Representing configurations We represent a configuration as us_iv, where u, v are strings in the tape alphabet $\tilde{A}$ and s_i is the current state of the machine. The tape contents are then the string uv and the current location of the tape head is on the first symbol of v. The assumption here is that the tape contains only blanks after the last symbol in v.

Example $\varepsilon i 001$ is the configuration $\boxed{0}\boxed{0}\boxed{1}\boxed{\sqcup}\cdots$ as we start

$\Uparrow$ tape head

examining the string 001 in our previous example of a Turing machine.

Def Let C_1, C_2 be two configurations of a given Turing machine. We say that the configuration C_1 yields the configuration C_2 if the Turing machine can go from C_1 to C_2 in one step.

Example If s_i, s_j are states, u and v are strings in the tape alphabet $\tilde{A}$, and $a, b, c \in \tilde{A}$. A configuration $C_1 = uas_ibv$ yields a configuration $C_2 = us_jacv$ if the transition mapping t specifies a transition $t(s_i, b) = (s_j, c, L)$. In other words, the Turing machine is in state s_i, it reads character b, writes character c in its place, enters state s_j, and its head moves left.

Types of configurations

- initial configuration with input u is iu, which indicates that the machine is in the initial state i with its head at the leftmost position on the tape (which is the reason why this configuration has no string left of the state).
- accepting configuration $us_{accept}v$ for $u, v \in \tilde{A}^*$ (u, v string in $\tilde{A}$),

namely the machine is in the accept state.

- rejecting configuration $u s_{reject} v$ for $u, v \in \tilde{A}^*$, namely the machine is in the reject state.
- halting configurations yield no further configurations; no transitions are defined out of their states. Accepting and rejecting configurations are examples of halting configurations.

Def A Turing machine M accepts input $w \in A^*$ (string over the input alphabet A) if $\exists$ sequence of configurations $C_1, C_2, \dots, C_k$ such that:

1. C_1 is the start configuration with input w.
2. Each C_i yields C_{i+1} for $i = 1, \dots, k-1$.
3. C_k is an accepting configuration.

Def Let M be a Turing machine. $L(M) = \{ w \in A^* \mid M \text{ accepts } w \}$ is the language recognized by M.

Def A language $L \subset A^*$ is called Turing-recognizable if $\exists M$ a Turing machine that recognizes L, i.e. $L = L(M)$.

NB Some textbooks use the terminology recursively enumerable language (RE language) instead of Turing-recognizable.

Turing-recognizable is not necessarily as strong a notion as we might need because a Turing machine can
- accept
- reject
- loop

Looping is any simple or complex behaviour that does not lead to a halting state. The problem with looping is that the user does not have infinite time. It can be difficult to distinguish between looping or taking a very long time to compute. We thus prefer deciders.

Def A decider is a Turing machine that enters either an accept state or a reject state for every input in A^*.

Def A decider that recognizes some language $L \subset A^*$ is said to decide that language.

Def A language $L \subset A^*$ is called Turing-decidable if $\exists$ a Turing machine M that decides L.

NB Some textbooks use the terminology recursive language instead of Turing-decidable.

Example $L = \{0^m 1^m \mid m \in \mathbb{N}, m \geq 1\}$ is Turing-decidable because the Turing machine we built that recognized it was in fact a decider (check again to convince yourself that machine did not loop.).

Turing-decidable $\Rightarrow$ Turing-recognizable, but the converse is not true: Turing-recognizable $\not\Rightarrow$ Turing-decidable. We will hopefully have time to cover an example of a language that is Turing-recognizable, but NOT Turing-decidable before the end of the term.

$\boxed{\text{Variants of Turing machines}}$

Task Explore variants of the original set-up of a Turing machine and show they do not enlarge the set of Turing-recognizable languages

(A) Add "stay put" to the list of allowable directions

Say instead of allowing just $\{L, R\}$ (the tape head moves left or right), we also allow the "stay put" option (no change in the position of the tape head). Thus, the transition mapping is defined as $t: S \times \hat{A} \to S \times \hat{A} \times \{L, R, N\}$ where N is for "no movement" (stay put) instead of $t: S \times \hat{A} \to S \times \hat{A} \times \{L, R\}$. We realize N is the same as $L+R$ or $R+L$ (move the tape head left by one cell, then right by one cell or the other way around) $\Rightarrow$ variant (A) yields no increase in computational power.

(B) Multitape Turing machine

We allow the Turing machine to have several tapes, each with its own tape head for reading and writing. Initially, the input is on tape 1, and the others are blank. The transition mapping then must allow for reading, writing, and moving the tape heads on some or all of the tapes simultaneously. If k is the number of tapes, then the transition mapping is defined as $t: \underbrace{S \times \tilde{A}^k}_{} \to \underbrace{S \times \tilde{A}^k}_{} \times \underbrace{\{L, R, N\}^k}_{}$

k-fold Cartesian product $\underbrace{\tilde{A} \times \ldots \times \tilde{A}}_{k \text{ times}}$

k-fold Cartesian product $\underbrace{\tilde{A} \times \ldots \times \tilde{A}}_{k \text{ times}}$

k-fold Cartesian product $\underbrace{\{L, R, N\} \times \ldots \times \{L, R, N\}}_{k \text{ times}}$

Since one of the tape heads or more might not move for some transitions, we make use of the option N ("no movement") besides left and right.
Multitape Turing machines seem more powerful than ordinary (single-tape) ones, but that is not the case.

__Def__ We call two Turing machines M_1 and M_2 __equivalent__ if $L(M_1) = L(M_2)$, namely if they recognize the same language.

__Theorem__ Every multitape Turing machine has an equivalent single-tape Turing machine.

__Sketch of proof__ Let $M^{(k)}$ be a Turing machine w/ k tapes. We will simulate it with a single-tape Turing machine $M^{(1)}$ constructed as follows. We add # to the tape alphabet $\tilde{A}$ and use it to separate the contents of the different tapes. $M^{(1)}$ also needs to keep track of the locations of the tape heads of $M^{(k)}$. It does so by adding a dot to the character to which a tape head is pointing. We thus only need to enlarge the tape alphabet $\tilde{A}$ by allowing a version with a dot above for every character in $\tilde{A}$ apart from # and the blank symbol $\sqcup$.

__Corollary__ A language L is Turing-recognizable $\iff$ some multitape Turing machine recognizes L. (r...ch)

__Proof__ "$\Rightarrow$" A language L is Turing-recognizable if $\exists$ M a single-tape

Turing machine that recognizes it. A single-tape Turing machine is a special type of a multitape Turing machine, so we are done.
" $\Longleftarrow$ " follows from the previous Theorem! (q.e.d.)

© A nondeterministic Turing machine

Just like a nondeterministic finite state acceptor, a nondeterministic Turing machine may proceed according to different possibilities, so its computation is a tree, where each branch corresponds a different possibility. The transition mapping of such a nondeterministic Turing machine is given by $t: S \times \tilde{A} \longrightarrow \mathcal{P}(S \times \tilde{A} \times \{L, R\})$

shows we have different possibilities on how to proceed.

Theorem Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Idea of the proof We construct a deterministic Turing machine that simulates the nondeterministic one by trying out all possible branches. If it finds an accept state on one of these computational branches, it accepts the input; otherwise, it loops.

Corollary A language is Turing-recognizable $\Longleftrightarrow$ some nondeterministic Turing machine recognizes it.

Proof " $\Longrightarrow$ " A deterministic Turing machine is a nondeterministic one, so this direction is obvious.
" $\Longleftarrow$ " follows from the previous Theorem.