In Computer Science, grammars are most often thought of as defining rules for accepting strings in a language.
- I.e., use in programming languages

However, a grammar can be used to generate strings in a language, as well as accept strings in a language.

The way grammars do this is by creating a system of rewriting rules.

Start with an initial string, and then successively apply rules to it to create new strings.
Generative Grammar

- A grammar is a set of production rules
  - Each production rule gives instructions for how to rewrite a string
  - That is, how to turn one string into another

- Production rules
  - Symbol(s) \(\rightarrow\) other symbols

- Mechanics
  - Start with an initial string
  - Each time a symbol (or symbols) is found that matches the left hand side (LHS) of a production rule…
  - The LHS symbols are replaced by the right hand side (RHS) symbols
Generative Grammar Example

- Example production rules:
  - 1. $A \rightarrow AB$
  - 2. $B \rightarrow b$

- Application
  - Initial String: $A$

```
A
↓
A → AB
↓
AB
A → AB
↓
AB
↓
AB
B → b
↓
ABb
A → AB
↓
AB
↓
AB
B → b
↓
ABb
↓
ABbb
...
```

Typically lower case letters are terminals, and cannot be expanded any further.

Rewriting will continue on indefinitely…
Choices in Applying Rewrite Rules (1)

- Deterministic vs. Nondeterministic
  - **Deterministic**: Exactly one rule that applies to each symbol, or sequence of symbols
    - Completely unambiguous which rule to use to rewrite the string
    - For a given initial string, the output string is always the same
  - **Nondeterministic (stochastic)**: Multiple rules can apply to a given string
    - Each rewrite step involves a choice among applicable rules
    - Random
      - Make a random choice among the matching rules
      - Grammar could have probabilities for selecting each rule
    - Example: Context Free Art
  - Parameters
    - Use some parameters for deciding which rule to apply
Nondeterministic Grammar Example

- Example production rules:
  1. $A \rightarrow AB$
  2. $[0.5] \quad B \rightarrow b$
  3. $[0.25] \quad B \rightarrow c$
  4. $[0.25] \quad B \rightarrow d$

- Initial string: $A$

- Rewrite step 1 (only 1 choice, deterministic)

- Rewrite step 2 (chose rule 2 to apply)

- Rewrite step 3 (chose rule 3 to apply)

Other possible output strings after 3 steps:
- $ABcd$
- $ABdd$
- $ABbb$
- $ABcc$
- $ABbc$, etc.
Choices in Applying Rewrite Rules (2)

- **Context sensitive grammars**
  - When the rule to be applied depends on the symbols appearing before and after the symbol to be rewritten
  - Example: $B <A> b \rightarrow AB$
    - Only rewrite $A$ to $AB$ if $A$ is preceded by $B$, and followed by $b$

- **Rewriting rule application**
  - Parallel: simultaneous application of rules
    - Go through the input string and apply every rule at the same time
    - Can think of having a second string that is created by application of rules to the initial string
  - Sequential: apply one rule at a time
    - Typical Chomsky grammar approach
Lindenmayer Systems (L Systems)

- Deterministic, parallel, context-free
  - Known as D0L
  - D – deterministic
  - 0 – context free
  - L - Language

- First explored by Aristid Lindenmayer in 1968
  - Had interest in developing axiomatic theory for how plants grow

- Deterministic application of rules
  - Turtle Graphics interpretation of strings

- See:
  - Algorithmic Beauty of Plants book
    - http://algorithmicbotany.org/papers/#abop
    - Many examples of using L systems to model plant growth
  - Introduction to L systems
    - http://www.biologie.uni-hamburg.de/b-online/e28_3/lsys.html
Interpretation of Strings

- Generating sequences of characters isn’t that interesting
- To create interesting content for games, need to give meaning:
  - To characters (terminals)
  - To proximity of characters (what does it mean to be next to each other in sequence)

- Some approaches
  - Turtle graphics
    - Strings are commands in a simple language
  - Map directly to geometric primitives (or more complex shapes)
  - Map directly to elements in a level
    - Example: platforms, gaps, portals (2D platform games)
Turtle graphics

- An imaginary turtle starts at position 0,0
  - When it moves, a pen attached to the turtle draws a line
  - Developed by Seymour Papert for Logo in late 1960s

- Commands
  - F – move forward a step of length d
  - f – move forward a step of length d without drawing a line
    - Can also have pen up and pen down commands
  - + turn left by angle $\delta$
  - - turn right by angle $\delta$
  - [ - push current turtle state (x, y, angle) onto pushdown stack
  - ] – pop current turtle state from the stack
    - Use of the stack makes L-systems “bracketed”
Koch Curve

- **Example L-System**
  - 1. $F \rightarrow F + F - F - F + F$

- **Starting this system with the axiom $F$**
  - Round 0: $F$
  - Round 1: $F + F - F - F + F$
  - Round 2: $F + F - F - F + F + F + F - F + F - F + F - F + F - F + F - F - F + F + F - F - F + F + F - F - F + F + F - F - F + F$

- Not so interesting as a string. But, apply a Turtle graphics interpretation:
Impact of the stack

- Consider the same production, with the addition of stack pushes and pops
  - Rule 1: $F \rightarrow F[-F]F[+F][F]
  - If the angle is 30 degrees, instead of 90 degrees:

```
\begin{tabular}{cccc}
  n = 1 & n = 2 & n = 3 & n = 4 \\
  & & & \\
\end{tabular}
```
Shape Grammars

- So far, Turtle graphics have worked with lines and points
- But, what if you have squares, triangles, circles, (or cubes, cones, and cylinders) etc. as basic geometric primitives?
  - Would want basic operations like: rotate rectangle 90 degrees and attach to center of right face of square

- Stiny and Gips (1971) developed the notion of Shape Grammars
  - Can express complex shapes via rewrite rules
  - Has built in notion of faces, orientation, attachment points, spatial area (volume)
Shape Grammars

- In Shape Grammars, symbols are shapes
- Rules generally:
  - Substitute one shape for another shape
  - Attach a shape to an existing shape (at an attachment point, or along a wall)
  - Transform an existing shape (e.g., rotation)
  - Modify attribute information associated with a shape (e.g., intended use information for spaces in a house)


From Prairie House grammar. Note encoding of attachment points and orientation
Split Grammar

- Shape grammars are good for building up a novel shape…
- …but they are not as good for refining an existing shape
  - Example: creating a building façade requires refining an existing surface

- Split grammars are a type of shape grammar
  - Transformation rules:
    - Split: Take an existing shape and split it (decompose it) into other shapes (with the total space consumed being the same)
    - Transform: Replace an existing shape \( a \) with another shape \( b \)
      - Where \( b \) has to be contained in the volume of \( a \)

Split Grammar Example