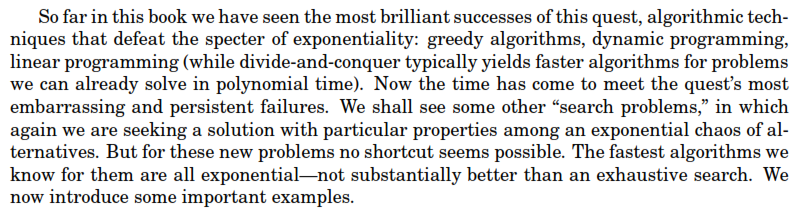
Complexity Theory 2

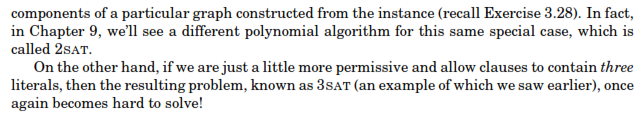
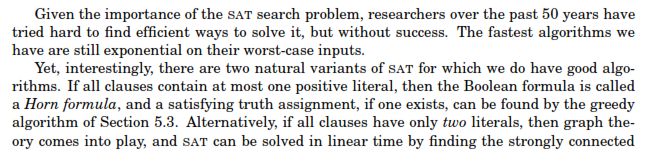
The Algorithm Design studies various techniques for devising efficient algorithms. Most of these are O(n), O(logn), O(n2), etc. – of polynomial efficiency.

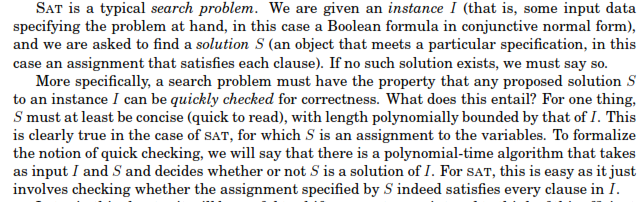
To better appreciate such efficient algorithms, consider the alternative: In all these problems we are searching for a solution (path, tree, matching, etc.) from among an exponential population of possibilities. Indeed, n boys can be matched with n girls in n! different ways, a graph with n vertices has nn-2  spanning trees, and a typical graph has an exponential number of paths from s to t. All these problems could in principle be solved in exponential time by checking through all candidate solutions, one by one (brute-force approach). But an algorithm, whose running time is exponential or worse, is all but useless in practice. The quest for efficient algorithms is about finding clever ways to bypass this process of exhaustive search, using clues from the input in order to dramatically narrow down the search space.

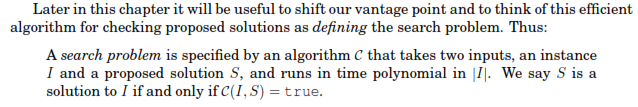
[Dasgupta 233-245]



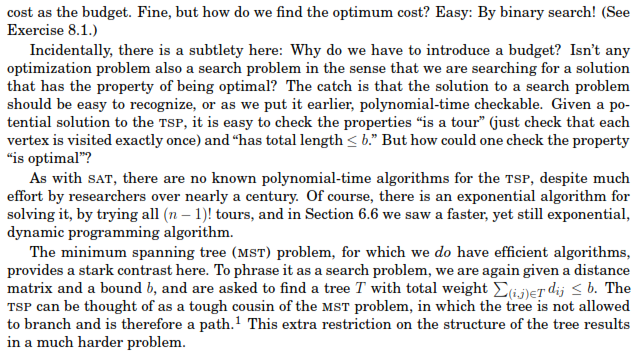
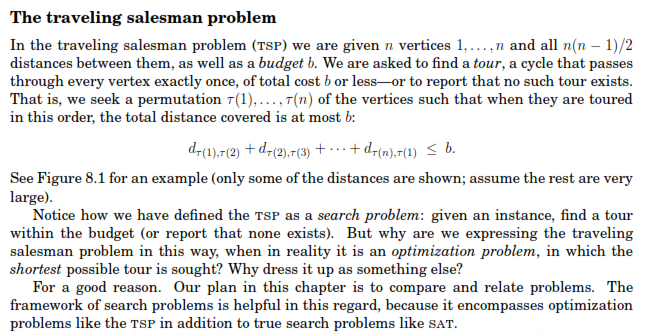




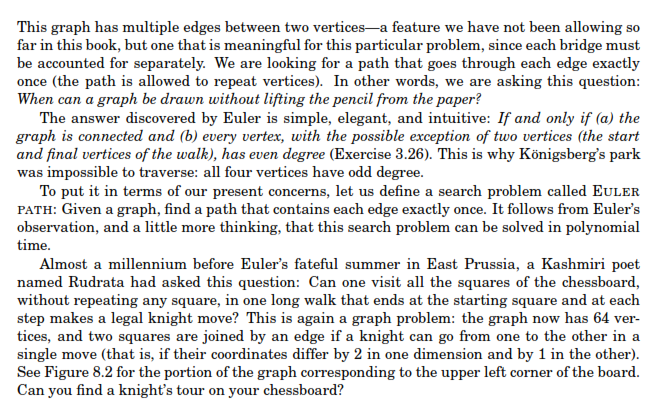
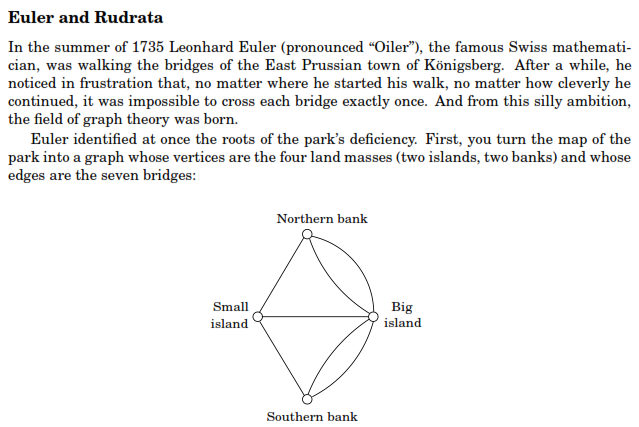
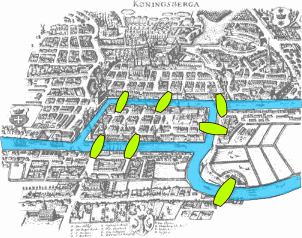


