Deterministic PDA’s

Deepak D’Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

25 October 2013
Outline

1. Deterministic PDA’s
2. Closure properties of DCFL’s
3. Complementing DPDA’s
A PDA with restrictions that:
- **At most** one move possible in any configuration.
  - For any state $p$, $a \in A$, and $X \in \Gamma$: at most one move of the form $(p, a, X) \rightarrow (q, \gamma)$ or $(p, \epsilon, X) \rightarrow (q, \gamma)$.
  - Effectively, a DPDA must see the current state, and top of stack, and decide whether to make an $\epsilon$-move or read input and move.
- Accepts by final state.
- We need a right-end marker “⊣” for the input.
Example DPDA for \( \{a^n b^n \mid n \geq 0\} \)
Example DPDA for \( \{ a^n b^n \mid n \geq 0 \} \)

\[
\begin{align*}
(s, a, \perp) & \rightarrow (p, A \perp) \\
(p, a, A) & \rightarrow (p, AA) \\
(p, b, A) & \rightarrow (q, \varepsilon) \\
(q, b, A) & \rightarrow (q, \varepsilon) \\
(q, \perp, \perp) & \rightarrow (t, \perp) \\
(s, \perp, \perp) & \rightarrow (t, \perp).
\end{align*}
\]

Class of languages accepted by DPDA’s are called DCFL’s.
Closure Properties of DCFL’s

- All languages over A
- CFL
- DCFL
- Regular

- $a^n b^n c^n$
- $a^n b^n c^n$
- $a^n b^n$

Closed? Complementation
Closure Properties of DCFL’s

- All languages over $A$
- Regular
- DCFL
- CFL

<table>
<thead>
<tr>
<th>Language</th>
<th>Closed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complementation</td>
<td>√</td>
</tr>
<tr>
<td>Union</td>
<td></td>
</tr>
</tbody>
</table>
## Closure Properties of DCFL’s

<table>
<thead>
<tr>
<th>Operation</th>
<th>Closed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complementation</td>
<td>√</td>
</tr>
<tr>
<td>Union</td>
<td>X</td>
</tr>
<tr>
<td>Intersection</td>
<td></td>
</tr>
</tbody>
</table>

All languages over $A$ are represented in the diagram. DCFL is a subset of CFL, which is itself a subset of Regular languages.

- **DCFL**: $a^n b^n c^n$ and $a^n b^n c^n$ are examples of DCFL languages.
- **CFL**: $a^n b^n c^n$ is a subset of CFL.
- **Regular**: $a^n b^n$ is a subset of Regular languages.

The diagram illustrates the closure properties of DCFL's with respect to Complementation and Union, and the lack of closure under Intersection.

### Examples
- **DCFL**: $L = \{a^n b^n c^n \mid n \geq 0\}$
- **CFL**: $L = \{a^n b^n c^n \mid 2 \leq n \leq 3\}$
- **Regular**: $L = \{a^n b^n \mid n \geq 0\}$
Closure Properties of DCFL’s

<table>
<thead>
<tr>
<th></th>
<th>Closed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complementation</td>
<td>√</td>
</tr>
<tr>
<td>Union</td>
<td>X</td>
</tr>
<tr>
<td>Intersection</td>
<td>X</td>
</tr>
</tbody>
</table>

All languages over A

DCFL

CFL

Regular

$\{a^n b^n c^n \mid n \geq 0\}$

$\{a^n b^n \mid n \geq 0\}$

Deterministic PDA’s

Closure properties of DCFL’s

Complementing DPDA’s
DCFL’s are closed under complementation

**Theorem (Closure under complementation)**

*The class of languages definable by Deterministic Pushdown Automata (i.e. DCFL’s) is closed under complementation.*
Problem with complementing a DPDA

Try flipping final and non-final states.
Problems?

Loops denote an infinite sequence of $\epsilon$-moves.
Desirable form of DPDA

Goal is to convert the DPDA into the form:

That is, always reads its input and reaches a final/reject sink state. Then we can make \( r' \) the unique accepting state, to accept the complement of \( M \).
Construction - Step 1

Let $M = (Q, A, \Gamma, s, \delta, \bot, F)$ be given DPDA. First construct DPDA $M'$ which

- Does not get stuck due to no transition or stack empty.
- Has only “sink” final states.
Construction - Step 1

Define $M' = (Q \cup Q' \cup \{s_1, r, r'\}, A, \Gamma \cup \{\bot\}, s_1, \delta', \bot, F')$ where

- $Q' = \{q' \mid q \in Q\}$ and $F' = \{f' \mid f \in F\}$.
- $\delta'$ is obtained from $\delta$ as follows:
  - Assume $M$ is “complete” (does not get stuck due to no transition). (If not, add a dead state and add transitions to it.)
  - Make sure $M'$ never empties its stack, keep track of whether we have seen end of input (primed states) or not (unprimed states):

- $(s_1, \epsilon, \bot) \rightarrow (s, \bot \bot)$
- $(p, \epsilon, \bot) \rightarrow (r, \bot)$ (if $p \in Q$)
- $(p', \epsilon, \bot) \rightarrow (r', \bot)$ (if $p' \notin F'$)
- $(p, \dashv, X) \rightarrow (q', \gamma)$ if $(p, \dashv, X) \rightarrow (q, \gamma) \in \delta$.
- $(p', \epsilon, X) \rightarrow (q', \gamma)$ if $(p, \epsilon, X) \rightarrow (q, \gamma) \in \delta$.
- $(r, a, X) \rightarrow (r, X)$
- $(r, \dashv, X) \rightarrow (r', X)$
- $(r', \epsilon, X) \rightarrow (r', X)$
- $(f', \epsilon, X) \rightarrow (f', X)$ (if $f \in F$) Also drop trans. going from $f'$.
After Step 1

DPDA $M'$ only has the following kinds of behaviours now:

Loops denote an infinite sequence of $\epsilon$-moves.
Construction - Step 2

A spurious transition in $M'$ is a transition of the form $(p, \epsilon, X) \rightarrow (q, \gamma)$ such that

$$(p, \epsilon, X) \Rightarrow^* (p, \epsilon, X\alpha)$$

for some stack contents $\alpha$.

Identify spurious transitions in $M'$ and remove them:
If $(p, \epsilon, X) \rightarrow (q, \gamma)$ is a spurious transition, replace it with

$$(p, \epsilon, X) \rightarrow (r, X) \quad \text{if } p \in Q$$

$$(p, \epsilon, X) \rightarrow (r', X) \quad \text{if } p \in Q' - F'.$$
Correctness

Argue that:
- Deleting a spurious transition (starting from a non-$F'$-final state) does not change the language of $M'$.
- All infinite loops use a spurious transition.
  - Look at graph of stack height along infinite loop, and argue that there are infinitely many future minimas.

  ![Graph showing stack height over infinite loop](image)

- Further look at transitions applied at these points and observe that one must repeat.
- Thus replacing spurious transitions as described earlier will remove the remaining undesirable loops from $M''$'s behaviours.
Complementing

- Resulting $M''$ has the desired behaviour (every run either reaches a final sink state or the reject sink state $r'$.)

- Now make $r'$ unique final state to complement the language of $M$. 
Detecting spurious transitions

Question: How can we effectively detect spurious transitions?
Detecting spurious transitions

Question: How can we effectively detect spurious transitions?
Use algorithm for pushdown reachability.