Inductive Definitions

- Inductive definitions are the main “constructive” way to define infinite sets.
- We will need infinite sets in much of what follows.

Example: Binary Trees

- is a binary tree.
- If $T_1$ and $T_2$ are binary trees, then so is:
  \[ \bullet \]
  \[ T_1 \quad T_2 \]
- Extremal clause: The only binary trees are those constructible by a finite number of applications of the above rules.
Examples of Binary Trees

Example: Natural Numbers $\omega$

- **Basis:** 0 is in $\omega$.
- **Induction:** If $n$ is in $\omega$, so is the successor of $n$ (variously denoted $n'$, $S(n)$, or $n+1$).
- **Extremal:** The only elements in $\omega$ are those derivable by applications of the above rules.
- Examples: 0, 0', 0'', 0''', 0''''', ... are all elements of $\omega$.

Notes

- $\omega$ is an infinite set.
- $\omega$ does *not* contain infinity ($\infty$) as an element.
  - Why?
- $\omega$'s members are all finite.

Decimal Numerals

- We can agree by convention that
  - 1 stands for 0',
  - 2 stands for 0'',
  - ...
  - 9 stands for 0'''''''''.
  - Beyond that, give an algorithm for generating additional numerals:
    - 10, 11, 12, 13, ...
1-adic Numerals

- The only digit is 1.
- The empty string (denoted \( \lambda \) so it is readable) stands for 0.
- 1X (1 followed by X) stands for \( X' \).
- The numerals are: \( \lambda, 1, 11, 111, 1111, 11111, \ldots \)

- Could also use lists: [ ], [1], [1, 1], [1, 1, 1], ...

2-adic Numerals

- The digits are 1 and 2.
- The empty string (denoted \( \lambda \) so it is visible) stands for zero.
- The numerals are: \( \lambda, 1, 2, 11, 12, 21, 22, 111, 112, \ldots \)
- Unlike binary numerals, there is no redundancy (1, 01, 001, 0001, ... all mean the same thing in binary).

Roman Numerals

- The digits are I, V, X, L, C, D, M.
- There is no string for 0.
- You know the rest.

Numerals vs. Numbers

- Numbers are abstract.
- Numerals are a concrete representation.
  - There's a reason why we don't call them "Roman numbers"
Strings over an alphabet

• An alphabet is a set of arbitrary symbols.
• The set of all finite strings over an alphabet $\Sigma$ is denoted $\Sigma^*$.
• Example:
  
  $\{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, \ldots\}$

Formal definition:

• Basis: $\lambda$ is in $\Sigma^*$.
• Inductive rule: If $x \in \Sigma^*$ and $\sigma \in \Sigma$, then $x\sigma$ (x followed by $\sigma$) is in $\Sigma^*$.
• Extremal clause.

Languages

• A language over $\Sigma^*$ is any subset of $\Sigma^*$.
  • A very general definition!

• Examples, where $\Sigma = \{a, b\}$
  • $\{a, b\}^*$ itself
  • $\{\}$ the empty language
  • $\{ba, baba\}$ maybe your first language
  • $\{\lambda, aa, aaaa, aaaaaa, \ldots\}$ the language of an even number of a’s.

More Languages

• Still with $\Sigma = \{a, b\}$:
  • $\{\lambda, ab, ba, aabb, abab, baab, bbbaa, aaabbb, aababb, \ldots\}$ the language in which the number of a’s equals the number of b’s.
  • $\{a, b, aa, bb, aab, aba, baa, abb, bab, bba, \ldots\}$ the language in which the number of a’s is not equal to the number of b’s.
  • $\{\lambda, ab, aab, aabbb, ababab, abababab, \ldots\}$ a slightly less obvious language.
Languages

• There are lots of languages, some very weird.

• To be of computational interest, a language needs to be defined **inductively**.

• Given a language, need a way of telling whether a given string is in the language or not (called *parsing* the string).

