TDDD14/TDDD85 Lecture 16: Chomsky hierarchy and a summary with extensions and outlooks

Jonas Wallgren

Abstract

This lecture presents the Chomsky hierarchy of languages and formalisms. It relates the various parts of the course to the hierarchy. It also presents some material outside the course range and gives some examples of further use of formal languages.

1 Introduction

Through the sequence of lectures we have found that some languages or formalisms (expressions, grammars) are more powerful than others. That feature allows the languages to be placed in a hierarchy of varying power.

2 The Chomsky hierarchy

To the general public Noam Chomsky maybe is most known for his political writings, but he is—among other things—a linguist that has tried to formalize (natural) languages. In 1956 he presented a hierarchy of languages which has become a standard classifiation scheme for formal languages.

2.1 The original levels

This table shows the original levels of the Chomsky hierarchy. The circled numbers indicate the order of description below.

Type	Language	Grammar	Automaton
0	⑧ recursively enumerable	⑨ unrestricted	⑦ TM
1	⑩ context-sensitive	⑪ context-sensitive	⑫ LBA
2	④ context-free	⑤ context-free	⑥ PDA
3	① regular	③ regular	② DFA

(Recursively enumerable is also called Turing-recognizable.)

①② The first part of the course (Victor's lectures) dealt with regular languages. A DFA (Deterministic Finite Automaton) consists of a number of

states connected by transitions. One state is the start state. There may be several accepting/final states. From a state, reading a symbol, there is a transition to exactly one state. In an NFA (N=nondeterministic) there may be several or no states in that situation. There may also be ε transitions with state changes without reading. An NFA can always be transformed to a DFA. A regular language is a language accepted by a DFA.

③ Every (original) hierarchy level has its corresponding grammar type. Regular grammars are mentioned in the tutorial problems but are not presented in the lectures. In a regular grammar either all rules are on the form $X \to wY$ or $X \to w$ (Then the grammar is right linear.) or all rules are on the form $X \to Yw$ or $X \to w$ (Then the grammar is left linear.).

Example 1. *A grammar for the regular expression* $0(10)^*$ *is*
$S \to 0A$
$A \to 10A|\varepsilon$
That right-linear grammar quite easily is transformed into an NFA accepting the regular expression:

	ε	0	1
$\to [S]$	$\{[0A]\}$	$\emptyset$	$\emptyset$
$[0A]$	$\emptyset$	$\{[A]\}$	$\emptyset$
$[A]$	$\{[\varepsilon], [10A]\}$	$\emptyset$	$\emptyset$
$F[\varepsilon]$	$\emptyset$	$\emptyset$	$\emptyset$
$[10A]$	$\emptyset$	$\emptyset$	$\{[10A]\}$

The start state represents the start symbol of the grammar. From a state representing a nonterminal there are ε *transitions to states representing the righthand sides of the grammar rules for that nonterminal. From other states there are transitions for reading symbols like the steps*
$[10A] \stackrel{1}{\to} [0A] \stackrel{0}{\to} [A]$.

Not mentioned in this hierarchy are regular expression. They are another way to describe regular languages.

④⑤ The next part of the course (Jonas' lectures) dealt with context-free languages. In a context-free grammar every rule has the form $A \to \alpha$, where α is a sequence of terminals and nonterminals. A context-free language is a language definable by a context-free grammar. A context-free grammar is in Chomsky normal form if all rules have the form $A \to BC$ or $A \to a$. A context-free grammar is in Greibach normal form if all rules have the form $A \to aB_1B_2B_3\ldots$. A problem in some applications of context-free grammars is left-recursion, when $A \to A\alpha|\beta$ occurs in a grammar. That can be rewritten to $A \to \beta A'$, $A' \to \alpha A'|\varepsilon$.

⑥ A PDA (PushDown Automaton) consists of a number of states connected by transitions plus a stack. One state is the start state. There may be several accepting/final states or none. A string can be accepted with

empty stack. From a state, reading a symbol or nothing, checking the stack top, there could be a possibility to go to one of a number of states and also to put symbols on the stack. Note that the PDA in its standard form is an NPDA, nondeterministic.

⑦ The third part of the course (Victor's lectures) dealt with Turing machines. A Turing machine (TM) consists of a number of states connected by transitions plus an infinite tape. One state is the start state. There may be several accepting/final states. From a state, reading a symbol from the tape, one symbol is written on the tape and there is one state to go to and a move of the tape head one step to the left or to the right. A TM may loop.

⑧ The languages that are recognized by a TM are called recursively enumerable or Turing-recognizable.

⑨ There is a grammar form related to this level in the Chomsky hierarchy. In an unrestricted grammar all rules have the form $\alpha \to \beta$, i.e. both left- and righthand sides can be sequences of terminals and nonterminals. An example:

Example 2.
$S \to ACaB$
$Ca \to aaC$
$CB \to DB|E$
$aD \to Da$
$AD \to AC$
$aE \to Ea$
$AE \to \varepsilon$

An example derivation:
$S \Rightarrow$
$ACaB \Rightarrow$
$AaaCB \Rightarrow$
$AaaDB \Rightarrow$
$AaDaB \Rightarrow$
$ADaaB \Rightarrow$
$ACaaB \Rightarrow$
$AaaCaaB \Rightarrow$
$AaaaaCB \Rightarrow$
$AaaaaE \Rightarrow$
$AaaaEa \Rightarrow$
$AaaEaa \Rightarrow$
$AaEaaa \Rightarrow$
$AEaaaa \Rightarrow$
$aaaa$

A and B can be seen as the (temporary) endpoints of the currently used part of a TM tape. C is moved to the right across each a and doubles them.

