Greibach Normal Form (GNF)

- Sheila Greibach’s normal form (GNF) for a CFG is one where EVERY production has the form:
  \[ A \rightarrow a\alpha \] , where \( a \in T \) and \( \alpha \in V^* \)
- We saw this construction previously

CFG to PDA - 1

- Given a CFG \( G = (V, T, S, P) \), in GNF, a PDA for \( L(G) \) is \( ([q], T, V, \delta, q, S, \emptyset) \) where
  - \( \delta(q, a, A) = \{(q, \gamma) : (A \rightarrow a\gamma) \in P \} \)
  - (see Theorem 5.3)
Example: $aabb$ is derived by:

- $S \rightarrow aS$
- $S \rightarrow aAA$
- $A \rightarrow bSSB$
- $A \rightarrow b$
- $B \rightarrow aSA$

- $\delta(q, a, S) = \{(q, S), (q, AA)\}$
- $\delta(q, b, A) = \{(q, \varepsilon), (q, SSB)\}$
- $\delta(q, a, B) = \{(q, SA)\}$
- $(q, aabb, S) |\rightarrow (q, abb, S) \text{ or } (q, abb, AA)$
- $(q, bbb, S) \text{ or } (q, bbb, AA) \text{ or } \text{fail}$
- $(q, b, A) \text{ or } (q, b, SSB) \text{ or } \text{fail}$
- $(q, \varepsilon, \varepsilon) \text{ or } (q, \varepsilon, SSB) \text{ or } \text{fail}$

If the PDA were deterministic, this is the basis of a shift-reduce parser.

---

Given a PDA, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ build a GNF grammar as follows:

- $S \rightarrow [q_0, Z_0, q]$ for each $q$ in $Q$
- for each possible choice of transition $\delta(q, a, A) = (q_1, B_1 B_2 \ldots B_m)$, choose any set of $q_j$ in $Q$ and

- Create the production rule:
- $[q, A, q_{m+1}] \rightarrow a [q_1, B_1, q_2] [q_2, B_2, q_3] \ldots [q_{m-1}, B_{m-1}, q_m][q_m, B_m, q_{m+1}]$
- For a transition $\delta(q, a, A) = (q_1, \varepsilon)$, you get the rule $[q, A, q_1] \rightarrow a$
- The textbook goes through Example 5.3
It is clear when a PDA is deterministic and not all CFL’s have a deterministic PDA (DPDA).

It is not clear when a grammar generates a language that has a DPDA!

The “compiler grammars” in class LR(0) do give CFL’s with DPDA’s.

The more general LR(1) grammars have a deterministic machine that slightly extends a DPDA by allowing a lookahead.

If we agree to put an extra symbol at the end of the input ($), this recognizer can be simulated with a DPDA.

The complement of a DPDA language is a DPDA language.

Results from Chapter 7
Theorem 7.1

Equivalence of 1-way infinite and 2-way infinite tape machines

Simulations

A two way infinite tape machine can simulate a one-way infinite machine by

– initially putting a maker to the left of the first tape symbol [using a couple of extra states and transitions] and then
– not allowing the simulation to move past that maker
One-way infinite and two-way infinite tapes (2)

- The simulation of a 2-way infinite tape machine by a 1-way infinite tape machine uses 2 tracks.
- The upper track simulates the right half of the tape.
- The lower track has a marker and the rest of it is used to simulate the left half of the tape.

Add D to the state when using the lower track

- Picture:

  **Simulation:**

Add U to the state when using the lower track

- Picture:

  **Simulation:**

Details

- Special transitions are needed to flip between \((q, U)\) and \((q, D)\) when the read head recognizes the marker "$" on the lower track.
- Whether you switch between \(U\) and \(D\) depends on whether the 2-way infinite tape head moves left or right.
- The direction of moves on the lower track are opposite to the moves on the left half of the 2-way infinite track.
Example: 2-way tape

- The head moves from left to right past the original start position

\[ Y_8 \ Y_7 \ Y_6 \ Y_5 \ Y_4 \ Y_3 \ Y_2 \ Y_1 \ X_1 \ X_2 \ X_3 \ X_4 \ X_5 \ X_6 \ X_7 \ X_8 \ X_9 \ X \]

\[ q \]

\[ q' \]

\[ q'' \]