Non-Trivial Language Defined Inductively

• \( L = \{ \lambda, \text{ab, abab, aabb, aababb, ...} \} \)

• Basis: \( \lambda \) is in \( L \).

• Inductive rules:
  - If \( x \) is in \( L \), so is \( axb \).
  - If \( x_1 \) and \( x_2 \) are in \( L \), so is \( x_1x_2 \).

Grammars: A Shorthand

• Spelling everything out with these inductive definitions is laborious.

• We need a shorthand, especially for more complex languages.

Grammatical Definition

• Pick a symbol \( S \) *not* in the alphabet of interest.

• \( \rightarrow \) is a symbol meaning “can be rewritten as”.

• Specify a collection of *grammar rules*:
  
  \[
  S \rightarrow \text{ab} \\
  S \rightarrow aSb \\
  S \rightarrow SS
  \]

• Starting with \( S \), what strings can we generated by replacing left-sides with right-sides of rules?
  
  - A sequence of such replacements is called a *derivation*.
  - Defines a language: the strings we can generate which *don’t* contain \( S \).
Using the Grammar Rules

- Given the grammar:
  \[
  S \rightarrow ab \\
  S \rightarrow aSb \\
  S \rightarrow SS 
  \]
- Example derivations of strings in the language:
  \[
  S \Rightarrow ab \\
  S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabbbb \\
  S \Rightarrow SS \Rightarrow SS \\Rightarrow abab \\
  S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbSb \Rightarrow aabbaabb 
  \]

More General Grammars

- Instead of just S, allow multiple symbols, (called nonterminals) none of which are in the alphabet of the language.
  - The symbols in the alphabet of the language are called terminals.
  - Pick one to be the *start symbol*.

Additive Arithmetic Expressions

- The start symbol is A.
- The terminals are \{a, b, c, +\}.
- The productions are:
  \[
  A \rightarrow V \\
  A \rightarrow V + A \\
  V \rightarrow a \\
  V \rightarrow b \\
  V \rightarrow c 
  \]

Phase-Structure Grammars

- In a phase-structure grammar, rules can have arbitrary left-hand and right-hand sides involving terminals and nonterminals:
  \[
  S \rightarrow abc \\
  S \rightarrow aSQ \\
  bQc \rightarrow bccc \\
  cQ \rightarrow Qc 
  \]
- Arbitrary grammars are too hard to work with
- In CS you're likely to see only context-free grammars
  - Left-hand side is always a nonterminal
Example Derivations

- The productions were:
  \( A \rightarrow V \)
  \( V \rightarrow a \)
  \( V \rightarrow b \)
  \( V \rightarrow c \)

- Sample derivations:
  \( A \Rightarrow V \Rightarrow a \)
  \( A \Rightarrow V \Rightarrow c \)
  \( A \Rightarrow V + A \Rightarrow c + A \Rightarrow c + V \Rightarrow c + a \)
  \( A \Rightarrow V + A \Rightarrow c + V + A \Rightarrow c + b + A \Rightarrow c + b + V \Rightarrow c + b + a \)

Shorthands on top of Shorthands

- The productions were:
  \( V \rightarrow a \)
  \( V \rightarrow b \)
  \( V \rightarrow c \)

- Group by common left-hand sides
  \( A \rightarrow V | V + A \)

- Use \( | \) (read “or”) to represent alternatives:
  \( A \rightarrow V | V + A \)
  \( V \rightarrow a | b | c \)

- Notes:
  - The symbol \( | \) “binds more loosely” than other symbols.
  - Same grammar, just a briefer notation.
  - Sometimes other separators are used for right-hand-sides

Derivation Tree Visualization

- Logics in red
- Auxiliaries in blue
- Arrows indicate that a production is being applied

Terminal string = red “fringe” of tree = “c + a + b”

Abstract Syntax Tree (≠ Derivation Tree)
Shows Implied Meaning of String

- Arrows indicate that a production is being applied
- Terminals in red
- Auxiliaries in blue

Derivation Tree

Syntax Tree