H11.1 Prove, by using Rice's theorem, that the decision problem

> Given an arbitrary Turing machine M, is the language recognised by M finite?

i.e., the language

$$\{c_M \mid M \text{ is a Turing machine and } \mathcal{L}(M) \text{ is finite}\}$$

is undecidable. Define the semantic property you apply precisely, and explain why it is not a trivial one.

Semantic property S is a collection of languages that has such property S. Let's define the semantic property as the set of languages that are finite, called FINITE.
The corresponding semantic property is FINITE = { L ∈ RE | L is finite }

- Finite languages are those having a finite number of accepted strings. As we know, all finite languages are regular. If you have a finite set of strings that your languages matches, you can simply use alternation (string1|string2|...) to construct a regular expression to match them, or construct a finite automaton in a straightforward manner.
For example, a finite language can be constructed as follows:
L = { x | x ∈ (cat | dog | bird) }
- Note: It is not true that all regular languages are finite. Even something as simple as 'a*' is a regular expression that matches an infinite set of strings: '' (the empty string), 'a', 'aa', 'aaa', ...

Therefore, the property FINITE is non-trivial because:
FINITE ≠ ∅ (witness a semi-decidable language L ≠ ∅ proved by the example above)
FINITE ⊂ RE (since ∅ ∈ RE/NE)
Thus by Rice's theorem, the language
codes(FINITE) = { c_M | M is a Turing Machine and ℓ(M) ∈ RE }
= { c_M | M is a Turing Machine and ℓ(M) ≠ ∅ }
is undecidable (proven)

H11.2 Design an unrestricted grammar that generates the language

$$\{ww \mid w \in \{a, b\}^*\}.$$

Show how the string *abab* can be derived in your grammar.

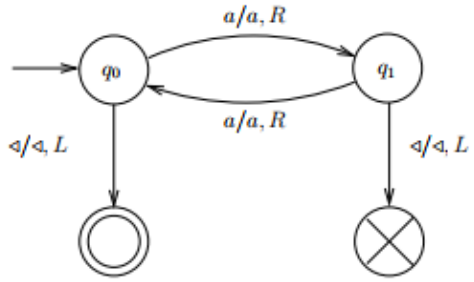
What we need to do is to generate ww^R and then, carefully, reverse the order of the characters in w^R. What we'll do is to start by erecting a wall (#) at the right end of the string. Then we'll generate ww^R. Then, in a second phase, we'll take the characters in the second w and, one at a time, starting with the leftmost, move it right and then move it past the wall. At each step, we

move each character up to the wall and then just over it, but we don't reverse characters once they get over the wall. The first part of the grammar, which will generate $wTw_R\#$, looks like this:
S → S1#
S1 → aS1a
S1 → bS1b
S1 → T
This inserts the wall # at the right. T will mark the left edge of the portion that needs to be reversed. At this point, we can generate strings such as abbbTbbba#. What we need to do now is to reverse the string of a's and b's that is between T and #. To do that, we let T spin off a marker Q, which we can pass rightward through the string. As it moves to the right, it will take the first a or b it finds with it. It does this by swapping the character it is carrying (the one just to the right of it) with the next one to the right. It also moves itself one square to the right. The four rules marked with * accomplish this. When Q's character gets to the # (the rules marked **), the a or b will swap places with the # (thus hopping the fence) and the Q will go away. We can keep doing this until all the a's and b's are behind the fence and in the right order. Then the final T# will drop out.
Here are the rules for this phase:
T → TQ
Qaa → aQa *
Qab → bQa *
Qbb → bQb *
Qba → aQb *
Qa# → #a **
Qb# → #b **
T# → ε

- Derivation of the string abab

S → S1# → aS1a# → abS1ba# → abTba# → abTQba# → abTaQb# → abTa#b → abTQa#b → abT#ab → abεab → abab

H11.3 Use the construction described at Lecture 11 to design an unrestricted grammar whose derivations simulate the computations of the following simple Turing machine that recognises the language $\{a^{2k} \mid k \geq 0\}$.



Give a derivation for the string *aa* in your grammar, and explain why the string *aaa* cannot be derived in it.

The unrestricted grammar is as follows:
$S \rightarrow LT \mid \varepsilon$
$T \rightarrow AAT \mid AA$
$LA \rightarrow a$
$aA \rightarrow aa$
Derivation of the string aa:
$S \rightarrow LT \rightarrow LAA \rightarrow aA \rightarrow aa$
- The reason the string aaa cannot be derived in it is because of $T \rightarrow AAT \mid AA$, which means T will always produce a pair of A. Since aaa is an odd number of a, this string does not belong to this language