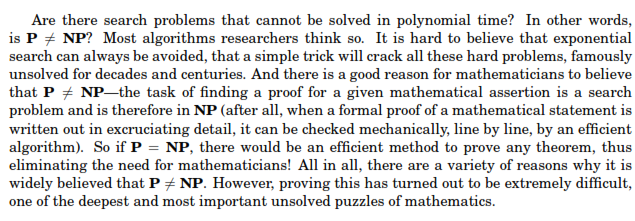
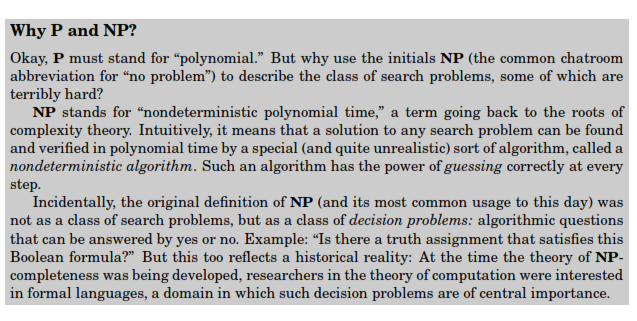
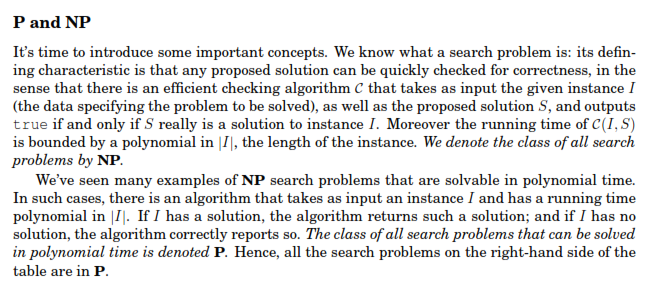
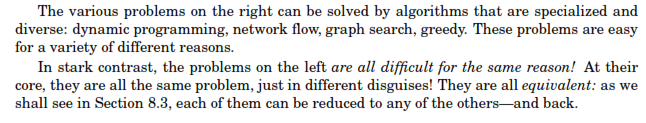
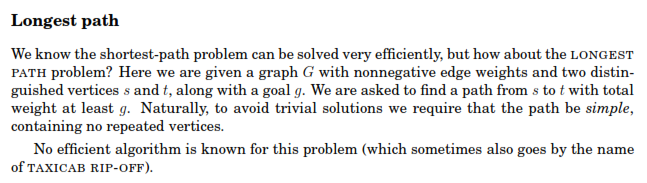
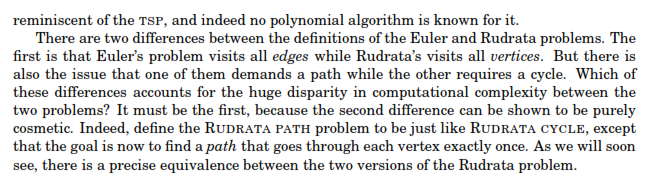
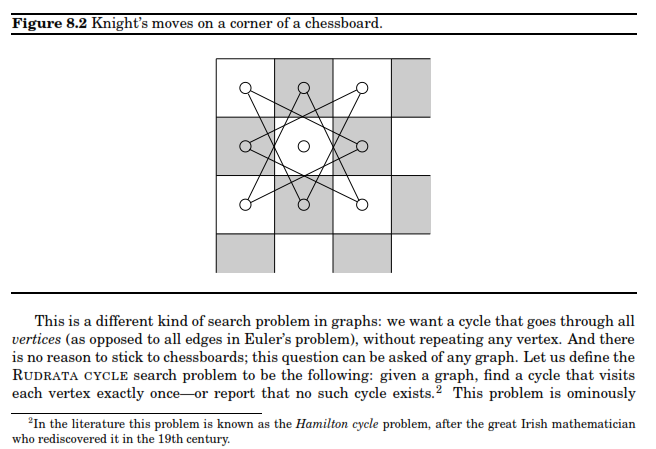


Equivalent terms for solution: witness, certificate, evidence.



We consider **search** problems

[Dasgupta 233-245]