Example: 1-way tape

- The simulation reverses direction

\[ X_1 \ X_2 \ X_3 \ X_4 \ X_5 \ X_6 \ X_7 \ X_8 \ X_9 \ X_{10} \ X_{11} \cdots \]

\[ Y_1 \ Y_2 \ Y_3 \ Y_4 \ Y_5 \ Y_6 \ Y_7 \ Y_8 \ Y_9 \ Y_{10} \cdots \]

\[ q, D \]

\[ X_1' \ X_2' \ X_3' \ X_4' \ X_5' \ X_6' \ X_7' \ X_8' \ X_9' \ X_{10} \]

\[ Y_1' \ Y_2' \ Y_3' \ Y_4' \ Y_5' \ Y_6' \ Y_7' \ Y_8' \ Y_9' \]

\[ q', D \]

\[ X_1'' \ X_2'' \ X_3'' \ X_4'' \ X_5'' \ X_6'' \ X_7'' \ X_8'' \ X_9'' \]

\[ Y_1'' \ Y_2'' \ Y_3'' \ Y_4'' \ Y_5'' \ Y_6'' \ Y_7'' \ Y_8'' \ Y_9'' \]

\[ q'', U \]

Summary

- There need to be four types of special movements at the left end of the simulating tape
- The transitions are given in the textbook
- If the head is moving left, you switch from \((q, U)\) to \((q', D)\)
- Otherwise the 2-way infinite tape head switches direction and the simulator stays in the \(U\) or \(D\) states

Move left past original position

- The head moves from right to left past the original start position

\[ Y_8 \ Y_7 \ Y_6 \ Y_5 \ Y_4 \ Y_3 \ Y_2 \ Y_1 \ X_1 \ X_2 \ X_3 \ X_4 \ X_5 \ X_6 \ X_7 \ X_8 \ X_9 \ X \]

\[ q \]

\[ q' \]

\[ q'' \]

CS 573 © L. Lander, 2000 Class 20 - 22

CS 573 © L. Lander, 2000 Class 20 - 23

CS 573 © L. Lander, 2000 Class 20 - 24

CS 573 © L. Lander, 2000 Class 20 - 25
Go from U to D

- The simulation reverses direction

\[ X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10} X_{11} \ldots \]
\[ Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8 Y_9 Y_{10} \ldots \]

\[ q, U \]

\[ X'_1 X'_2 X'_3 X'_4 X'_5 X'_6 X'_7 X'_8 X'_9 X'_{10} X'_{11} \ldots \]
\[ Y'_1 Y'_2 Y'_3 Y'_4 Y'_5 Y'_6 Y'_7 Y'_8 Y'_9 Y'_{10} \ldots \]

\[ q', U \]

\[ X''_1 X''_2 X''_3 X''_4 X''_5 X''_6 X''_7 X''_8 X''_9 X''_{10} \]
\[ Y''_1 Y''_2 Y''_3 Y''_4 Y''_5 Y''_6 Y''_7 Y''_8 Y''_9 Y''_{10} \]

Stay on D

- The simulation reverses direction

\[ X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10} X_{11} \ldots \]
\[ Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7 Y_8 Y_9 Y_{10} \ldots \]

\[ q, D \]

\[ X'_1 X'_2 X'_3 X'_4 X'_5 X'_6 X'_7 X'_8 X'_9 X'_{10} X'_{11} \ldots \]
\[ Y'_1 Y'_2 Y'_3 Y'_4 Y'_5 Y'_6 Y'_7 Y'_8 Y'_9 Y'_{10} \ldots \]

\[ q', D \]

\[ X''_1 X''_2 X''_3 X''_4 X''_5 X''_6 X''_7 X''_8 X''_9 X''_{10} \]
\[ Y''_1 Y''_2 Y''_3 Y''_4 Y''_5 Y''_6 Y''_7 Y''_8 Y''_9 Y''_{10} \]

turn round at original position, starting on left

- The head moves right to original position and then moves left again

\[ Y_8 Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y'_1 X'_1 X'_2 X'_3 X'_4 X'_5 X'_6 X'_7 X'_8 X'_9 X \]
\[ q \]

\[ Y'_8 Y'_7 Y'_6 Y'_5 Y'_4 Y'_3 Y'_2 Y''_1 X''_1 X''_2 X''_3 X''_4 X''_5 X''_6 X''_7 X''_8 X''_9 X \]
\[ q' \]

