Consequences of the Universal RAM Simulator

For every program $P$ written in a (known) HI, we can build a DTM $M_P$ s.t. for all $x$ on stdin $M_p(x)$ has an emulating computation of $P(x)$.

Proof: 

1. Compile $P$ to object code $O_P$ in "my mini-assembly".
2. Design $M_P$ with an "initial bank of states" that:
   - First copy $x$ as $[x]$ on Tape 3 of $U$.
   - Use $|O_P|$ special one-off states to overwrite $x$ on tape 1 with the textual code of $O_P$'s instructions.

Now we have the initial setup for $U$ running $O_P$ on $x$ so go to the start state of $U$. $M_P = \text{Init} \xrightarrow{?} \text{U}$

$M_p(x) = U(P,x) = P(x)$  Size $\propto |O_P| + |U| \times |P| + \text{ca}$

- The time used by a computation is its # of steps.
- The space used is the # of cells in which a char was changed to a different char.
Theorem [Steve Cook, 1971]

There is a clever way to build \( U \) so that every \( t \) steps of \( P(x) \) judged at "fairros" is simulated by \( O(t^2) \) steps of \( U \).

Ideas Involved. Suppose we get input \( x = x_1, x_2, \ldots \) left-justified on a tape and want to prepend a 1

\[
\begin{array}{cccc}
1 & x_1 & x_2 & \cdots & x_n \\
\wedge & x_1 & x_2 & \cdots & x_n
\end{array}
\]

Furthermore, this "shift over" routine can be called to make more room in the middle of a tape, e.g. to write to a "register" of \( U \).

We can attach a separate copy of "sh.over," entered on *1* at any state of \( A \). Can picture it like a Unix "daemon" (or vector gives the \( O(t^2) \)).
A TM with 3 (or any number $K$) tapes can be simulated by a one-tape TM in time $O(t^2)$.

Go back to $\wedge$ (execute each one in $K$ scanned chains, read $L \to R$ sweep and ask $r'$ as $\pi'(\pi r)^K$ possible $c_1 \ldots c_K$.

Overhead scales as $\#$ steps already taken (time is $O(t^2)$, space is same.

Added: The notes mention that Sipser's text gives a different method that manages $K$ regions on one tape (like my "registers" but separated by # marker) that uses a similar "shrink-over" routine and (hence) is less efficient as well as messy.