2-color

3-color

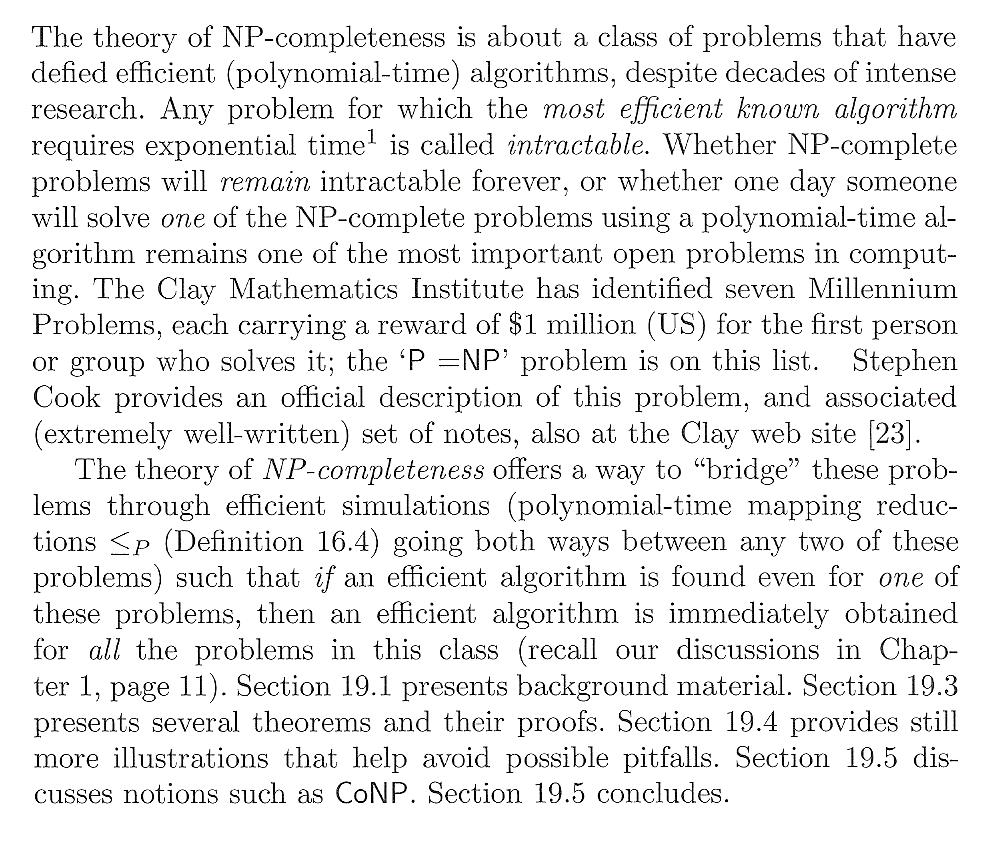
These problems are also in NP

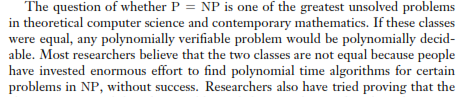
Taxicab rip-off

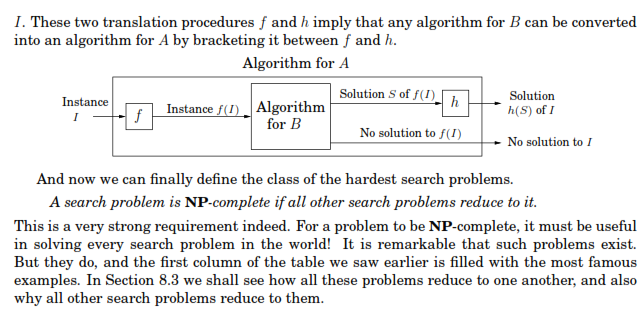
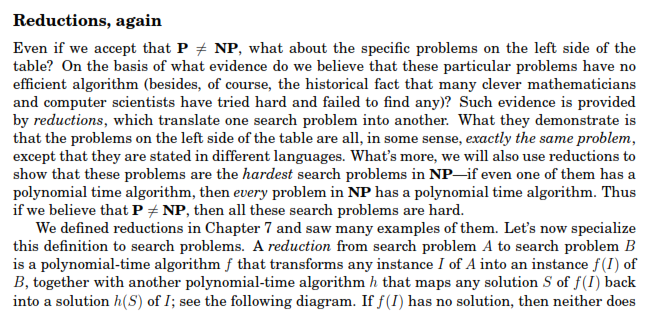
Hamiltonian cycle

**NP** - Only **checks** a solution

**P - Finds** a solution



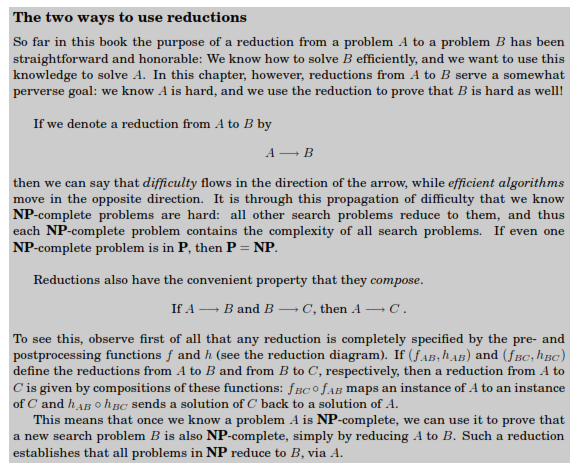


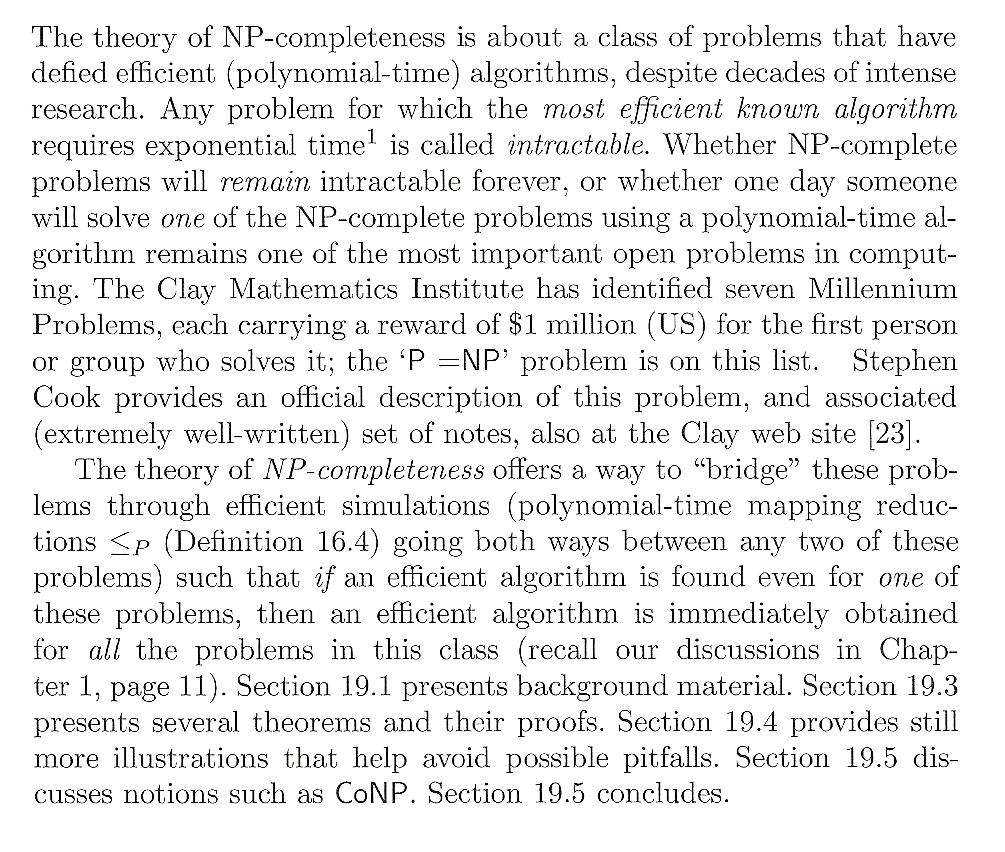


I.e., the definitions is that a NP problem is NP-complete iff all other NP problems reduce to it.

