More unsolvable problems..

why are we interested in problems which can't be solved?

..we should know fundamental results in Computing as part of education in the field

….and avoid trying to solve them!

..we should understand that these results are independent of future hardware developments

..we learn methods applicable to more unsolved problems

GÖDEL's Incompleteness Theorem

"there is no Turing Machine which prints out all true statements "sentences" about arithmetic and no false ones."

The language of arithmetic has

- function symbols + and .
- successor function S
- the relation symbols < and =
- the constant symbol 0
- variables v, v', v''..(infinitely many)(symbols are v and ')
- connectives □ (and), ∨ (or), → (implies), ¬ (not), ↔ (iff)
- quantifiers ∀, ∃
- brackets (and)

e.g. $\Box(x) =_{\mathbf{def}} (x>S0) \rightarrow (\forall y \forall z (x = y.z \rightarrow y=x \vee z=x))$

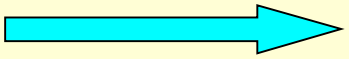
Method of proof of Gödel's incompleteness theorem..

We reduce the Halting Problem to the Gödel machine problem..

express a run of a Turing Machine as an arithmetic formula describing conditions which must be satisfied for there to be a successful sequence of configurations (a run ending with a Halting State) of the TM.

We run the hypothetical Gödel Machine and wait for it to produce this formula, or the negation of it, as a true statement of arithmetic.

..that is, whether the TM on a given input will halt or not..
but this is a solution to the Halting Problem..impossible.



the assumption of the Gödel machine is invalid
..there is NO Gödel Machine

Deciding whether a sentence of first-order predicate logic is valid or not..

Church showed that an algorithm to do this could be modified to print out all true statements of arithmetic and no false ones
i.e. he reduced **Gödel's incompleteness problem** to this..

as we know Gödel's theorem has been disproved,
we deduce that this is unsolvable too.

Post's Correspondance Problem.
Given words $a_1,a_2,..a_n$, $b_1,b_2,..b_n$ of C….
is there a non-empty sequence i(1),i(2),..i(k) of numbers ≤n such that $a_{i(1)}$, $a_{i(2)}..a_{i(k)}$ and $b_{i(1)}$, $b_{i(2)}..$ $b_{i(k)}$ are the same or not. There is no algorithm to decide the general case…can be shown by reducing HP to this problem.

Turing Machine Summary

We defined a TM, M = (Q, ∑, I, q0, □, F),

Where:

