Lecture 25: Language Hierarchies: Decidable & Semi-Decidable

CSCI 101
Spring, 2019

Kim Bruce

Decision Problems for Regular Languages

- For FSMs/Regular languages, most things decidable:
  - A_{FSM} = {⟨M,w⟩ | M is an FSM and w ∈ L(M)}
  - E_{FSM} ={M | M is a FSM and L(M)=∅}
  - $TOTAL_{FSM}$ ={M | M is a FSM and L(M)=Σ*}
  - $EQUAL_{FSM}$ ={<M,N> | M,N are FSMs and L(M)=L(N)}

Decision Problems for CFGs

- Not for PDAs/CFGs:
  - A_{CFG} = {⟨G,w⟩ | G is a CFG and w ∈ L(G)}
  - E_{CFG} ={G | G is a CFG and L(G)=∅}
  - $Finite_{CFG}$ = {G | G is a CFG and L(G) is finite}
  - $TOTAL_{CFG}$ ={G | G is a CFG and L(G)=Σ*}
  - $EQUAL_{CFG}$ ={< G, G' > | G, G' are CFGs and L(G)=L(G')}
  - ....

Not Decidable!

$TOTAL_{CFG}$ is the key!

- Assume $TOTAL_{CFG}$ is undecidable and show others undecidable.
- $EQUAL_{CFG}$ ={< G, G' > | G, G' are CFGs & L(G)=L(G')}
  - Let G_{Tot} be a fixed grammar s.t. $L(G_{Tot})$ = Σ*
  - Suppose *Oracle* decides $EQUAL_{CFG}$
  - To decide if G total, ask oracle about <G, G_{Tot}>.
  - If yes, then G is total. If no, then not.
  - By contradiction, $EQUAL_{CFG}$ is undecidable

Minimizing PDA's

- MIN_{PDA} = {<M_1, M_2>: M_2 is a minimization of M_1} is undecidable.
  - Proof: Suppose Oracle to solve MIN_{PDA}.
    Let P_a be PDA with one state that accepts everything (never push anything on stack).
  - Given cfg G, construct equivalent PDA P s.t. L(P) = L(G).
  - Submit <P,P_a> to Oracle and get answer to L(P) = L(G) = Σ^*

Other Undecidable

- Is L(G) inherently ambiguous?
- Is $L(G) \cap L(G') = \emptyset$?
- If $L(G) \subseteq L(G')$?
- Is complement of L(G) a cfl?
- Is L(G) regular?

$Total_{CFG}$ is Undecidable

- Recall: Configuration of TM M is a 4 tuple:
  - M's current state
  - nonblank portion of the tape before the read head,
  - the character under the read head,
  - the nonblank portion of the tape after the read head

Computation

- A computation of M is a sequence of configurations:
  $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that:
  - C_0 is the initial configuration of M,
  - C_n is a halting configuration of M, and:
  - $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \ldots \vdash_M C_n$.
- Computation history is sequence of configurations.

Proof

- Theorem: $Total_{CFG}$ is undecidable
  - Proof: Reduction via halting
  - Given M,w, build grammar G generating language L composed of all strings in Σ* except any representing a (halting) computation history of M on w.
  - Suppose *Oracle* solves $Total_{CFG}$. Run on G.
    - If says yes, then M doesn't halt on w
    - If says no, then exist halting computation from w.
    - Contradiction!

Recognizing Computation Histories

- Build PDA rather than CFG and then convert.
- For s to be computation history of M on w:
  - It must be a syntactically valid computation history.
  - C_0 must correspond to M being in its start state, with w on the tape, and with the read head to the left of w.
  - The last configuration must be a halting configuration.
  - Each configuration after C_0 must be derivable from the previous one according to the rules in δ_M.

Invalid Computation Histories

- Recognizing valid computations hard!
  - Can get as intersection of two cfls!
- Invalid easier! PDA can guess one of the following fails (use non-determinism!)
  - Invalid syntax for configuration sequence.
  - C_0 not rep. opening config (bad state or input)
  - Last configuration not halting
  - Successor config not follow from previous according to transition function.

Recognizing Invalid Computations

- Last check can be done easily if have extra tape on TM (or extra read head)
- To check last point (transitions incorrect) with pda, must save a configuration on stack in order to check next.
- But elements popped off stack in opposite order added (LIFO). How to compare??

Boustrophedon??

- Solve by writing every other configuration backwards, so can compare via stack.
  - This text is written
  - ot yaw yzarc siht ni
  - show Boustrophedon style.
- Assume computation history written in Boustrophedon style
- Exists iff regular history exists!

Invalid Computation History

- If guessing particular step of computation is wrong in C_0 C_1^R C_2 ...
  - Keep track of which direction going
  - Push C_i (possibly reversed) onto stack
  - Compare C_{i+1} to make sure that there is an error
    - In copying unchanged portion of configuration or
    - In changing part reflecting transition.
- Hence whether $L(G) = \Sigma^*$ is undecidable.

Unrestricted Grammars

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 $V - \Sigma$.
- 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

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

$WW = \{ww : w \in \{a, b\}^*\}$

- 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 $S \Rightarrow^* w$.
    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
      # □□ q000 a1 a1 a2 a2 a3 a3 □□ #
      representing input $a_1a_2a_3$ and q000 is encoding of start state
  - Phase 2: If $\delta(p,a) = (q,b,\rightarrow)$ add rule:
    - p' z a → z b q' *where p', q' are codes of states p,q*
    - *If $\delta(p,a) = (q,b,\leftarrow)$ add* *x y p' z a → q' x y z b*