In other words, there are two conditions for a problem to be NP-complete:

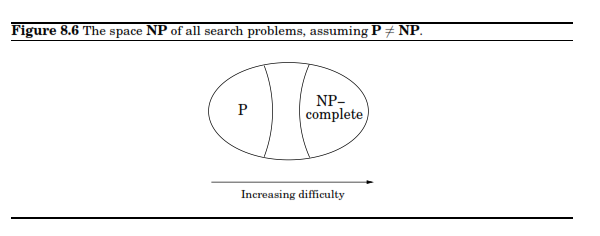
1. Be in NP
2. All other NP problems reduce to it





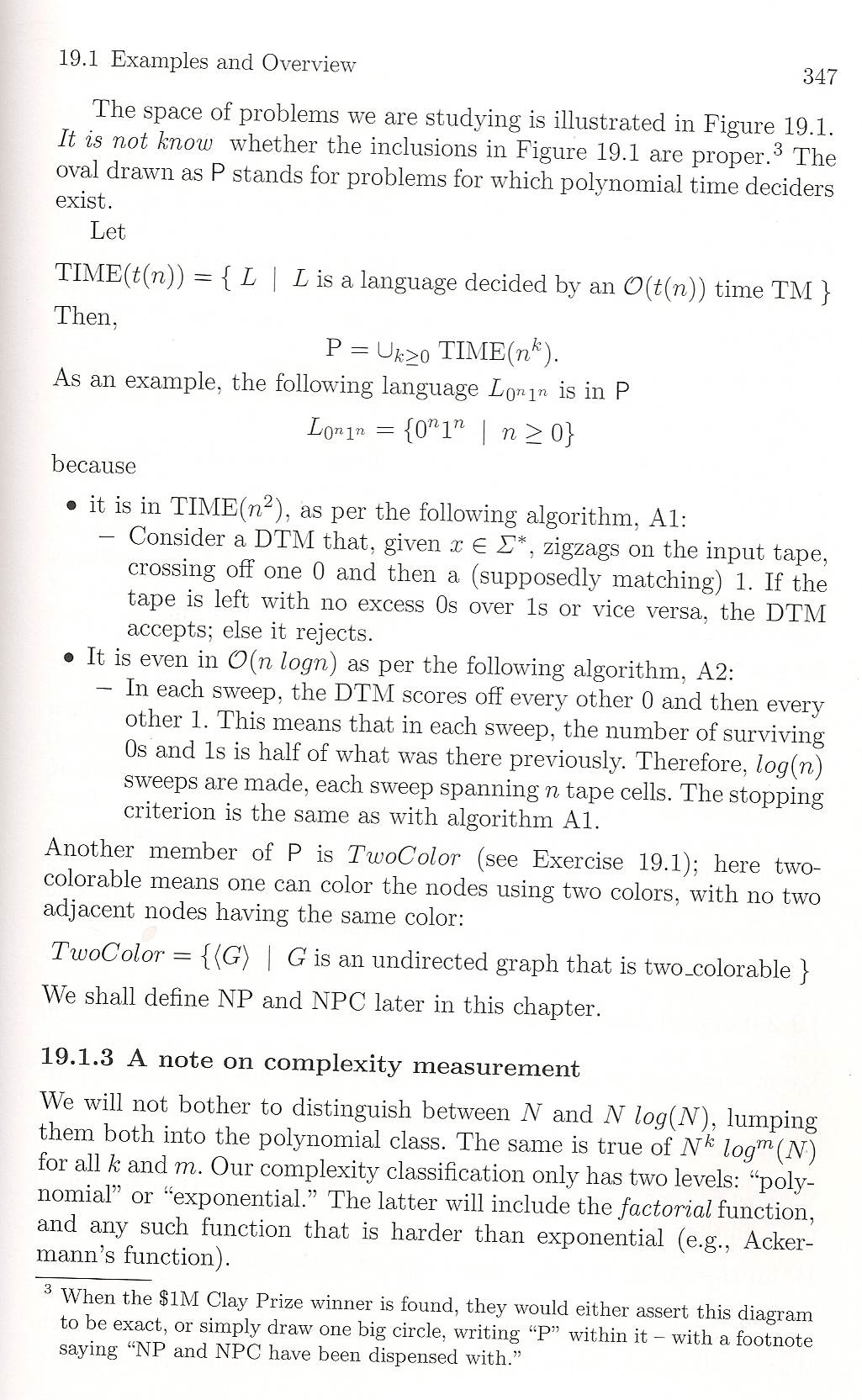
The whole oval is NP.

