2 Universal Algorithm; Diagonal Results

2.1 Universal Turing Machine

The first computers were hardware-programmable. To change the function computed, one had to reconnect the wires or even build a new computer. John von Neumann suggested using Turing's Universal Algorithm. The function computed can be then specified by just giving its description (program) as part of the input rather than by changing the hardware. This was a radical idea, since in the classical mathematics universal functions do not exist (as we will see in Sec. 2.2).

Let R be the class of all TM-computable functions: total (defined for all inputs) and partial (which may diverge). Surprisingly, there is a universal function u in R. For any Turing Machine M that computes $f{\in}R$ in time T and space S, u uses a program m of length c listing the commands and initial head state of M. Then $u(mx)$ simulates $M(x)$ in time c^2T and space $S{+}c$. It operates in cycles, each simulating one step of $M(x)$. After i steps of $M(x)$, let s_i be the head's state, l_i be the left from it part of the tape; r_i be the rest of the tape. After i cycles $u(mx)$ has the tape configuration $t_i = l_i m s_i r_i$ and looks up m to find the command corresponding to the state s_i and the first symbol of r_i. It modifies t_i accordingly. When $M(x)$ halts, $u(mx)$ erases the (penciled) ms_i from the tape and halts too. Universal Multi-head TM works similarly but can also determine in time $O(t(x))$ whether it halts in t steps (given $x, t(x)$ and an appropriate program).

Exercise. Design a universal multi-head TM with a constant factor overhead.
Hint: When heads split or merge in the first cell, the room u needs for their programs
creates sparse or dense content regions that propagate right (sparse faster).

We now describe in detail a simpler but slower universal [Ikeno 58] TM U. It simulates any other TM M that uses only $0, 1$ bits on the tape. (Other alphabets are encoded as bit strings for this.) M lacks the blank symbol that usually marks the end of input. Thus input needs to be given in some prefixless form, e.g., with a padding that encodes input length $l = \|x\|$ as a binary number preceded by $2\|l\|$ zeros. In the cells carrying this padding, two counters are initiated that monitor the distance to both ends of the used part of M's tape (initially the input). M's head moving on the tape pulls these counters along and keeps them updated. When the right end of the used tape is reached, any subsequent characters are treated as blanks.

U has 11 states + 6 symbols; see its transition table at the right. It shows the states and tape digits only when changed, except that the prime is always shown. The head is on the tape: lower case states look left, upper case – right. The external choice, halt, etc. commands are special states for M; for U they are shown as "A/B" or =. U works in cycles, simulating one transition of M each. The tape is infinite to the right (the leftward head in the leftmost cell halts). It consist of 0/1 segments, each preceded with a *. Some symbols are primed. The rightmost infinite part of one or two segments is always a copy of M's tape. The other segments describe one transition each: a command $(s, b) \to (s', b', d)$ for M to change state s to s', tape bit b to b' and turn left or right.

	1	0	*	1'	0'	*'
A	f	f	e0			
B	F	F	e1			
f,F	b*	a*	F			c
D	d'	–				e'
E	=	–	'	'	'	e'
d	'	'		'	'	D
b	'	'	'		a'	D
a	'	'	'	b'	F	E'
c	'	'		=	F	E'
e	'	'	'	B	A	A/B

The transition segments are sorted in order of (s, b) and never change, except for priming. Each transition is represented as $*Sdb$, where b is the bit to write, d the direction $R{=}0/L{=}1$ to turn. S points to the next state represented as 1^k, if it is k segments to the left, or 0^k (if to the right). Each cycle starts with U's head in state F or f, located at the site of M's head. Primed are the digits of S in the prior command and all digits to their left. An example of the configuration: $*'0'0'0'1'0' *' 0'0'0'0'01 * 011 * \ldots * 00\ \boxed{\text{head}}\ 00 \ldots$

U first reads the bit of an M's cell changing the state from F or f to a/b, puts a * there, moves left to the primed state segment S, finds from it the command segment and moves there. It does this by repeatedly priming nearest unprimed * and 1s of S (or unpriming 0s) while alternating the states c/F or d/D. When S is exhausted, the target segment $\|S\| + b$ stars away is reached. Then U reads (changing state from e to A or B) the rightmost symbol b' of the command, copies it at the * in the M area, goes back, reads the next symbol d, returns to the just overwritten (and first unprimed) cell of M area and turns left or right. As CA, M and U have in each cell three standard bits: present d and previous d' pointer directions and a "content" bit to store M's symbol. In addition U needs just one "trit" of its own!