 2**
VALC(M) is context-free iff $L(M)$ is finite

If $L(M)$ is finite, then VALC(M) is finite, and therefore context-free

If $L(M)$ is infinite, then VALC(M) is infinite, and we can use pumping lemma for CFLs (Ogden’s lemma)

Pumping a computation in VALC(M) would imply a repeating configuration, which is not possible

Hence VALC(M) is not context-free

Invalid computations generated by linear CFG

Now consider set INVALC(M) of invalid computations

In other words, complement of VALC(M) but $\#$ must still occur precisely once

In

$w_0\#w_2\#\ldots\#w_{2k}\#\tilde{w}_{2k+1}\#\ldots\#\tilde{w}_3\#\tilde{w}_1$

some ‘mistake’ is made

There is a finite number of possible mistakes, each describable by linear CFG

Union is again linear CFG

"(linear) CFL is regular?" is undecidable (cont.)

Take TM $M$

If $L(M)$ is finite, then VALC(M) is finite and therefore regular, and INVALC(M) is regular

If INVALC(M) is regular, then VALC(M) is regular and therefore context-free, and therefore $L(M)$ is finite

Because INVALC(M) is a linear CFG, if we can decide "(linear) CFL is regular?" then we can decide "$L(M)$ is finite"

But we cannot decide "$L(M)$ is finite?"

Hence we cannot decide "(linear) CFL is regular?"

Also: Nederhof and Satta, Computational Linguistics 37, 2011