CS 60

Inductive Definitions and Grammars

October 31, 2001

Inductive Definitions

- Inductive definitions are the main "constructive" way to define infinite sets.

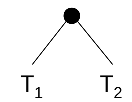
- We will need infinite sets in much of what follows.

Inductive Definitions

- Elements of an inductive definition of a set S:
  - **Basis**: Specifies that certain items are in S.
  - **Induction rule**(s): Introduce new items in S based on existence of other, usually simpler, items.
  - **Extremal clause**: Says that the only items in S are those derivable by the previous two elements, applied any finite number of times.
    - Often implicit

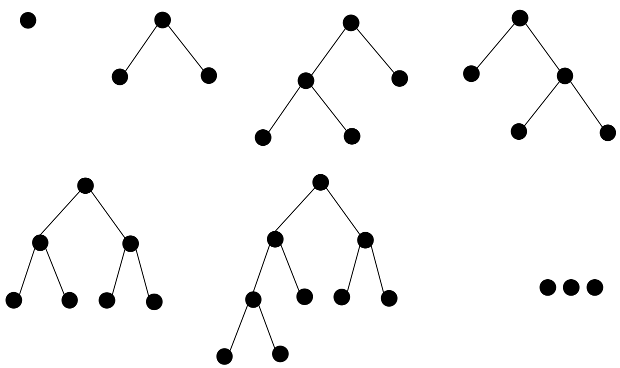
Example: Binary Trees

- ● is a binary tree.
- If T_1 and T_2 are binary trees, then so is:



- 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 ω

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

Notes

- ω is an infinite set.
- ω does *not* contain infinity (∞) as an element.
  - Why?
- ω’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 λ so it is readable) stands for 0.
- 1X (1 followed by X) stands for X'.
- The numerals are:
  λ, 1, 11, 111, 1111, 11111, ...
- Could also use lists: [ ], [1], [1, 1], [1, 1, 1], ...

2-adic Numerals

- The digits are `1` and `2`.
- The empty string (denoted λ so it is visible) stands for zero.
- The numerals are:
  `λ, 1, 2, 11, 12, 21, 22,`
  `111, 112, …`
- 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 Σ is denoted Σ*.
- Example:

```
{a, b}* =
   {λ, a, b, aa, ab, ba, bb, aaa,
    aab, aba, …}
```

Formal definition:

- Basis:

  λ is in Σ*.
- Inductive rule:

  If $x \in \Sigma^*$ and $\sigma \in \Sigma$, then
  $x\sigma$ (x followed by σ) is in Σ*.
- Extremal clause.

Languages

- A *language* over Σ* is any subset of Σ*.
  - A very general definition!

- Examples, where Σ = `{a, b}`
  - `{a, b}*` itself
  - `{ }` the empty language
  - `{ba, baba}` maybe your first language
  - `{λ, aa, aaaa, aaaa, aaaaaa, …}` the language of an even number of a's.

More Languages

- Still with Σ = `{a,b}`:
  - `{λ, ab, ba, aabb, abab, baab, bbaa, aaabbb, aababb, …}` 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, …}` the language in which the number of a's is *not* equal to the number of b's.
  - `{λ, ab, abab, aabb, aababb, abaabb, ababab, …}` a slightly less obvious langauge.

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 = {λ, `ab, abab, aabb, aababb,` …}
- Basis: λ 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 $\rightarrow$ 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:
  $S \Rightarrow ab$
  $S \Rightarrow aSb \Rightarrow aabb$
  $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
  $S \Rightarrow SS \Rightarrow abS \Rightarrow abab$
  $S \Rightarrow SS \Rightarrow SSS \Rightarrow ababab$
  $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbaSb \Rightarrow aabbaabb$

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:

  $S \rightarrow abc$
  $S \rightarrow aSQ$
  $bQc \rightarrow bbcc$
  $cQ \rightarrow Qc$
  - 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

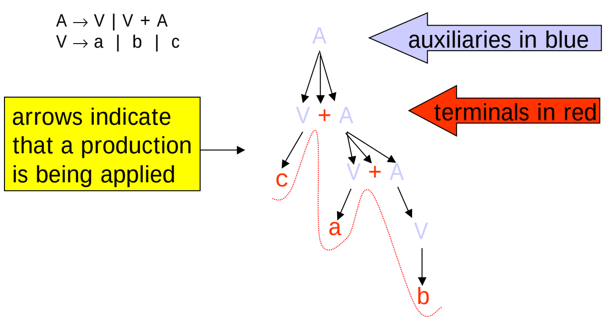
- The productions were:
  - $A \rightarrow V$
  - $A \rightarrow V + A$
  - $V \rightarrow a$
  - $V \rightarrow b$
  - $V \rightarrow c$
- Sample derivations:
  - $A \Rightarrow V \Rightarrow a$
  - $A \Rightarrow V \Rightarrow c$
  - $A \Rightarrow V + A \Rightarrow c + A \Rightarrow c + V \Rightarrow c + a$
  - $A \Rightarrow V + A \Rightarrow c + A \Rightarrow c + V + A \Rightarrow c + b + A \Rightarrow c + b + V \Rightarrow c + b + a$

Shorthands on top of Shorthands

- The productions were:
  - $A \rightarrow V$
  - $A \rightarrow V + A$
  - $V \rightarrow a$
  - $V \rightarrow b$
  - $V \rightarrow c$
- Group by common left-hand sides
- Use | (read "or") to represent alternatives:
  - $A \rightarrow V \mid V + A$
  - $V \rightarrow a \mid b \mid c$
- 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

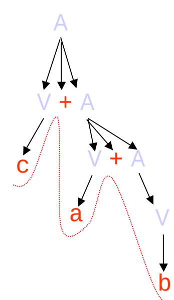
$A \rightarrow V \mid V + A$
$V \rightarrow a \mid b \mid c$



Terminal string = red "fringe" of tree = "c + a + b"

Abstract Syntax Tree (!= Derivation Tree) Shows Implied Meaning of String

Derivation Tree



Syntax Tree