\[ Y''_8 Y''_7 Y''_6 Y''_5 Y''_4 Y''_3 Y''_2 Y'''_1 X'''_1 X'''_2 X'''_3 X'''_4 X'''_5 X'''_6 X'''_7 X'''_8 X'''_9 X \]
\[ q'' \]

turn round at original position, starting on right

- The head moves left to original position and then moves right

\[ Y_8 Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X \]
\[ q \]

\[ Y'_8 Y'_7 Y'_6 Y'_5 Y'_4 Y'_3 Y'_2 Y''_1 X''_1 X''_2 X''_3 X''_4 X''_5 X''_6 X''_7 X''_8 X''_9 X \]
\[ q' \]

\[ Y''_8 Y''_7 Y''_6 Y''_5 Y''_4 Y''_3 Y''_2 Y'''_1 X'''_1 X'''_2 X'''_3 X'''_4 X'''_5 X'''_6 X'''_7 X'''_8 X'''_9 X \]
\[ q'' \]
Results from Chapter 7
Theorem 7.2

Equivalence of multi-tape and one tape machines

Simulating multitape machines

- We have already defined multitape machines (assume all tapes 2-way)
- They can be simulated by a single-tape, many track machine
- A multitape machine has many heads
- They can move independently left or right or remain stationary, so long as one moves
- We have to simulate this design with only ONE tape head

Simulate $n$ tapes with $2n$ tracks

- The $X$'s represent head positions
Sweep left and right

- Start at the right-most X, sweep left across until the \( n \)-th X is seen
- As you sweep across, record the \( n \) tape symbols above the X’s
- Once all \( n \) symbols are recorded, find the multitape move (new state, \( n \) new symbols and \( n \) directions to move)
- Now move back right, writing the new tape symbols and moving the X’s left or right to simulate the moves of the separate heads

Results from Chapter 7
Lemma 7.3

Simulation using a 2-stack machine

Two stack machines

- We can have a read-only input tape if we copy the input onto a second tape and then run the TM on the second tape
- Hence we can design a TM to have a read-only input tape and a second read-write tape
- Now a read-write tape can be cut in two and used as two stacks:
Initial situation

Tape:

Stacks:

Results from Chapter 7
Lemma 7.4, Theorem 7.9

Simulation using counter machines

Count with blanks

A 2-step encoding process shows that the content of the stacks using a multisymbol tape-alphabet can be simulated by

• A 2-counter machine where, there is an input tape and two stacks that have a Z at the bottom and only blanks above that

• By analyzing the number of blanks, the content of a stack can be simulated

Interesting parts of the encoding

• If the tape alphabet had symbols $Z_1, Z_2, \ldots, Z_{k-1}$ then think of those symbols as digits for the numbers with radix $k$

• A stack configuration is then just a number

• For example $k = 8$, then $Z_1, Z_2, \ldots, Z_7$ can be identified with $1, 2, \ldots, 7$ and a particular stack is just some octal number such as 6271773114
Four counter stacks

- With 2 stacks, you can work with the representation of $Z_{i_1}Z_{i_2} \ldots Z_{i_m}$ as $j = i_m + k_i_{m-1} + k^2i_{m-2} + \ldots + k^{m-1}i_1$, which is represented on one stack as $B^jZ$

A lot of counting

- To determine $Z_{im}$ copy from one stack to the other--pop $k$ $B$'s and push ONE $B$ to the other stack--moving through $k$ states
- you reach “$Z$” in the state $i_m$
- Use $Z_{im}$ to make move $j-i_m$

The move

- Assume that from $Z_{im}$, the new tape symbol is $Z_{is}$ on one of the stack pairs
- Move blanks from left to right, for 1 blank on left push $k$ blanks on right
- At end push $i_s$ extra blanks on stack
- The new symbol may go to left or right stack pair

One number can represent four numbers

- The coding so far will give a 4-counter machine--a 2-stack machine is simulated by a 4-counter machine
- Another step simulates a 4-counter machine by a 2-counter machine
- Here the counts $i, j, k, m$ on the 4 counter stacks is simulated by the number $2^i3^j5^k7^m$ on one stack (more details are sketched in the textbook)