CS:4330 Theory of Computation

Spring 2018

Context-Free Languages

Deterministic Context-Free Languages

Haniel Barbosa

The University of Iowa

Readings for this lecture

Chapter 2 of [Sipser 1996], 3rd edition. Section 2.4.

Deterministic CFLs vs CFLs

- ▷ Nondeterministic and deterministic PDAs are not equivalent in expressive power
- ▷ Deterministic Pushdown Automata (DPDAs) are strictly less expressive than PDAs
- ▷ Context-free languages not recognizable by a DPDA are called *Deterministic Context-Free Languages* (DCFLs)
- ▷ An example of application is parsing programming languages, as DCFLs are easier to parse than general CFLs

DPDA

Differently from DFAs, DPDAs may have ϵ-transitions

- ▷ At any moment, at most *one* transition is possible

For $\mathrm{a} \in \Sigma$, $\mathrm{b} \in \Gamma$, and $c \in \Gamma_\epsilon$

$$q_1 \xrightarrow{\mathrm{a}, \mathrm{b} \to \mathrm{c}} q_2$$

or

$$q_1 \xrightarrow{\epsilon, \mathrm{b} \to \mathrm{c}} q_2$$

or

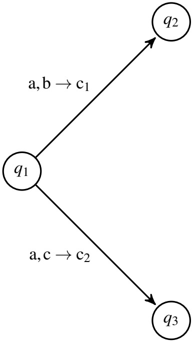
$$q_1 \xrightarrow{\mathrm{a}, \epsilon \to \mathrm{c}} q_2$$

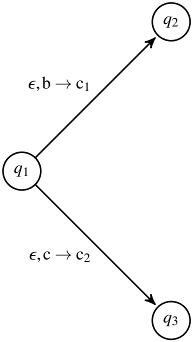
or

$$q_1 \xrightarrow{\epsilon, \epsilon \to \mathrm{c}} q_2$$

but only one of them is possible

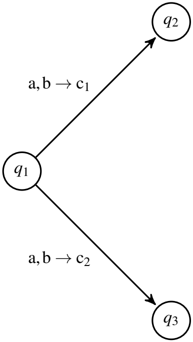
Allowed transitions

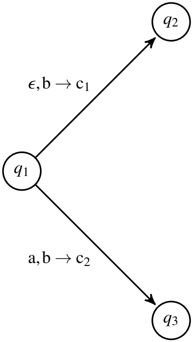




Deterministic choices

Disallowed transitions





Nondeterministic choices

Defintion of DPDA

A DPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, in which Q is its set of states, Σ its input alphabet, Γ its stack alphabet, δ its transition function, q_0 the start state, and $F \subseteq Q$ the set of accept states.

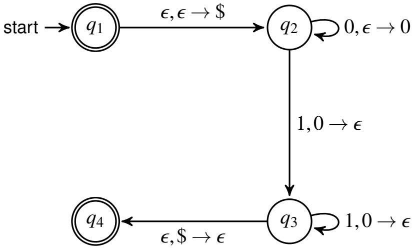
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to (Q \times \Gamma_\epsilon) \cup \varnothing$
- In which δ satisfies the following condition:
  - For every $q \in Q$, $\mathrm{a} \in \Sigma$ and $x \in \Gamma$, *exactly one* of the values

$$\delta(q, \mathrm{a}, x), \quad \delta(q, \mathrm{a}, \epsilon), \quad \delta(q, \epsilon, x), \quad \text{and} \quad \delta(q, \epsilon, \epsilon)$$