Q finite, non-empty set of states

∑ finite set of at least 2 symbols: the alphabet. ^ □ ∑

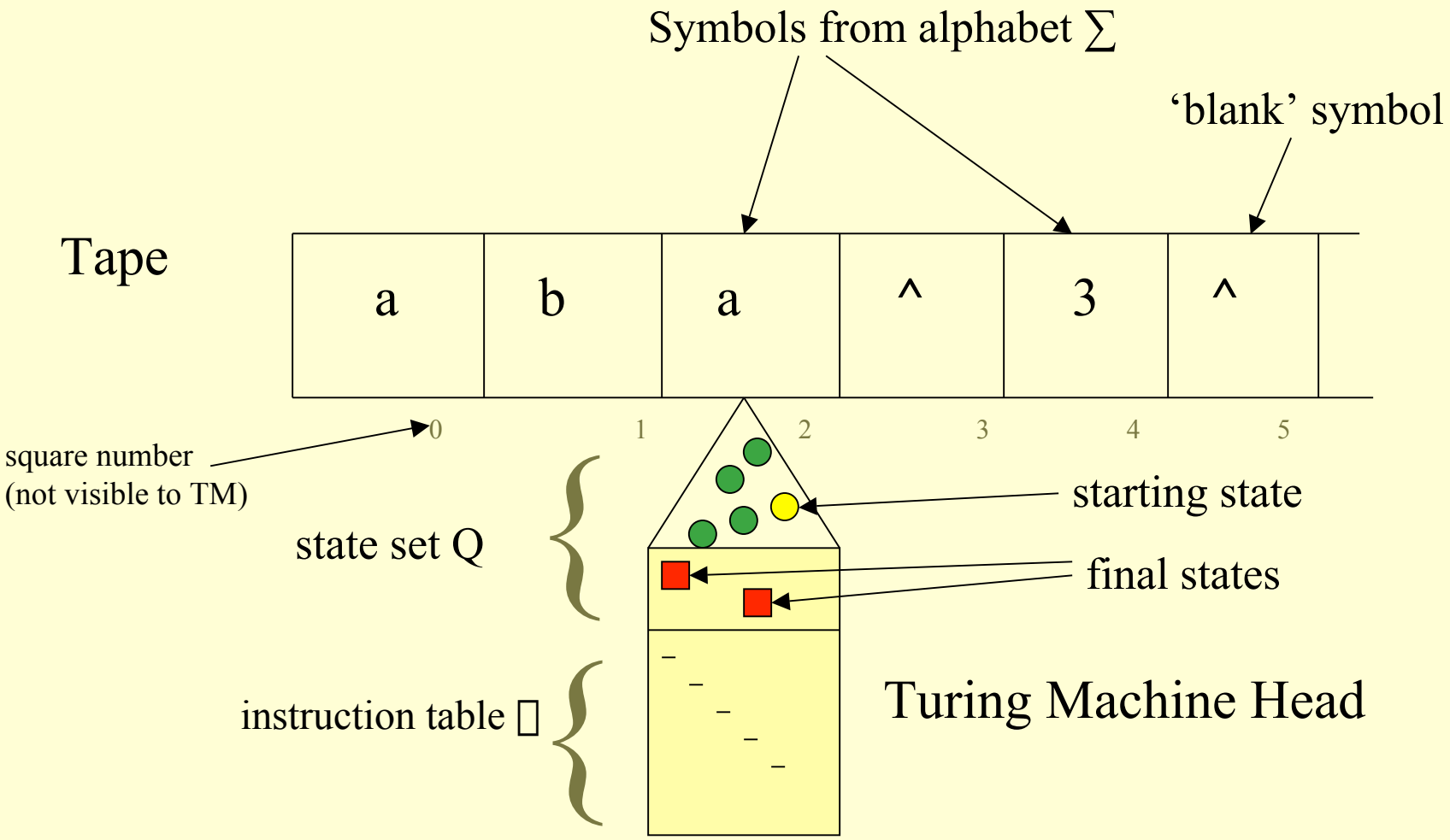
I non-empty subset of ∑; ^ □ I; input alphabet

q_0 q_0 □ Q; starting or initial state

□: (Q\F) x ∑ □ Q x ∑ x {-1, 0, 1}, a partial function,
the instruction table

F F □ Q, the set of final or halting states

A Turing Machine



The Church-Turing Thesis: **a problem is algorithmically solvable iff it can be solved by a Turing Machine**

evidence (not proof) for this:

- intuition
- very large set of examples
- equivalence to other formalisms (eg Church's).

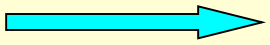
we looked at ways we might extend the capacity of the defined TM:

- multiple tracks
- holding finite amount of data in a state

and found these were just programmers' conveniences - they did not extend the TM-solvable problems

and at hardware extensions:

- 2-way infinite tape
- multiple tapes
- 2-dimensional tape..and showed no extension to computability



support for the Church-Turing thesis

We defined the **Universal Turing Machine, U**

- the first programmable computer

which takes the code of a standard TM, S, and an input word w, and produces the same outcome as S running on input w.

Could we use U to tell us whether an arbitrary standard TM Halts and Succeeds or not?

..is there any TM which could do this..solve the Halting Problem?

