VALID COMPUTATIONS: This is a tool for establishing many undecidability results about context-free languages. Here is the headline for this section:

⇛ Given a TM and word $\langle M, w\rangle$ we can effectively construct a PDA (or a grammar) which recognizes (resp. generates) the COMPLEMENT of the set of valid computations of M on w. ⇚

What does "effectively construct" mean? It means that we can describe the PDA without knowing in advance whether M halts on w. What is a valid computation?

This requires a long answer but the short answer: it is a string which describes all the steps that M makes as it processes w. If M does not halt on w there is no such string. If M does halt on w there is at least one such string. If M is nondeterministic there may be many such strings.

Long answer: (1) A configuration of a TM is a description of its state, its ~~~~ tape & the head position. We describe this as follows: suppose the tape contains $abbaab$, the state is q and the head is on the third square of the tape we write $abqbaab$. The name of the state is written just to the left of the symbol which the head is looking at.

(2) To define a computation we need a separation symbol $\# \notin \Gamma$ (Γ is the tape alphabet). We assume $Q \cap \Gamma = \emptyset$ & $\# \notin Q$.

Suppose $\delta(q, b) = (q', a, R)$ then we have the transition $abqbaab \rightarrow abaq'aab$

The q has become a q', the b has changed to a & the head has moved one step to the right.
We write consecutive configurations of the TM next to each other separated by # as follows

$$\# \cdots \# \cdots \# \, a b q b a a b \# \; a b a q' a a b \# \cdots$$

The start configuration looks like

$$\# q_0 a_1 \cdots a_n \#$$

where $\omega = a_1 \cdots a_n \in \Sigma^*$

A <u>valid</u> <u>computation</u> for M, ω is a sequence of configurations ~~whose this~~

$$\# \alpha_0 \# \alpha_1 \# \alpha_2 \# \cdots \# \alpha_N \#$$

where: (i) α_0 is the start configuration
(ii) α_N is a halting configuration, i.e. the state is either q_a or q_r (iii) α_{n+1} follows from α_n according to the transition table of the Turing machine

We call this VALCOMPS(M, ω)

If M does not halt on ω then there are no valid computation & VALCOMPS$(M, \omega) = \emptyset$.

Let $\Delta = \Gamma \cup Q \cup \#$.

Δ is the alphabet with which we describe the computations of the TM; it is <u>NOT</u> the alphabet of the TM itself. Now if VALCOMPS$(M, \omega) = \emptyset$

$$\overline{\text{VALCOMPS}(M, \omega)} = \Delta^*.$$

Here is the main point: <u>we can describe a</u> <u>PDA</u> P such that P accepts $\overline{\text{VALCOMPS}(M, \omega)}$ without knowing in advance whether ~~VALC~~ M halts on ω!!

Equivalently: <u>we can describe a CFG G s.t.</u>

$$L(G) = \overline{\text{VALCOMPS}(M, \omega)}.$$

If we can decide whether $L(G) = \Sigma^*$ we have solved the halting problem. If $L(G) \neq \Sigma^*$ then $M(\omega)\downarrow$. If $L(G) = \Sigma^*$ then $M(\omega)\uparrow$.

I will now describe the PDA, this will complete the reduction

$$\neg H_{TM} \leq_m \{ \langle G \rangle \mid L(G) = \Sigma^* \}$$

Here are the conditions that VALCOMPS(M,ω) must satisfy:
If $z \in$ VALCOMPS(M,ω)

(a) z begins and ends with # and between each successive pair of # we must have a non-empty string over the alphabet $\Delta \setminus \{\#\}$. $z = \#\alpha_0\#\alpha_1\#\cdots\#\alpha_N\#$
(b) Each α_i must contain exactly one letter from Q
(c) α_0 must be the start configuration
(d) α_N must be a halt configuration
(e) For each i $\alpha_i \rightarrow \alpha_{i+1}$ according to the rules of the TM transition table.

