Objectives for Today

- Multi-Tape Turing Machines
- Universal Turing Machine
Multiple tapes

- We could define a TM with multiple tapes:
  - input on first tape, other tapes initially blank
  - transition function has type \( \delta : Q \times \Gamma^3 \rightarrow Q \times \Gamma^3 \times \{L, R\}^3 \)

- **Q.** More powerful than an ordinary one-tape TM?
  - **A. No!** We can use a similar construction to the one in Lecture 14 to construct a one-tape TM \( N \) that simulates a multi-tape TM \( M \);
    - one \( M \)-move simulated with several \( N \)-moves (see Kozen pp. 222-223)
Universal Turing Machine

- there exists a TM $U$ such that:

$$L(U) = \{ M \#x \mid x \in L(M) \}$$

- here ‘$M \#x$’ means: encoding of $M$ followed by a separator $\#$ followed by encoding of string $x$ in $M$’s input alphabet.

- $U$ will work roughly as follows:

1. check whether $M$ and $x$ are correct encodings, reject if not
2. simulate $M$ on $x$;
3. accept if $M$ accepts, reject if $M$ rejects.
The UTM and Real Life

- The UTM is a complex exercise in hacking TMs;
- Not just a theoretical curiosity, we use things like UTM every day
- In terms of programming langs, it is an interpreter - a program that takes in program + input and simulates the expected computation, eg:
  - a C interpreter written in C
  - a Java interpreter written in Java
  - a Java interpreter written in C, etc.
- clearly related to compilers... (except the UTM was described in the 1930’s!)
Encoding Turing Machines

• details not important, many possible ways;

• e.g. (Kozen) – the encoding starts with

\[ 0^n 10^m 10^k 10^s 10^t 10^r 10^u 10^v \]

• decodes to: states \( \{0, 1, \ldots, n - 1\} \), tape alphabet \( \{0, 1, \ldots, m - 1\} \) of which first \( k \) numbers is the input alphabet. The start, accept and reject states are \( s, t \) and \( r \). The blank symbols is \( u \) and the endmarker is \( v \).

• the remainder might consist of strings:

\[ 0^p 10^a 10^q 10^b 10 \]

that decode to: \( \delta(p, a) = (q, b, L) \).
Constructing a UTM $U$

- On $M \neq x$, first checks that encodings are correct;
- Use three tapes to store:
  1. description of $M$;
  2. contents of $M$’s tape;
  3. $M$’s current state & position on its tape;
- in each step, $U$:
  - looks at $M$’s current state and head position (tape 3);
  - reads the tape contents at the correct position (tape 2);
  - reads the relevant transition (tape 1);
  - simulates transition, updating tape, state and head position;
  - accepts if $M$ accepts, rejects if $M$ rejects.
UTM and the halting problem

- the UTM is quite primitive – it reads in (encoding of) a TM \( M \), an (encoding of) input \( x \) and simulates \( M \) on \( x \) – accepting, rejecting or looping depending on what \( M \) does, indeed:

\[
L(U) = \{ M \# x \mid x \in L(M) \}
\]

and the UTM will loop on \( M \# x \) whenever \( M \) loops on \( x \);

- can we come up with a smarter TM?

- for example, consider the following set: is it recursive?

\[
\text{HP} \overset{\text{def}}{=} \{ M \# x \mid M \text{ halts on } x \}
\]
The halting problem

- In other words, can we come up with the following total TM. On $M \#x$ it:
  - halts and accepts if $M$ halts on $x$ (either accepts or rejects);
  - halts and rejects if $M$ loops on $x$.

- Equivalent to writing a C program that takes a .zip file containing some C program $X$ together with some input $x$, and returns
  - 0 if $X$ halts on $x$;
  - 1 if $X$ gets into an infinite loop on $x$.

- This would be useful for compiler engineers!