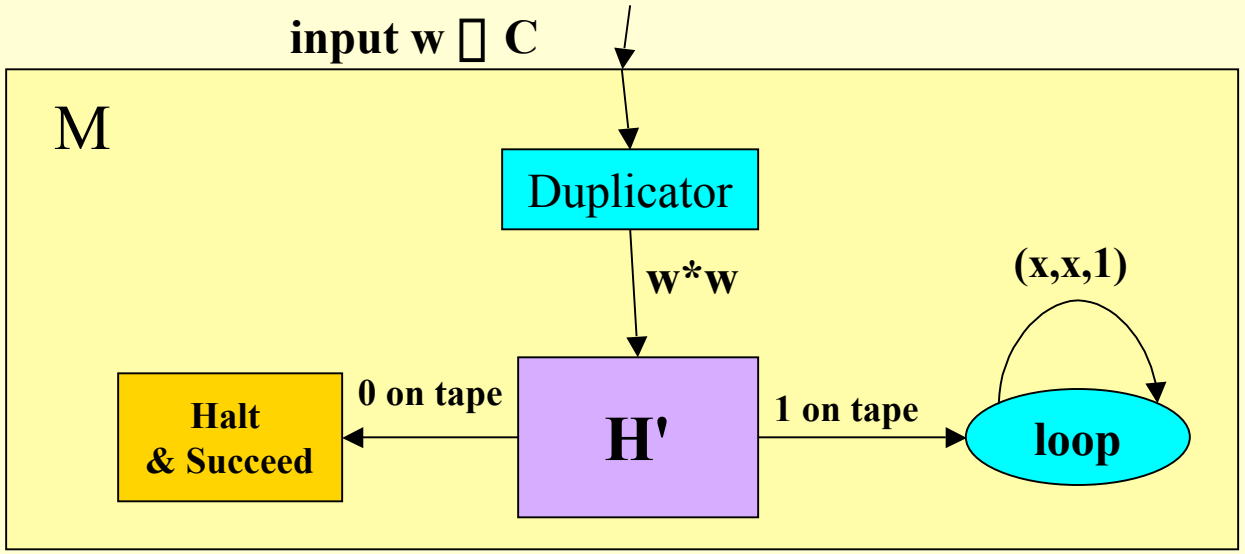
We proved directly that **there is no TM to solve the Halting Problem**

and derived other unsolvability results by reduction of HP (showing that a solution of the new problem could be used to solve HP..already shown to be unsolvable)…reduction of HP to the new problem.

Later in the course - **non-deterministic TMs**

Tutorial Sheet 5 - Qu 1

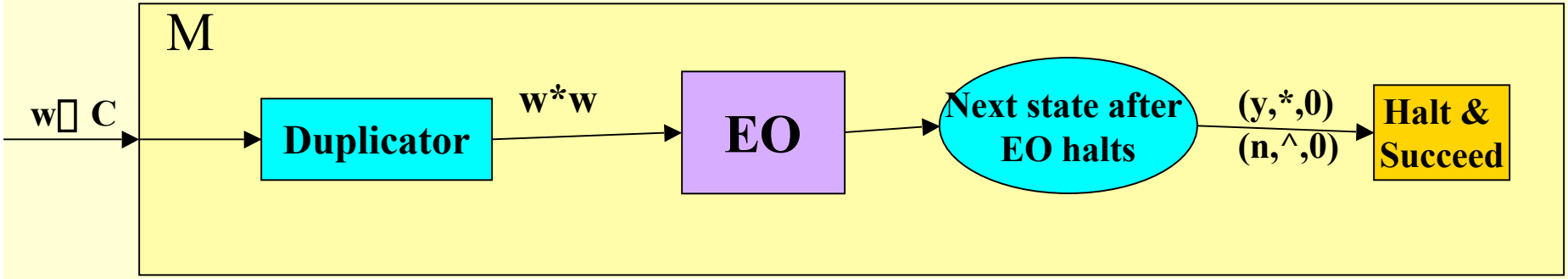
Assume (to get a contradiction proof) that H' exists.
Build M:



By elimination of scratch characters we can assume M is standard, so M has a code; input code(M) to M:
M runs forever (moving right)
iff f_H(code(M)*code(M)) = 1
iff M halts on input code(M) [defn of H']…**Contradiction**
so H' does not exist

Tutorial Sheet 5 - Qu 2

Assume (to get a contradiction) that EO exists. Build M as below



By elimination of scratch characters we can assume M is standard, So it has a code.