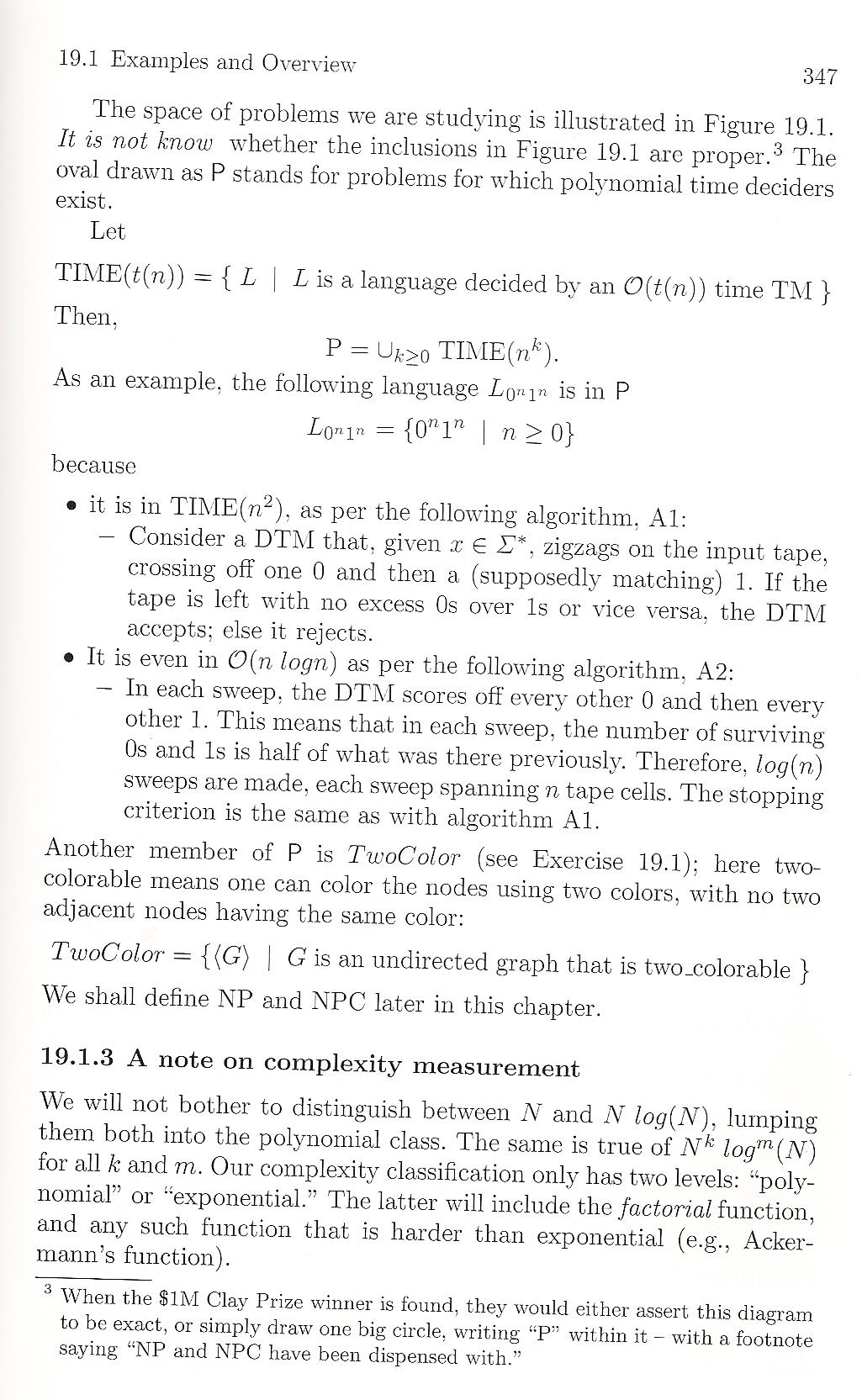
There are also problems outside NP.

