# Lecture 34: Reductions and Undecidability

CSCI 81
Spring, 2015

Kim Bruce

## Application of Homework

- Can define the typed lambda calculus.
- Can prove that every program in typed lambda calculus is total.
- Therefore does not include all total computable functions.
- Type systems are either conservative or incorrect.

## Rice's Theorem

- No non-trivial property of the SD languages is decidable
  *or equivalently*
- Any language that can be described as
  {<M> | P(L(M)) = true}
  for any non-trivial property P, is not in D
- A property is *trivial* if it is either true for all languages or false for all languages.

## Applying Rice

- Must specify property P
- Show domain of P is SD languages, e.g., languages accepted by TM's.
- Show P is non-trivial
  - *true of at least one , false of at least one.*

# Proof of Rice

- Proof: Let P be a non-trivial property of SD languages. Show can reduce Halting to $L = \{\langle M \rangle \mid P(L(M)) = \text{true}\}$
- $\varnothing$ is an SD language. Suppose $P(\varnothing)$ = true
  - Similar proof if it is false (use not-P instead).
- Since P is non-trivial, $\exists L'$ in SD s.t. $P(L')$ = false. Let M' semi-decide L'

# Proof of Rice *(cont.)*

- Build new TM from M' that will decide $H_{TM}$.
- Given $\langle M,w \rangle$, build $M_w$' s.t., when started w/ x:
  - Copy x onto another work tape
  - Erase and write w on input tape
  - Run M on w
  - Copy x back on tape and run M' on x
- Recall M' semi-decides L' and $P(L')$ = false

# Proof of Rice *(cont.)*

- Recall M' semi-decides L' and $P(L')$ = false
- If $\langle M,w \rangle \in H_{TM}$ then $M_w$' accepts L'
- If $\langle M,w \rangle \notin H_{TM}$ then $M_w$' accepts $\varnothing$.
- Thus $\langle M,w \rangle \in H_{TM}$ iff $P(L(M_w')) = \text{false}$
  iff $\text{not}(\langle M_w' \rangle \in L)$
- Therefore $H_{TM} \leq_M L$ and L is undecidable!

# More Undecidable

- Is L(M) regular?
- Is L(M) context-free?
- Can we automatically check if your program is correct (e.g., matches the solution)?
- Does M ever halt with an error?

# 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 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 contradication, $EQUAL_{CFG}$ 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 build grammar G generating language L composed of all strings in $\Sigma^*$ 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.
  - $C_o$ 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 Co must be derivable from the previous one according to the rules in $\delta_M$.

# 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.
  - $C_o$ 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 $C_0\ C_1^R\ C_2 \ldots$
  - 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.