$f_M(\text{code}(M)) = *$ iff $f_{EO}(\text{code}(M)*\text{code}(M)) = y$

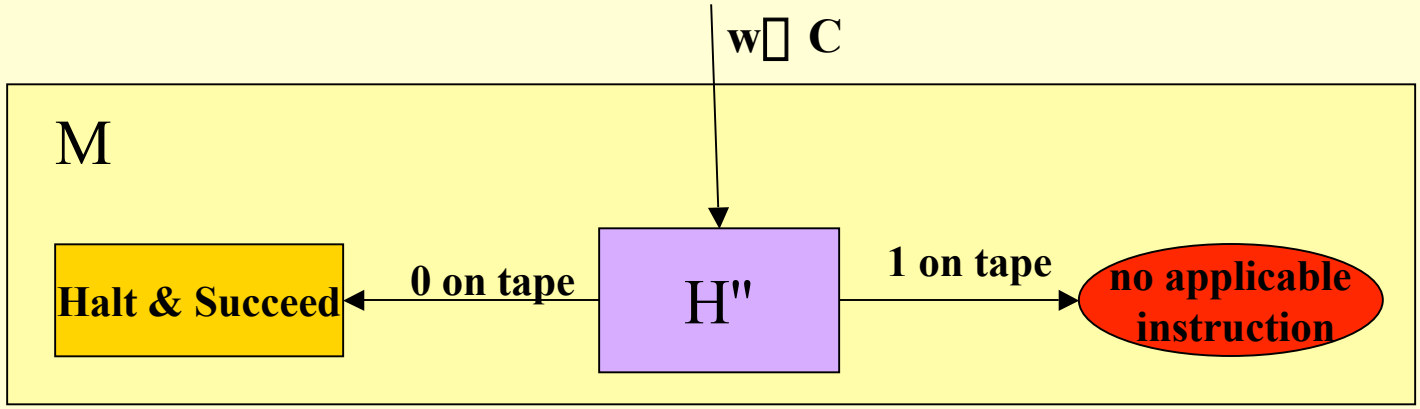
iff $f_M(\text{code}(M)) =$ □

Contradiction

So EO does not exist.

Tutorial Sheet 5 - Qu 3

If H" exists, consider the following TM, M:



The duplicator is not needed (see defn of $f_{H"}$) . We can assume M is standard by elimination of scratch characters, so it has a code.

M Halts and Fails on input code(M)

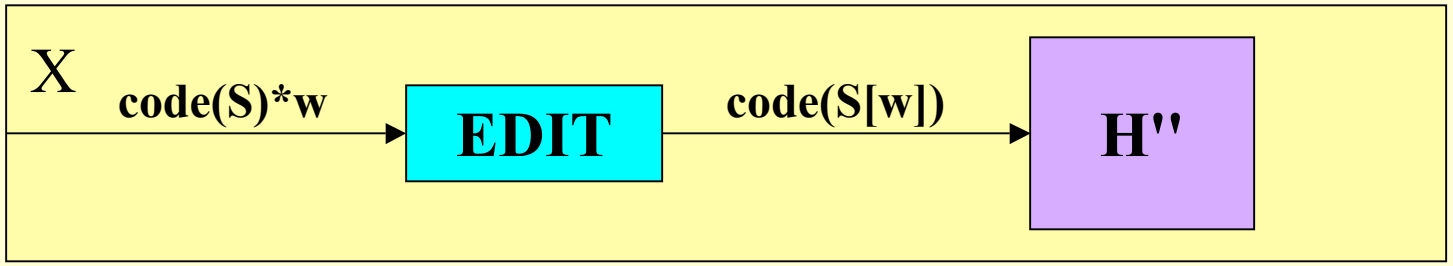
iff $f_{H"}$(code(M)) = 1

iff M Halts and Succeeds on input code(M).

Contradiction there is no such H"

Tutorial Sheet 5 - Qu 4

..we build the TM X:



X solves the Halting Problem: ..if S is standard, and w ∈ C, then
$f_X(\text{code}(S)*w)) = f_{H''}(\text{code}(S[w]))$.
By definition of H'' this is 1 if $f_{S[w]}(\text{code}(S[w]))$ is defined
0 if not defined.
But $f_{S[w]}(\text{code}(S[w])) = f_S(w)$.
So $f_X(\text{code}(S)*w) = 1$ if $f_S(w)$ is defined, and 0 if not. ..X solves HP
but HP has no TM solution..H'' cannot exist.