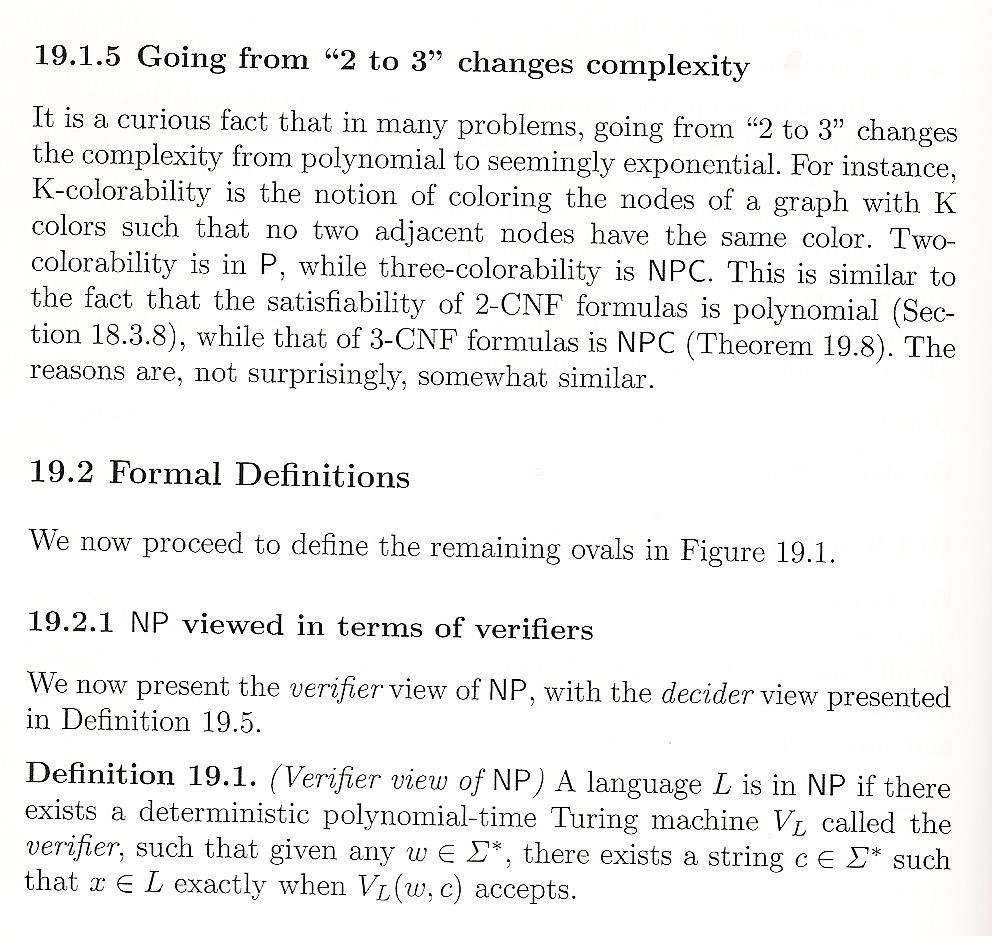


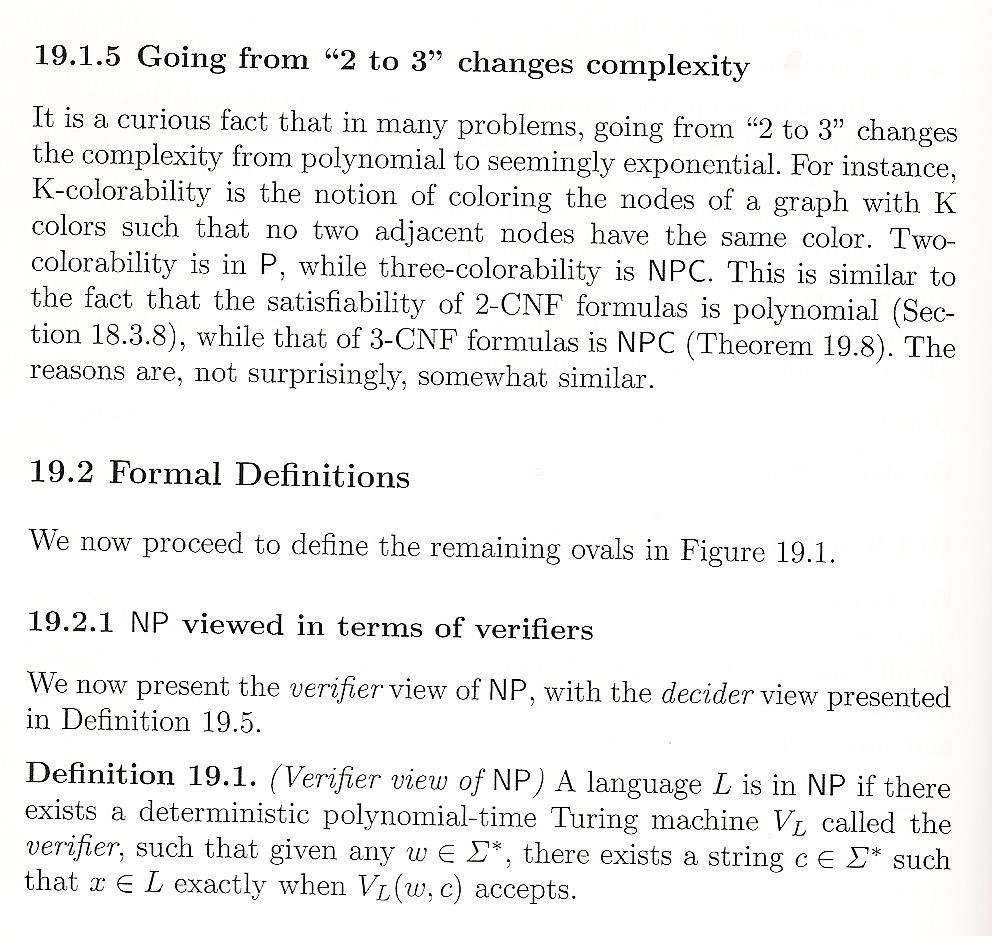
If P = NP than all the three classes on the diagram will coincide.

Below are definitions of the classes using **decision problems** and Turing machines to measure efficiency (efficiency is the number of steps of TM). [Gopalakrishnan 347]



