Lecture 10

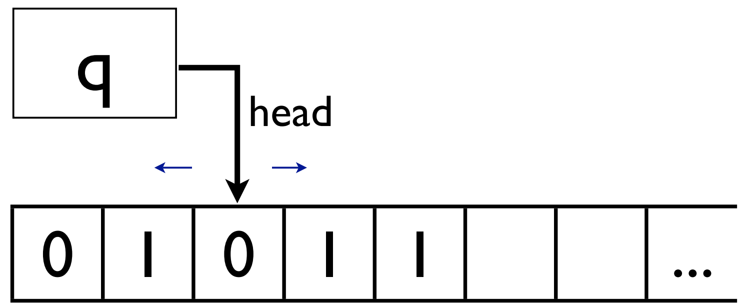
$\text{EMPTY}_{\text{LBA}}$ is undecidable

An alternate proof, using a new technique –

Computation histories

Computation Histories

Configuration:
state, head, tape



Encoding Configs:
0 1 q 0 1 1
A string in $\Gamma^* \, Q \, \Gamma^*$ (trailing blanks optional)

Accepting (Rejecting) *History*: $C_1, C_2, \ldots, C_n$ s.t.
1. C_1 is M's initial configuration
2. C_n is an accepting (rejecting, resp.) config, and
3. For each $1 \leq i < n$, C_i moves to C_{i+1} in one step

Checking Histories

Many proofs require *checking* that a string, say

$\# C_1 \# C_2 \# \ldots \# C_n \#$ in $(\{\#\} \cup Q \cup \Gamma)^*$

is/is not an accepting history:

1. C_1 is M's initial configuration:

   $C_1 \in q_0 \Sigma^*$

2. C_n is an accepting config:

   it contains q_{accept}

3. For each $1 \leq i < n$, C_i moves to C_{i+1} in one step

   ...

"C_i moves to C_{i+1} in one step of M"

$$\#a_1a_2\ldots \boxed{a_k p a_{k+1}} a_{k+2}\ldots a_n\#b_1b_2\ldots \boxed{b_j q b_{j+1}} b_{j+2}\ldots b_m\#$$

No change:

$a_i = b_i \in \Gamma$

$p,\ q \in Q,$

$j = k \pm 1$

$n = m$

Except for adjustments, all near the head, reflecting the move:

$\delta(p, a_{k+1}) = (q, b_{k+1}, L/R),$

$j = k+1$ if R else $\max(k-1,0)$

and injecting blanks on the right as needed:

if $n = k$, then "a_{k+1}" = blank

$m = n+1$, ...

Aside: one reason TM's have been so useful for computation theory is that they make questions like this very simple; "config" and "move" are much messier for "real" computers.

$A_{TM} \leq_T EMPTY_{LBA}$

Given $\langle M, w \rangle$, build an LBA $L_{M,w}$ that recognizes

$$AH_{M,w} = \{\ x \mid x = \#\ C_1\ \#\ C_2\ \#\ \ldots\ \#\ C_n\ \#,\ \text{an } \textit{A}\text{ccepting computation } \textit{H}\text{istory of M on w}\ \}$$

Then pass $\langle L_{M,w} \rangle$ to the hypothetical subr for $EMPTY_{LBA}$

Specifically, $L_{M,w}$ operates by checking that:

1. Its input is of the form $\#\ C_1\ \#\ C_2\ \#\ \ldots\ \#\ C_n\ \#$
2. C_1 is the initial config of M on w
3. C_n has M's accept state, and
4. For each $1 \leq i < n$, C_i moves to C_{i+1} in one step of M
   (ziz-zag across adjacent pairs, checking as on prev slide)

Correctness

$L(L_{M,w}) = AH_{M,w} = \{ x \mid x = \# C_1 \# C_2 \# \ldots \# C_n \#,$ an accepting computation history of M on w $\}$

Empty if M rejects w – no such x

Non-empty if M accepts w – there is one such history

So, "M accepts w" is equivalent to (non-)emptyness of $AH_{M,w}$

$\therefore A_{TM} \leq_T EMPTY_{LBA}$ QED

Notes

Similar ideas can be used to give reductions like

$$A_{TM} \leq_T EMPTY_X$$

for any machine or language class X expressive enough that we can easily, given M & w, represent $AH_{M,w}$ in X

