COMP 1002

Logic for Computer Scientists

Lecture 24
Puzzle

• A ship leaves a pair of rabbits on an island (with a lot of food).
• After a pair of rabbits reaches 2 months of age, they produce another pair of rabbits, and keep producing a pair every month thereafter.
• Which in turn starts reproducing every month when reaching 2 months of age...
• How many pairs of rabbits will be on the island in \( n \) months, assuming no rabbits die?
Fibonacci sequence

- A ship leaves a pair of rabbits on an island (with a lot of food).
- After a pair of rabbits reaches 2 months of age, they produce another pair of rabbits, and keep producing a pair every month thereafter. Which in turn starts reproducing every month when reaching 2 months of age...
- How many pairs of rabbits will be on the island in \( n \) months, assuming no rabbits die?

- Basis: \( F_1 = 1, \ F_2 = 1 \)
- Recurrence: \( F_n = F_{n-1} + F_{n-2} \)
- Sequence: 1,1,2,3,5,8,13...
Partial sums

- Properties of a sum ("linearity"):
  \[ \sum_{i=m}^{n} (f(i) + g(i)) = \sum_{i=m}^{n} f(i) + \sum_{i=m}^{n} g(i) \]
  \[ \sum_{i=m}^{n} c \cdot f(i) = c \sum_{i=m}^{n} f(i) \]

Sum of arithmetic progression:
\[ s_n = c + nd \quad \text{for some } c, d \in \mathbb{R} \]
Sequence: \( c, c + d, c + 2d, c + 3d, \ldots, c + nd, \ldots \)
Partial sum:
\[ \sum_{i=0}^{n} s_n = \sum_{i=0}^{n} (c + id) = \sum_{i=0}^{n} c + \sum_{i=0}^{n} id = c(n + 1) + d \sum_{i=0}^{n} i = c(n + 1) + d \frac{n(n + 1)}{2} \]

Sum of geometric progression:
\[ s_n = c \cdot r^n \quad \text{for some } c, r \in \mathbb{R} \]
Sequence: \( c, cr, cr^2, cr^3, \ldots, cr^n, \ldots \)
Partial sum:
\[ \sum_{i=0}^{n} s_n = \begin{cases} 
  c(n + 1), & \text{if } r = 1 \\
  \frac{cr^{n+1} - c}{r - 1}, & \text{if } r \neq 1
\end{cases} \]
Fractals

• Can use recursive definitions to define fractals
  – And draw them
  – And prove their properties.

• Self-similar: a part looks like the whole.
Fractals in nature

• A fern leaf

• Broccoli

• Mountains

• Stock market

• Heat beat
Mathematical fractals

• Koch curve and snowflake

• Sierpinski triangle, pyramid, carpet

• Hilbert space-filling curve

• Mandelbrot set
Koch curve

- **Basis**: an interval
- **Recursive step**: Replace the inner third of the interval with two of the same length
- ...
Playing with fractals

- Fractal Grower by Joel Castellanos:
Recursive definitions of sets

• So far, we talked about recursive definitions of sequences. We can also recursively define sets.
  – E.g: recursive definition of a set $S=\{0,1\}^*$
    • Basis: empty string is in $S$.
    • Recursive step: if $w \in S$, then $w0 \in S$ and $w1 \in S$
      – Here, $w0$ means string $w$ with 0 appended at the end; same for $w1$
  – Alternatively:
    • Basis: empty string, 0 and 1 are in $S$.
    • Recursive step: if $s$ and $t$ are in $S$, then $st \in S$
      – Here, $st$ is concatenation: symbols of $s$ followed by symbols of $t$
      – If $s = 101$ and $t = 0011$, then $st = 1010011$
  – Additionally, need a restriction condition: the set $S$ contains only elements produced from basis using recursive step rule.
Trees

• In computer science, a **tree** is an undirected graph without cycles
  – **Undirected**: all edges go both ways, no arrows.
  – **Cycle**: sequence of edges going back to the same point.

• Recursive definition of trees:
  – Base: A single vertex is a tree.
  – Recursion:
    • Let \( T \) be a tree, and \( v \) a new vertex.
    • Then a new tree consist of \( T, v \), and an edge (connection) between some vertex of \( T \) and \( v \).
  – Restriction:
    • Anything that cannot be constructed with this rule from this base is not a tree.
Arithmetic expressions

• Suppose you are writing a piece of code that takes an arithmetic expression and, say evaluates it.
  – “5*3-1”, “40-(x+1)*7”, etc

• How to describe a valid arithmetic expression? Define a set of all valid arithmetic expressions recursively.
  – Base: A number or a variable is a valid arithmetic expression.
    • 5, 100, x, a,
  – Recursion:
    • If A and B are valid arithmetic expressions, then so are (A), A + B, A − B, A * B, A / B.
      – Constructing 40-(x+1)*7: first construct 40, x, 1, 7. Then x+1. Then (x+1). Then (x+1)*7, finally 40-(x+1)*7
      – Caveat: how do we know the order of evaluation? On that later.
  – Restriction: nothing else is a valid arithmetic expression.
Formulas