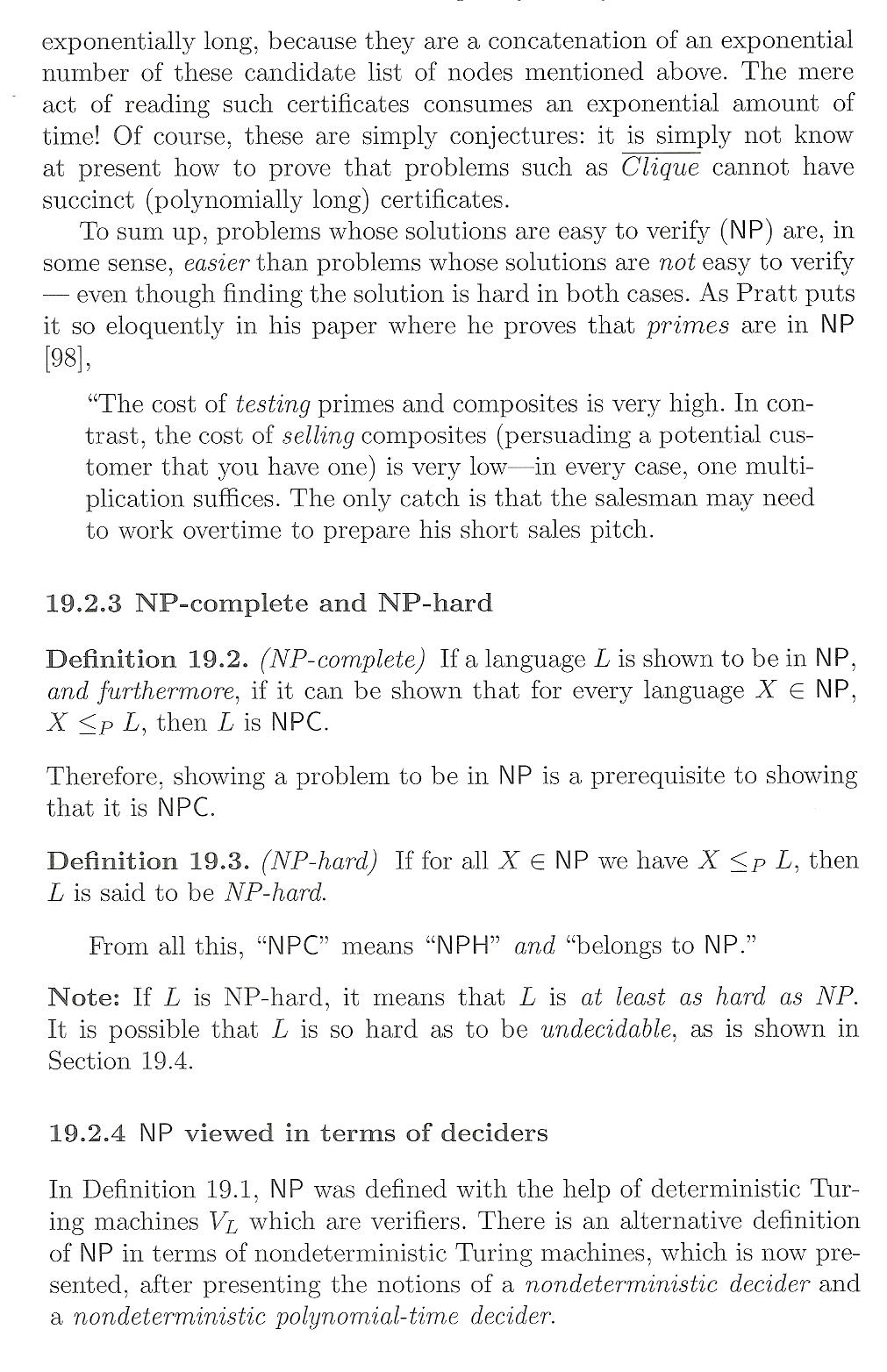
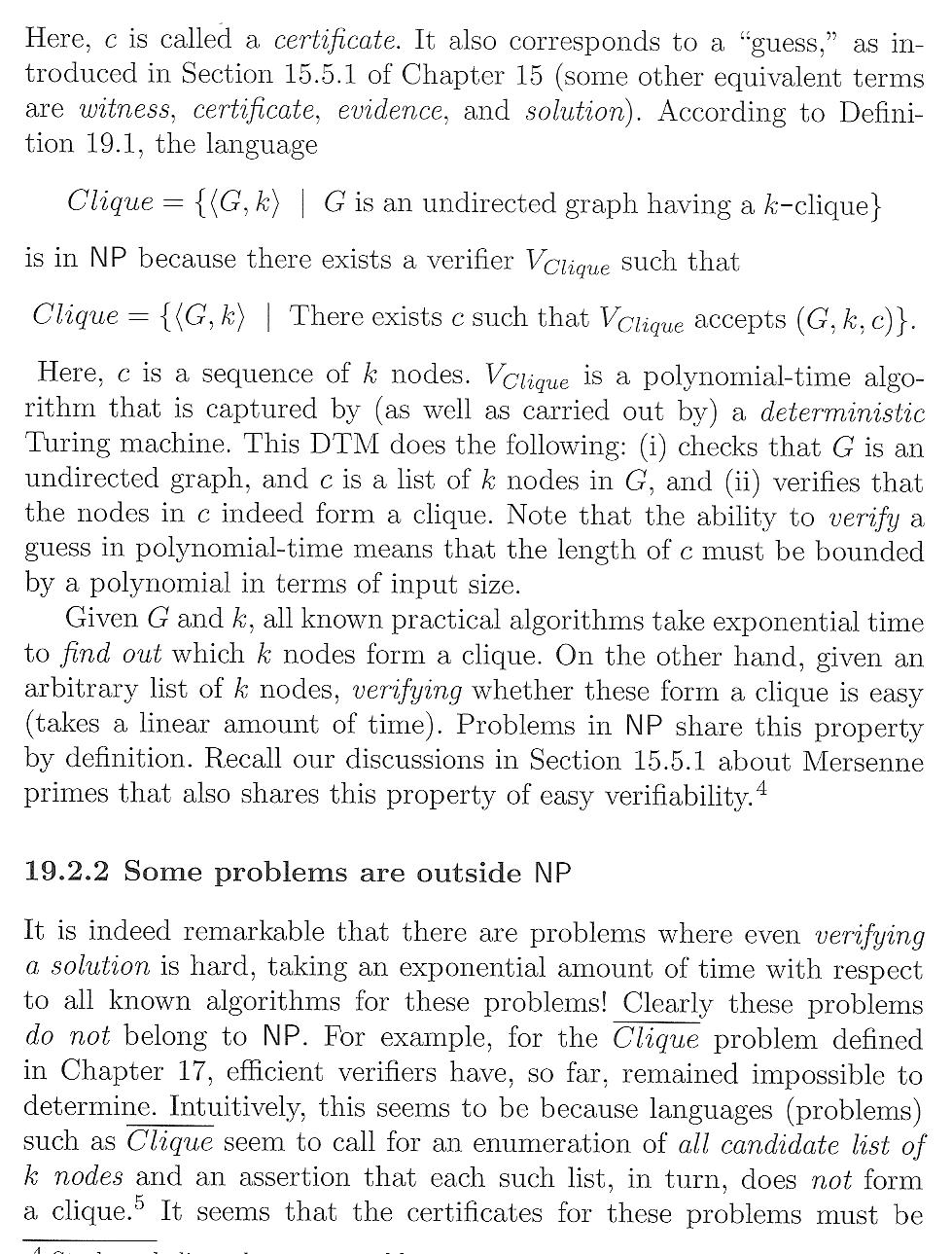


