Lecture 27: Unrestricted Grammars

CSCI 101
Spring, 2019

Kim Bruce

Unrestricted Grammars

- An unrestricted, or type 0 grammar G is a quadruple (V, Σ, R, S), where:
  - V is an alphabet,
  - Σ (the set of terminals) is a subset of V,
  - R (the set of rules) is a finite subset of $(V^+ \times V^*)$,
  - S (the start symbol) is an element of V - Σ.
- The language generated by G is:
  $\{w \in \Sigma^* : S \Rightarrow_G^* w\}$.

Example 1:

- $A^nB^nC^n = \{a^nb^nc^n, n \geq 0\}$
  - S → aBSc
    S → ε
    Ba → aB
    Bc → bc
    Bb → bb
- Proof:
  - Gives only strings in $A^nB^nC^n$:
  - All strings in $A^nB^nC^n$ are generated:

Example 2

- $\{w \in \{a, b, c\}^* : \#_a(w) = \#_b(w) = \#_c(w)\}$
  - S → ABCS
    S → ε
    AB → BA
    BC → CB
    AC → CA
    BA → AB
    CA → AC
    CB → BC
    A → a
    B → b
    C → c

Example 3

- WW = {ww : w ∈ {a, b}*}
  - *Idea: Generate $wCw^R\#$ and then reverse last part*

WW = {ww : w ∈ {a, b}*}

- S → T# /* Generate the wall exactly once.
  T → aTa /* Generate wCw^R.
  T → bTb "
  T → C "
  C → CP /* Generate a pusher P
  Paa→ aPa /* Push one character to the right to get ready to jump.
  Pab → bPa "
  Pba → aPb "
  Pbb → bPb "
  Pa# → #a /* Hop a character over the wall.
  Pb# → #b "
  C# → ε

Computability

- Theorem: A language is generated by an unrestricted grammar if and only if it is in SD.
- Proof:
- (Grammar ⇒ TM): by construction of an NDTM.
- (TM ⇒ grammar): by construction of a grammar that mimics the behavior of a semi-deciding TM.

Proof of Equivalence

- (Grammar ⇒ TM): by construction of a two-tape NDTM.
  - Suppose S ⇒* w.
    w is on tape 1. Start w/S on tape 2.
  - Tape 2 simulates derivation.
  - Non-deterministically choose production to apply to contents of tape 2. Rewrite string as appropriate.
  - After each step see if matches input. If yes, halt.
  - Semi-decides L(G).

Proof of Equivalence

- (TM ⇒ Grammar): Construct grammar G to simulate TM M.
  - Phase 1 generates a candidate string for acceptance.
    Phase 2 will then simulate the TM computation on the string.
    Phase 3 will clean up the tape, so tape only contains original candidate string.
  - Problem: Original string got replaced during simulation!
  - Solution: Duplicate it on odd cells of tape and only compute on the evens, preserving odds.

Proof of Equivalence

- (TM ⇒ Grammar): Construct grammar G to simulate TM M.
  - Phase 1: Generate a candidate string for acceptance.
    - Generate a string of form *Note duplicates!!*
      # □□ q000 a1 a1 a2 a2 a3 a3 □□ #
      representing input $a_1a_2a_3$ and q000 is encoding of start state
  - Phase 2: If δ(p,a) = (q,b,→) add rule:
    - p' z a → z b q' *where p', q' are codes of states p,q*
    - *If δ(p,a) = (q,b,←) add x y p' z a → q' x y z b*

Proof of Equivalence

- (TM ⇒ Grammar): Construct grammar G to simulate TM M.
  - Phase 3: If get to accept state A, clean up:
    - x A → A x for x ≠ #, *move A to left edge of input*
    - # A x y → x #A *sweep through gathering odd -- erasing even*
    - # A # → ε *leaves original string as final string*

Other formalisms

- Partial recursive functions:
  - projection, constant, successor, closed under composition, primitive recursion, and minimization
  - To show equivalence with TM's must encode configurations, configuration histories, etc. as numbers.
- Lambda calculus
- RAM machines

Undecidability

- All undecidability results carry over as could use to solve corresponding TM problems using simulations.

Self-Reproducing Program

- Can you write a program in your favorite programming language that prints out a copy of itself?
  - Try it!

Virus Program

- virus() =
  1. For each address in address book do:
     1.1. Write a copy of myself.
     1.2. Mail it to the address.
  2. Do something malicious like change one bit in every file on the machine.
  3. Halt.
- Can we implement step 1.1?
  - Print two copies of the following with the second in quotes: "Print two copies of the following with the second in quotes:"

Fixed Points

- Consider $f(k) = k \quad$ if $k \le 1$
  $\qquad\qquad\quad = f(k-1) + f(k-2) \quad$ otherwise
- Function f defined in terms of itself.
- Think of as equation to be solved.
  - f = fun(k). if k≤1 then k else f(k-1)+f(k-2)
- Write right side as function of g:
  - F(g) = fun(k). if k≤1 then k else g(k-1)+g(k-2)
- Looking for f s.t. f = F(f) *f is fixed pt for F*

Recursion Theorem

- Rough versions:
  - First Thm: If F is computable then F has a computable fixed point.
  - Second Thm: We can compute the program for a fixed point of F from a program for F.
- True for any formalism giving all computable fcns.
  - In lambda calculus, $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ gives fixed points. I.e. for all f, if $x_o = Yf$ then $f(x_o) = x_o$

Harder for Turing machines!