• What is a well-formed propositional logic formula?
  – \((p \lor \neg q) \land r \rightarrow (\neg p \rightarrow r)\)
  – Base: a propositional variable \(p, q, r \ldots\)
    • Or a constant \(TRUE, FALSE\)
  – Recursion:
    • If \(F\) and \(G\) are propositional formulas, so are \((F), \neg F, F \land G, F \lor G, F \rightarrow G, F \leftrightarrow G\).
  – And nothing else.
Formulas

• What is a well-formed predicate logic formula?
  – $\exists x \in D \ \forall y \in \mathbb{Z} \ P((x, y) \lor Q(x, z)) \land x = y$
  – Base: a predicate with free variables
    • $P(x), \ x = y, ...$
  – Recursion:
    • If $F$ and $G$ are predicate logic formulas, so are $(F), \ \neg F, \ F \land G, \ F \lor G, \ F \rightarrow G, \ F \leftrightarrow G$.
    • If $F$ is a predicate logic formula with a free variable $x$, then $\exists x \in D \ F$ and $\forall x \in D \ F$ are predicate logic formulas.
  – And nothing else.
    • So $\exists x \in \text{People} \ Likes(x, y \land x), \ Likes(y \neq x)$ is not a well-formed predicate logic formula!
Grammars

• A general recursive definition for these is called a grammar.
  – In particular, here we have “context-free” grammars, where symbols have the same meaning wherever they are.

• A context-free grammar consists of
  – A set \( V \) of variables (using capital letters)
    • Including a start variable \( S \).
  – A set \( \Sigma \) of terminals (disjoint from \( V \); alphabet)
  – A set \( R \) of rules, where each rule consists of a variable from \( V \) and a string of variables and terminals.
    • If \( A \rightarrow w \) is a rule, we say variable \( A \) yields string \( w \).
      – This is not the same “\( \rightarrow \)" as implication, a different use of the same symbol.
    • We use shortcut “|” when the same variable might yield several possible strings: \( A \rightarrow w_1 | w_2 | ... | w_k \)
    • Can use \( A \) again within the rule: Recursion!
      – Different occurrences of the same variable can be interpreted as different strings.
  • When left with just terminals, a string is derived.
Grammars

• A language generated by a grammar consists of all strings of terminals that can be derived from the start variable by applying the rules.
  – All strings are derived by repeatedly applying the grammar rules to each variable until there are no variables left (just the terminals).

– Language \{1, 00\} consisting of two strings 1 and 00
  \[
  S \rightarrow 1 \mid 00
  \]
  – Variables: S. Terminals: 1 and 00.

– Language of all strings over \{0, 1\} with all 0s before all 1s.
  \[
  S \rightarrow 0S \mid S1 \mid _\]
  – Variables: S. Terminals: 0 and 1.
More context-free grammars

- Propositional formulas.
  1. \( F \rightarrow F \lor F \)
  2. \( F \rightarrow F \land F \)
  3. \( F \rightarrow \neg F \)
  4. \( F \rightarrow F \)
  5. \( F \rightarrow p \mid q \mid r \mid TRUE \mid FALSE \)
     - Here, the only variable is \( F \) (it is a start variable), and terminals are \( \lor, \land, \neg, (, ), p, q, r, TRUE, FALSE \)
     - To obtain \( (p \lor \neg q) \land r \), first apply rule 2, then rule 4 to strip parentheses from \( (p \lor \neg q) \), then rule 1, then rule 5 to get \( p \), then rule 3, then rule 5 to get \( q \), then rule 5 to get \( r \).

- Arithmetic expressions.
  - \( EXPR \rightarrow EXPR + EXPR \mid EXPR - EXPR \mid EXPR \ast EXPR \mid EXPR / EXPR \mid (EXPR) \mid NUMBER \mid -NUMBER \)
  - \( NUMBER \rightarrow 0DIGITS \mid ... \mid 9DIGITS \)
  - \( DIGITS \rightarrow _\mid NUMBER \)
     - Here, _ stands for empty string. Variables: EXPR, NUMBER, DIGITS (S is starting).
       Terminals: +,-,*,, /,, 0,,9,,(,).
     - We used separate NUMBER to avoid multiple “-”.
     - And separate DIGITS to have an empty string to finish writing a number, but to avoid an empty number.
Encoding order of precedence

• Easier to specify in which order to process parts of the formula.
  – Better grammar for arithmetic expressions (for simplicity, with x,y,z instead of numbers):
    1. \( EXPR \rightarrow EXPR + TERM \mid EXPR - TERM \mid TERM \)
    2. \( TERM \rightarrow TERM \ast FACTOR \mid TERM / FACTOR \mid FACTOR \)
    3. \( FACTOR \rightarrow (EXPR) \mid x \mid y \mid z \)
  – Here, variables are EXPR, TERM and FACTOR (with EXPR a starting variable).
  – Now can encode precedence.
    • And put parentheses more sensibly.
Parse trees.

• Visualization of derivations: parse trees.

1. $EXPR \rightarrow EXPR + TERM \mid EXPR - TERM \mid TERM$
2. $TERM \rightarrow TERM \ast FACTOR \mid TERM / FACTOR \mid FACTOR$
3. $FACTOR \rightarrow (EXPR) \mid x \mid y \mid z$

• String $(x+y)z$
  – Simpler example:
    • $S \rightarrow 0S \mid S1 \mid _$
• String 001
Puzzle

• Do the following two English sentences have the same parse trees?

– Time flies like an arrow.

– Fruit flies like an apple.