    is not $\varnothing$

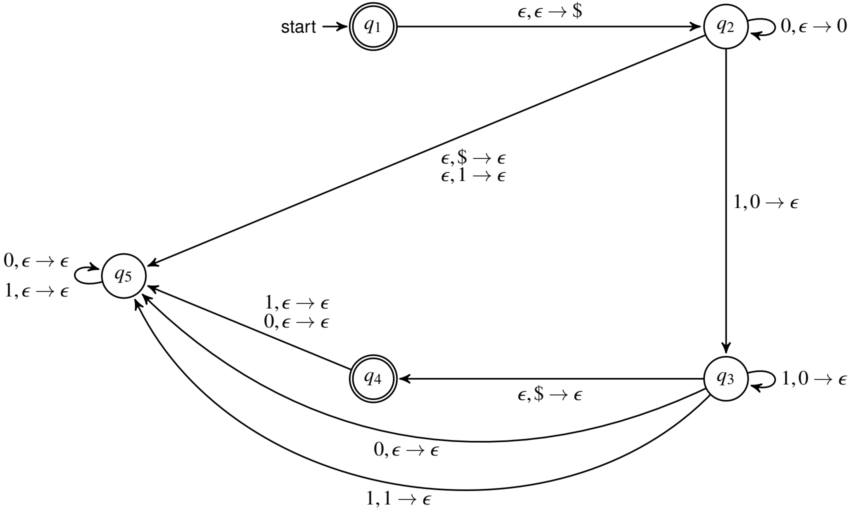
DPDA example

Consider the language $\{0^n 1^n \mid n \geq 0\}$



DPDA example

Consider the language $\{0^n 1^n \mid n \geq 0\}$



Deterministic Context-free Languages

Definition

A language L is *deterministic context-free* (DCFL) if there exists a DPDA that accepts it.

Example

The language $\{0^n 1^n \mid n \geq 0\}$ is deterministic context-free.

There are, however, CFLs that are *not* DCFLs. For example:

$$A = \{\mathrm{a}^i \mathrm{b}^j \mathrm{c}^k \mid i,j,k \geq 0 \land i \neq j \lor j \neq k\}$$

is a CFL but not a DCFL. How can we show this?

CFL but not DCFL

Theorem

The class of DCFLs is closed under complementation.

- ▷ We use the above theorem and the fact that CFLs are *not* closed under complementation, i.e. the complement of a CFL may *not* be a CFL, to show that $A = \{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \mid i,j,k \geq 0 \wedge i \neq j \vee j \neq k\}$ is not a DCFL.

- ▷ If A were a DCFL, $\bar{A}$ would also be a DCFL, which means that $\bar{A}$ would also be a CFL.

CFL but not DCFL

Theorem

The class of DCFLs is closed under complementation.

- ▷ We use the above theorem and the fact that CFLs are *not* closed under complementation, i.e. the complement of a CFL may *not* be a CFL, to show that $A = \{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \mid i,j,k \geq 0 \wedge i \neq j \vee j \neq k\}$ is not a DCFL.

- ▷ If A were a DCFL, $\bar{A}$ would also be a DCFL, which means that $\bar{A}$ would also be a CFL.

- ▷ $\bar{A} = \{a^n b^n c^n \mid n \geq 0\}$ is *not* a CFL, therefore A is an example of a CFL that is *not* a DCFL.

DCFL in practice

- ▷ LR(k) grammar: it generates a DCFL, with ‘L’ meaning “the input is read from Left to right” and ‘R(k)’ meaning “Right most derivation decided by the first k input symbols”

- ▷ Backtracking and guessing is avoided by the parser being allowed to *lookahead* at k input symbols before deciding how to parse earlier symbols.

- ▷ Most programming languages are specified by LR(k) grammars, with $k \leq 2$

- ▷ LR(k) parser: an efficient algorithm to *decide*, in linear time, if a word is in $\mathscr{L}(G)$, in which G is an LR(k) grammar.

Example: an LR(1) grammar

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T \times F \mid F \\
F &\rightarrow (E) \mid \mathrm{a}
\end{aligned}
$$

Rightmost derivation

$$
\begin{aligned}
E &\Rightarrow E + T \\
&\Rightarrow E + F \\
&\Rightarrow E + \mathrm{a} \\
&\Rightarrow T + \mathrm{a} \\
&\Rightarrow T \times F + \mathrm{a} \\
&\Rightarrow T \times \mathrm{a} + \mathrm{a} \\
&\Rightarrow F \times \mathrm{a} + \mathrm{a} \\
&\Rightarrow \mathrm{a} \times \mathrm{a} + \mathrm{a}
\end{aligned}
$$

