Proving a grammar generates a language

Prove all strings generated by the grammar are in the language

Prove all strings in the language are generated by the grammar
Example

Language = \{w \in \{a,b\}^* \mid w \text{ has equal numbers of } a\'s \text{ and } b\'s\}
Grammar: \[ S \rightarrow SS \mid aSb \mid bSa \mid \epsilon \]

Prove all strings generated by the grammar are in the language
- By induction on the length of the derivation of \(w\) in \(L(G)\)

Example

Language = \{w \in \{a,b\}^* \mid w \text{ has equal numbers of } a\'s \text{ and } b\'s\}
Grammar: \[ S \rightarrow SS \mid aSb \mid bSa \mid \epsilon \]

Prove all strings in the language are generated by the grammar
- By induction on the length of \(w \in L\)
Don’t use entire power of CFG

Instead, use what’s called LR(k) subset

- L means “read Left-to-right”
- R means “Rightmost derivation”
- k is number of symbols that must be looked ahead before deciding how to construct parse tree (usually is 1)

Modifications of CFG

BNF (Backus-Naur Form)
- Many variations
- | for or
- Optional parts (inside [])
- Kleene star (inside {})
- Quoted-characters or bold for terminals

\[
\text{try_statement} ::= \\
\text{try} \text{ statement} \\
\{ \text{catch } "(\text{ parameter })" \text{ statement } \} \\
[ \text{finally} \text{ statement } ]
\]

\[
\text{switch_statement} ::= \\
\text{switch } "(" \text{ expression } ")" \ "\{" \\
\{ \text{case expression } ":\" \\
| \text{default } ":;\" \\
| \text{statement } \} \\
")\"
\]
Chomsky Normal Form (CNF)

Every production looks like:
- A → BC
  - S never appears on RHS
- A → x
  - Can have S → ε

Every CFG can be rewritten in CNF

Why do we care?
- Easier to build PushDown Automata (PDA) for constrained CNF than for any arbitrary CFG
- Gives an algorithm to tell whether a CFG generates a particular string

- Gives an algorithm to tell whether a CFG generates the empty language
Example conversion to CNF

S → XSX | aY
X → Y | S
Y → b | ε

How to convert to CNF

4 things we need to do:
What is a pushdown automata?

A Finite-state automata augmented with a stack

Stack:
- Holds stack symbols (stack alphabet distinct from input alphabet)
- Can pop symbol from the stack
  - popping empty stack causes this computation to not accept
  - Can only retrieve topmost symbol
- Can push a stack symbol
  - Always goes on top
- No way to explicitly test whether stack is empty
  - But we’ve got a trick to be able to tell!

State diagram
- labels become: a, b→c
  - means
    - reading a from input
    - and top of stack is b
    - pop b
    - push c
Example

Language = \{0^n1^n | n \geq 0\}

Example

Language = \{w \in \{0, 1\}^* | w \text{ has equal numbers of } 0\text{'s and } 1\text{'s}\}
Example

Language = \{w#w^R | w \in \{0, 1\}^*\}
Example

Language = w \textit{not} a palindrome (over \{0, 1\}*)

Example

Language: L\{xy \mid |x|=|y|, x \neq y, x, y \in \{0, 1\}^*\}