A nice thing about histories is that they are so transparent that this is easy, even for more restricted models than LBA's

(One example in homework; another below)

ALL_{CFL} is Undecidable

ALL_{CFL} = { <G> | G is a CFG with $L(G) = \Sigma^*$ }

A variant on the above proof, but instead of using $AH_{M,w}$, (the set of accepting histories of M on w), we use its *complement:*

$NH_{M,w}$ = { x | x is *not* an accepting computation history* of M on w }

* and change the representation of a history so that alternate configs are reversed:

$\# C_1 \# C_2^R \# C_3 \# C_4^R \#... \# C_n^{(R?)} \#$

$A_{TM} \leq_T ALL_{CFL}$

Given M, w, build a PDA P that, on input x, accepts if x does *not* start and end with #; otherwise, let

$$x = \# C_1 \# C_2^R \# C_3 \# C_4^R \# \ldots \# C_n^{(R?)} \#$$

and nondeterministically do one of:

1. accept if C_1 is *not* M's initial config on w
2. accept if C_n is *not* accepting, or
3. nondeterministically pick i and verify that C_i does *not* yield C_{i+1} in one step. (Push 1st; pop & compare to 2nd, with the necessary changes near the head.)

From P, build equiv CFG G; ask the hypothetical ALL_{CFL} subr if G generates all of $(\{\#\} \cup Q \cup \Gamma)^*$

Computable Functions

In addition to language recognition, we are also interested in computable functions.

Defn: a function $f: \Sigma^* \rightarrow \Sigma^*$ is *computable* if $\exists$ a TM M s.t. given *any* input $w \in \Sigma^*$, M *halts* with just f(w) on its tape.
(Note: domain(f) = Σ^*; crucial that M always halt, else value undefined.)

Ex 1: $f(n) = n^2$ is computable

Ex 2: $g(\langle M,w \rangle) = \langle L_{M,w} \rangle$ (as in the EMPTY_{LBA} pf) is computable

Ex 2: $h(\langle M,w \rangle)$ = "1 if M acc w else 0" is *un*computable
(Why? Reduce A_{TM} to it.)

Reducibility

"A reducible to B" means could solve A *if* had subr for B

Can use B in arbitrary ways–call it repeatedly, use its answers to form new calls, etc. E.g.,

$$\text{WHACKY} \leq_T A_{TM}$$

where WHACKY = { <M,w_1, w_2, ...,w_n> | M accepts

$a_1 \cdots a_n$, where a_i = 0 if M rejects w_i, 1 if accepts w_i }

BUT in "practice," *reductions rarely exploit this generality* and a more refined version is better for some purposes

Reduction

Notation (not in book, but common):

$A \leq_T B$ means "A is Turing Reducible to B"

I.e., if I had a TM deciding B, I could use it as a subroutine to solve A

Facts:

$A \leq_T B$ & B decidable implies A decidable (definition)

$A \leq_T B$ & A *un*decidable implies B undecidable (contrapositive)

$A \leq_T B$ & $B \leq_T C$ implies $A \leq_T C$

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \leq_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\leq_T$:

Call subr only once; its answer is *the* answer

Facts:

$A \leq_m B$ & B decidable $\Rightarrow$ A is too

$A \leq_m B$ & A *un*decidable $\Rightarrow$ B is too

$A \leq_m B$ & $B \leq_m C \Rightarrow A \leq_m C$

Mapping Reducibility

Defn: A is *mapping reducible* to B ($A \leq_m B$) if there is computable function *f* such that $w \in A \Leftrightarrow f(w) \in B$

A special case of $\leq_T$:

Call subr only once; its answer is *the* answer

Facts:

$A \leq_m B$ & B decidable (recognizable) $\Rightarrow$ A is too

$A \leq_m B$ & A *un*decidable (unrecognizable) $\Rightarrow$ B is too

$A \leq_m B$ & $B \leq_m C \Rightarrow A \leq_m C$

Most reductions we've seen were actually $\leq_m$ reductions.