D is moved back to the beginning. C at the right end can remove B and become an E which moves to the left and deletes A. Since the number of a:s doubles for each passage of C from A to B the language is $\{a^{2^n} | n \geqslant 1\}$.

⑩⑪ Type 1 languages are not used that much in popular computer science or technology applications, but they deserve to be mentioned in this type of overview. In a context-sensitive grammar (CSG) all rules have the form $\alpha \to \beta$ with the restriction that the righthand side is as least as long as the lefthand side. In one normal form of a CSG all rules have the form $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$. The $A \to \beta$ part works like a CFG rule but the whole rule means that it can be applied only in the *context* $\alpha_1 \_ \alpha_2$. A context-sensitive language is a language definable by a CSG.

⑫ A linear-bounded automaton (LBA) is a TM with a limited tape—there is a left and a right endmarker on the tape.

2.2 Some intermediate levels

Development of theory and its applications have led more levels to appear in the hierarchy. Some of them have been treated in this course. This extended table shows their position in the hierarchy.

Type	Language	Grammar	Automaton
0	recursively enumerable	unrestricted	TM
	⑰ recursive		⑯ total TM
1	context-sensitive	context-sensitive	LBA
2	context-free	context-free	PDA
	⑭ DCFL	⑮ LR(1)	⑬ DPDA
3	regular	regular	DFA

(Recursive is also called decidable.)

⑬⑭ One could say that a DFA is such a simple formalism that trying to make it nondeterminstic doesn't help (It could be *easier* to define languages but the formal power doesn't increase.) and that a TM already is so powerful that it's impossible to reach any higher by nondeterminism. But in between there is a difference. A(n) (N)PDA is strictly more powerful than a DPDA which is the automaton for this level in the hierarchy. For each state there is for a given stack top either at most(!) one possibility for each input symbol to read it or a possibility to change state without reading. A language accepted by a PDA is context-free (⑥④). A language accepted by a DPDA is called deterministic context-free.

⑮ A language accepted by a DPDA can be described by an LR(1) grammar. Remember that LR parsing is deterministic.

First a comment on LR(0) (a little bit more on LR(0) below). When you construct an LR(0) parser you start with an automaton containing items in each state. An item is a grammar rule with a dot in its righthand side that represents how much of the righthand side that is read during

parsing. If a grammar contains $A \to aBC$, $B \to dE$ and $C \to fGH$ and the item $A \to a \cdot BC$ is found in a state, also the item $B \to \cdot dE$ should be in that state.

In LR(1) lookahed is used to increase the power of the formalism a bit. Every item is equipped with a lookahead set that indicates what could follow the lefthand nonterminal. In the example an item should be $B \to \cdot dE\{f\}$ since after B can follow f (from the C after B in the item with $\cdot BC$).

LR(1) has more formal power than LR(0) but increasing the amount of looakahead to e.g. LR(2) or LR(3) maybe makes grammar writing more practical for some languages but the formal power is not greater—the same class of languages is definable.

⑯⑰ A somewhat limited class of TM are those that always halt. They are called total Turing machnes. The corresponding languages are called recursive or decidable.

2.3 More levels

Even more levels in the hierarchy have been defined, investigated, and characterized. They get their applications mainly inside formal language theory. See the table at bottom of most formal language articles in English Wikipedia.

Some ideas don't result in new levels. What about a 2PDA, a PDA with two stacks? Doesn't the possibility to use two stacks increase the power? Yes it does! Very much, indeed. You can use the two stacks to simulate a TM tape: Move tape head to the left=Pop from stack 1 and push onto stack 2, Move tape head to the right=Pop from stack 2 and push onto stack 1. But it didn't result in a new level.

2.4 What about LR(0)?

LR(0) is mentioned above but still isn't in the hierarchy. We know that the language $\{0^n1^n|n \geqslant 1\}$ isn't regular but it is LR(0). The language a^* is regular but it isn't LR(0). It doesn't possess the prefix property: *aaa* is a string in the language and so is *aaaa*. Having read *aaa* you can't tell whether to accept or shift without using lookahead.

So neither of regular languages and LR(0) is a subset of the other one. LR(0) doesn't fit into the hierarchy.

3 Looking forward

After this course, into coming courses and work in computer science and technology, what will be most important, what will be used most?

One main area is interpreters and compilers. There you will use regular expressions or something equivalent to define the tokens. Often some grammar formalism between LR(0) and LR(1) is used.

Another main area is complexity and computability. There one important tool is to construct reductions between problems. For example, one could be interested in mapping reductions between computational problems where the mapping function can be computed by a Turing machine which does not use too much computational resources. In this manner one can e.g. define complexity classes and relate computational problems to each other, leading to the area of *complexity theory*.

(This doesn't mean that these are the most important concepts for the exam!)