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$a \times a + a\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$a \times a + a\$$	*shift*
a	$\times a + a\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$a \times a + a\$$	*shift*
a	$\times a + a\$$	*reduce* by $F \to a$
F	$\times a + a\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$a \times a + a\$$	*shift*
a	$\times a + a\$$	*reduce* by $F \to a$
F	$\times a + a\$$	*reduce* by $T \to F$
T	$\times a + a\$$	*shift*
$T\times$	$a + a\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T\times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T\times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T \times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	*reduce* by $T \to T \times F$
T	$+\mathrm{a}\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T\times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	*reduce* by $T \to T \times F$
T	$+\mathrm{a}\$$	*reduce* by $E \to T$
E	$+\mathrm{a}\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a}\times\mathrm{a}+\mathrm{a}\$$	*shift*
a	$\times\mathrm{a}+\mathrm{a}\$$	*reduce* by $F\to\mathrm{a}$
F	$\times\mathrm{a}+\mathrm{a}\$$	*reduce* by $T\to F$
T	$\times\mathrm{a}+\mathrm{a}\$$	*shift*
$T\times$	$\mathrm{a}+\mathrm{a}\$$	*shift*
$T\times\mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F\to\mathrm{a}$
$T\times F$	$+\mathrm{a}\$$	*reduce* by $T\to T\times F$
T	$+\mathrm{a}\$$	*reduce* by $E\to T$
E	$+\mathrm{a}\$$	*shift*
$E+$	$\mathrm{a}\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T \times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	*reduce* by $T \to T \times F$
T	$+\mathrm{a}\$$	*reduce* by $E \to T$
E	$+\mathrm{a}\$$	*shift*
$E+$	$\mathrm{a}\$$	*shift*
$E + \mathrm{a}$	$\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T\times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	*reduce* by $T \to T \times F$
T	$+\mathrm{a}\$$	*reduce* by $E \to T$
E	$+\mathrm{a}\$$	*shift*
$E+$	$\mathrm{a}\$$	*shift*
$E + \mathrm{a}$	$\$$	*reduce* by $F \to \mathrm{a}$
$E + \mathrm{F}$	$\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$a \times a + a\$$	*shift*
a	$\times a + a\$$	*reduce* by $F \to a$
F	$\times a + a\$$	*reduce* by $T \to F$
T	$\times a + a\$$	*shift*
$T\times$	$a + a\$$	*shift*
$T \times a$	$+a\$$	*reduce* by $F \to a$
$T \times F$	$+a\$$	*reduce* by $T \to T \times F$
T	$+a\$$	*reduce* by $E \to T$
E	$+a\$$	*shift*
$E+$	$a\$$	*shift*
$E + a$	$\$$	*reduce* by $F \to a$
$E + \mathrm{F}$	$\$$	*reduce* by $T \to F$
$E + \mathrm{T}$	$\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T\times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	*reduce* by $T \to T \times F$
T	$+\mathrm{a}\$$	*reduce* by $E \to T$
E	$+\mathrm{a}\$$	*shift*
$E+$	$\mathrm{a}\$$	*shift*
$E + \mathrm{a}$	$\$$	*reduce* by $F \to \mathrm{a}$
$E + \mathrm{F}$	$\$$	*reduce* by $T \to F$
$E + \mathrm{T}$	$\$$	*reduce* by $E \to E + T$
E	$\$$	

LR(k) parser

A stack is used to store the derivation steps backward. Two actions on stack:

1. *shift*: move a symbol from input to the stack;
2. *reduce*: replace the *rhs* of a rule in the top of the stack by its *lhs*

The actions of an LR(1) parser for the previous derivation

stack	input	action
	$\mathrm{a} \times \mathrm{a} + \mathrm{a}\$$	*shift*
a	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
F	$\times \mathrm{a} + \mathrm{a}\$$	*reduce* by $T \to F$
T	$\times \mathrm{a} + \mathrm{a}\$$	*shift*
$T \times$	$\mathrm{a} + \mathrm{a}\$$	*shift*
$T \times \mathrm{a}$	$+\mathrm{a}\$$	*reduce* by $F \to \mathrm{a}$
$T \times F$	$+\mathrm{a}\$$	*reduce* by $T \to T \times F$
T	$+\mathrm{a}\$$	*reduce* by $E \to T$
E	$+\mathrm{a}\$$	*shift*
$E+$	$\mathrm{a}\$$	*shift*
$E + \mathrm{a}$	$\$$	*reduce* by $F \to \mathrm{a}$
$E + \mathrm{F}$	$\$$	*reduce* by $T \to F$
$E + \mathrm{T}$	$\$$	*reduce* by $E \to E + T$
E	$\$$	*accept*

A hierachy of languages