Theorem 7.27 is stated on page 300, but Sipser doesn’t give a proof there, because the proof uses a method (polynomial-time reduction) and a new concept (NP-completeness) that require quite a bit of explanation and some theorems of their own. Once he’s filled in all of this background, Theorem 7.27 has been reduced to an immediate corollary of Theorem 7.37, with a proof so trivial that Sipser doesn’t even bother to present it. Instead, we get a single sentence: “This theorem implies Theorem 7.27.”

Recall that the statement of the theorem is a Boolean equality: \( \text{SAT} \in \text{P} \iff \text{P} = \text{NP} \). In case the missing proof isn’t quite as obvious as Sipser supposes, here it is:

Suppose first that \( \text{SAT} \in \text{P} \). Then, since \( \text{SAT} \) is NP-complete (Theorem 7.37), \( \text{P} = \text{NP} \), by Theorem 7.35.

Conversely, suppose that \( \text{P} = \text{NP} \). By Theorem 7.37, \( \text{SAT} \) is NP-complete; hence, by Definition 7.34, \( \text{SAT} \in \text{NP} \). Therefore, \( \text{SAT} \in \text{P} \), as required.

It’s a little strange to use higher-numbered definitions and theorems to prove a lower-numbered one, but in this case there is no circularity in the reasoning, since none of the premisses (Definition 7.34 and Theorems 7.35 and 7.37) depends on Theorem 7.27, either directly or indirectly. The proof is correct—it’s just that there is an unusually large gap between the point at which the theorem was stated and the point at which it was proven.

In Definition 7.29, note that \( f \) doesn’t qualify as a polynomial-time reduction unless the implication between ‘\( w \in A \)’ and ‘\( f(w) \in B \)’ holds in both directions. To prove that \( A \leq \text{P} B \), one must find a function \( f \) that not only maps members of \( A \) to members of \( B \) but also maps every non-member of \( A \) to a non-member of \( B \).

In the proof of Theorem 7.31, it may not be apparent why Sipser emphasizes (in the last sentence of the proof) that the composition of two polynomials is a polynomial. This is relevant because \( M \) runs in polynomial time as a function of the length of its input, which in this case is \( f(w) \). The running time for \( M \) as a function of the length of the original input \( w \) is bounded above by the composition of the running-time function for the Turing machine \( M_f \) that computes \( f \) and the running-time function for \( M \), since the length of \( f(w) \) cannot be greater than the number of transitions that \( M_f \) makes as it computes \( f(w) \).

The proof of Theorem 7.32 makes it clear that it’s not necessary for \( f \) to be a surjection in order for it to be a valid polynomial-time reduction. In reducing a problem \( A \) to a problem \( B \), we often don’t need the full generality of \( B \); it’s okay if we just take advantage of a few special cases. In this proof, the graphs that \( f \) produces as components of its outputs all have a particular structure that is not representative of all graphs that have to be considered in an algorithm that decides \( \text{CLIQUE} \). This makes no difference. We do have to consider every possible cnf-formula and show that \( f \) maps
all the satisfiable ones to members of $CLIQUE$, but maps all of the unsatisfiable ones to non-members of $CLIQUE$, but we don’t have to consider any of the graphs that $f$ never generates.

At first glance, Definition 7.34 sets a standard that seems almost impossible to meet. If proving that a language is NP-complete involves proving that every language in NP is polynomial-time reducible to it, we certainly can’t go through the languages in NP one by one and find a polynomial-time reduction for each one. We’d have to find a general strategy that would work for any language, without making any assumptions about that language except that a nondeterministic Turing machine can decide it in polynomial time. That must be quite a strategy!

Theorem 7.36 assures us that at least we’ll only have to find such a strategy for one language $B$. If we can manage to prove that $B$ is NP-complete, we can leverage that knowledge to prove that some other language $C$ is also NP-complete, just by showing that $C \in NP$ and then finding a polynomial-time reduction from $B$ to $C$.

But we can’t use that leverage unless we have a language that we already know to be NP-complete to use as a starting point for reductions. So for the Cook-Levin Theorem, Theorem 7.37, we actually have to do the very difficult work of finding a general strategy for reducing every language in NP to the chosen starting point (SAT) in polynomial time.

As in the proof of Theorem 7.20, Sipser uses $n^k$ as a stand-in for the time-complexity function $t(n) \in O(n^k)$ for the nondeterministic Turing machine $N$ that decides the reduction-candidate language $A$. In a more rigorous presentation of the proof, the tableau would have $t(n) + 1$ rows, one for each configuration of $N$ as it processes the worst-case input $w$ of length $n$, and $t(n) + 3$ columns, one for each symbol in a configuration that $N$ might reach by making $t(n)$ transitions (since in the worst case it might move the tape head rightwards on every transition), and two more for the endmarker symbols # to enclose the configuration. The total number of cells in the tableau is $t(n)^2 + 4t(n) + 3$, which is still, clearly, a polynomial function of $n$.

When the variables $x_{i,j,s}$ are introduced (in the third paragraph on page 306), $i$ would then range over all the possible row numbers, from 1 to $t(n) + 1$, and $j$ over all the possible column numbers, from 1 to $t(n) + 3$. The upper bounds for the iterated AND and OR operations that range over $i$ and $j$ should be adjusted similarly.

If the nondeterministic Turing machine $N$ has a large tape alphabet and a large number of states, the number of legal windows for the tableau can be breathtakingly large. This consideration doesn’t figure into the estimate of the size of the Boolean formula $\phi$, however, because $N$ has the same number of states and the same number of symbols in its tape alphabet regardless of how long its input $w$ is. In other words, since we’re estimating the size of $\phi$ as a function of $|w|$, the number of legal windows is treated as a (possibly extremely large) constant. It might affect the coefficient on the highest-order term of the function that relates the length of $\phi$ to $|w|$, but it does not affect its degree of that function or its polynomial nature.
The situation is exactly the same with the conversion of each conjunct within $\phi_{\text{move}}$ to conjunctive normal form in the proof of Corollary 7.42: In each case, we wind up with a Boolean formula that may be longer than the one it replaces but is logically equivalent to it. For the proof that the length of $\phi$ is still a polynomial function of $|w|$, however, the key observation is that nothing in the original OR-of-ANDs subformula, the one in the lower display on page 309, depends on $w$. Consequently, the only effect of converting it to conjunctive normal form by applying the distributive law of conjunction over disjunction is to increase the coefficient on the leading term of the function that relates the length of $\phi$ to $|w|$. Once again, the proposed transformation does not affect its degree of that function or its polynomial nature.