Now conditions (a), (b), (c) & (d) can be checked by a DFA. Thus we can safely ignore them as we can easily run these DFAs in parallel with our PDA. If any of these first 4 conditions are violated we accept the string (Recall we are checking $\overline{\text{VALCOMPS}(M,\omega)}$).
Now for case (e) which I will sketch:
the crucial idea $\alpha_i \rightarrow \alpha_{i+1}$ means that α_i & α_{i+1} can differ only in a window of 3 symbols near the head position. For example if $\delta(q,a) = (p,b,L)$

$$\# \, ab\underline{aqa}bba \, \# \rightarrow ab\underline{pab}bba$$

In the valid computation this would look like
$\cdots\# \, ab\underline{aqa}bba \, \# \, ab\underline{pab}bba\#\cdots$
I have underlined the 3 symbol window where they differ. Notice outside this window the symbols are identical. We call two pairs of 3 symbol sequences <u>consistent</u> if (i) they are identical &

neither 1 contains the head OR (ii) one or both contain the head & they differ according to the rules of the transition table of the TM.

There are only finitely many possible pairs of such consistent sequences & they can be remembered in the finite-state ~~mem~~ memory of the PDA. We need the stack to find the corresponding position in 2 consecutive configurations

$$\cdots \# \underbrace{\qquad}_{w_1} \overbrace{xxx} \underbrace{\qquad}_{w_2} \# \overbrace{\qquad}^{w_1'} \overbrace{YYY} \overbrace{\qquad}^{w_2'} \# \cdots$$

The PDA is looking for an <u>invalid</u> string so it just has to guess <u>one place</u> where condition (e) is broken. It stacks w_1 on its stack, it remember the string xxx in its memory it ignores w_2 & goes to the next #. Then it pops its stack while reading w_1' so it correctly finds the corresponding position & compares xxx with YYY. If they do <u>NOT</u> match it accepts.

WE ARE DONE!

Some points to remember:

1. Since we are looking for something that is <u>not</u> valid we only have to find <u>one</u> place where it is not according to the rules in (a) – (e). It is not possible to check VALCOMPS(M,w) with a PDA because all rules must be respected everywhere.
2. You might be confused by the logic: how did we construct the PDA P when the HP is undecidable? Can't we use P to solve the HP by asking it to check if a given string is a valid computation? No! This will only tell you about one particular

string. In order to know $M(\omega)\downarrow$ we need to know whether $L(G) = \Delta^*$ or not.

(3) We have an <u>algorithm</u> to answer $L(G) = \emptyset$? Can't we flip that and solve the halting problem? No! The opposite of $L(G) = \emptyset$ is $L(G) \neq \emptyset$; this is not the same as $L(G) = \Sigma^*$.

Our reduction is

$$\neg HP \leq_m \{\langle G\rangle \mid L(G) = \Sigma^*\}$$

So if we could solve $L(G) = \Sigma^*$ we can solve the complement of the Halting Problem.

We can define a different type of valid computations. we call them VALCOMPS2(M, ω):

$$\# \alpha_0 \# \alpha_1^{REV} \# \alpha_2 \# \alpha_3^{REV} \# \cdots \# \alpha_N^{\odot} \#$$

The $\odot$ at the end means reverse if N is odd. Now with 2 PDAs working independently we can have one check $\alpha_i \Rightarrow \alpha_{i+1}$ for i odd & the other one checks $\alpha_i \rightarrow \alpha_{i+1}$ for i even. If both say OK we have a string in VALCOMPS2(M, ω). The fact that consecutive configurations are reversed makes it easy for a stack machine to check the entire configurations. We need two machines because one of them cannot check off both the even & the odd case. Isn't this giving us the power of a 2 stack machine (AKA Turing machine)? No! The two PDA's are <u>independent</u>.

VALCOMPS 2 $(M, \omega) = L(G_1) \cap L(G_2)$ for two grammars. We have shown (in outline)

$$HP \leq_m \{\langle G_1, G_2\rangle \mid L(G_1) \cap L(G_2) \neq \emptyset\}.$$

Summary of results

1. Given a grammar G, is $L(G) = \Sigma^*$?
2. Given 2 grammars G_1, G_2 is $L(G_1) \cap L(G_2) \neq \phi$?

Neither ~~one~~ is computable (decidable).

1 is coCE & not CE

2 is CE & not coCE.

Why is (1) coCE? Given a grammar G we have an algorithm which <u>always</u> <u>terminates</u> s.t. it answers $w \in L(G)$? for any $w \in \Sigma^*$. So we keep trying all the words in Σ^*. If one of them is rejected we will eventually find out so we can always give a <u>NO</u> answer if the answer is indeed "no". However if the answer is YES we will never find out for sure.

Convince yourself that 2 is CE.