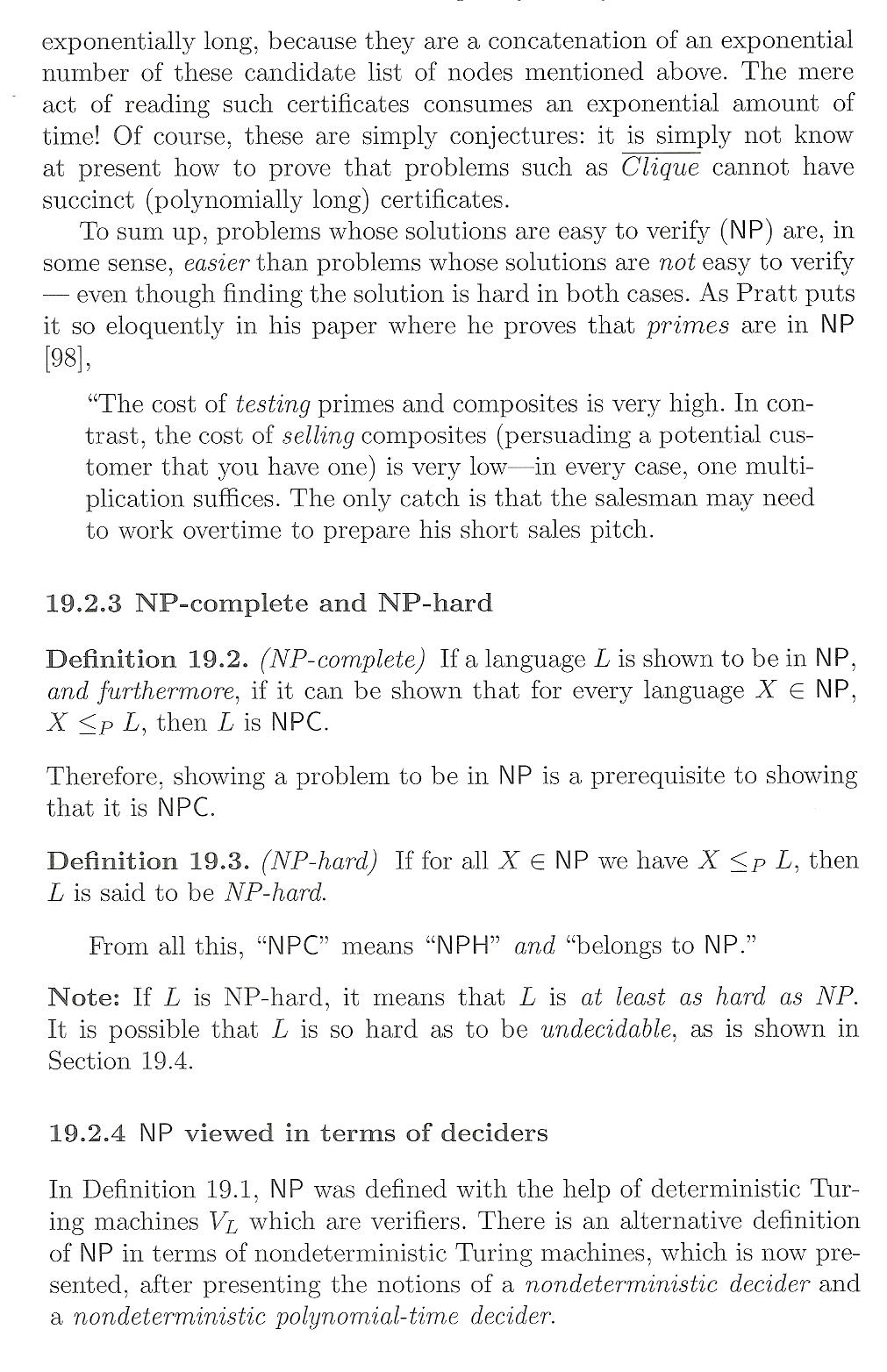


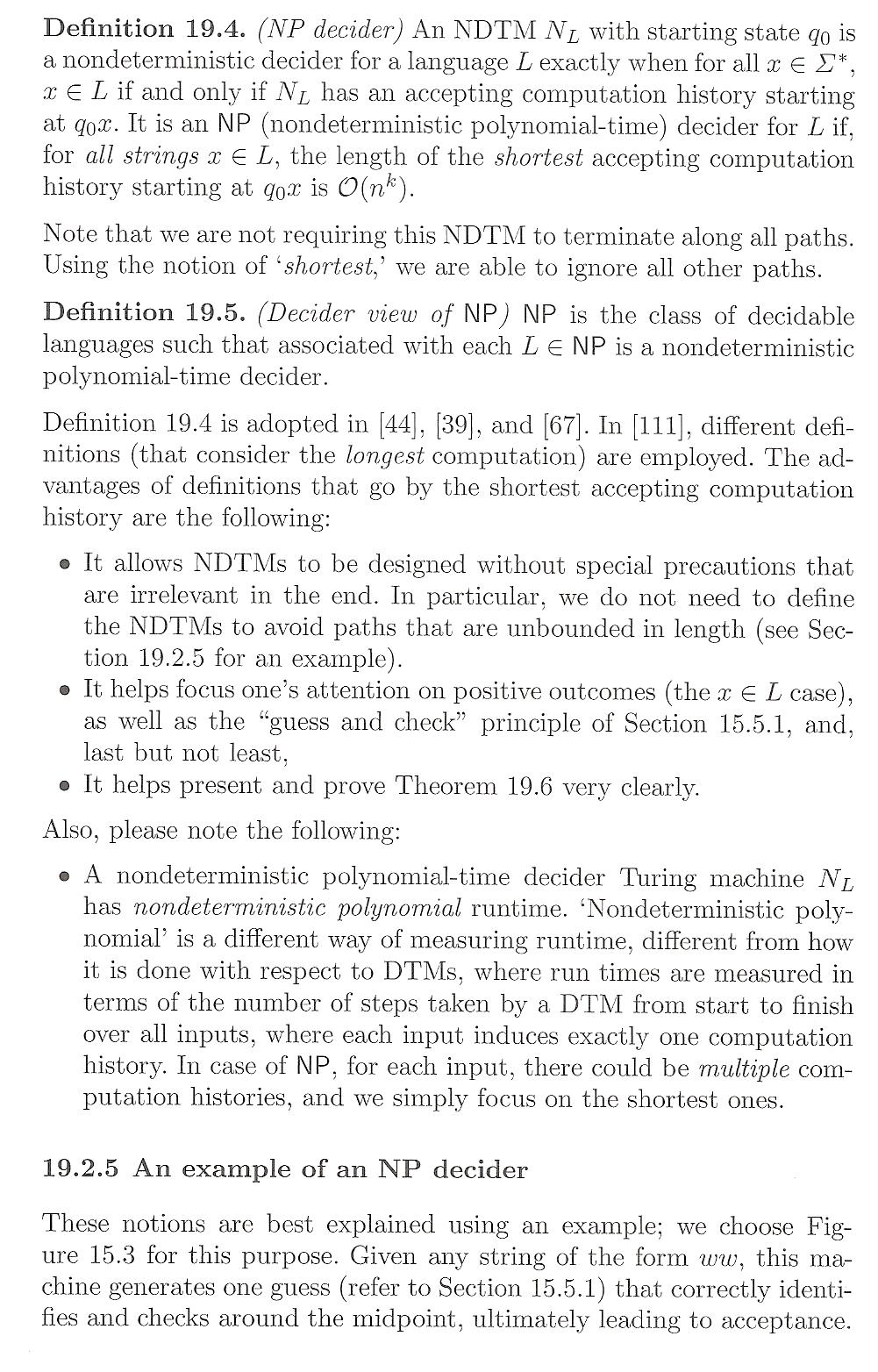


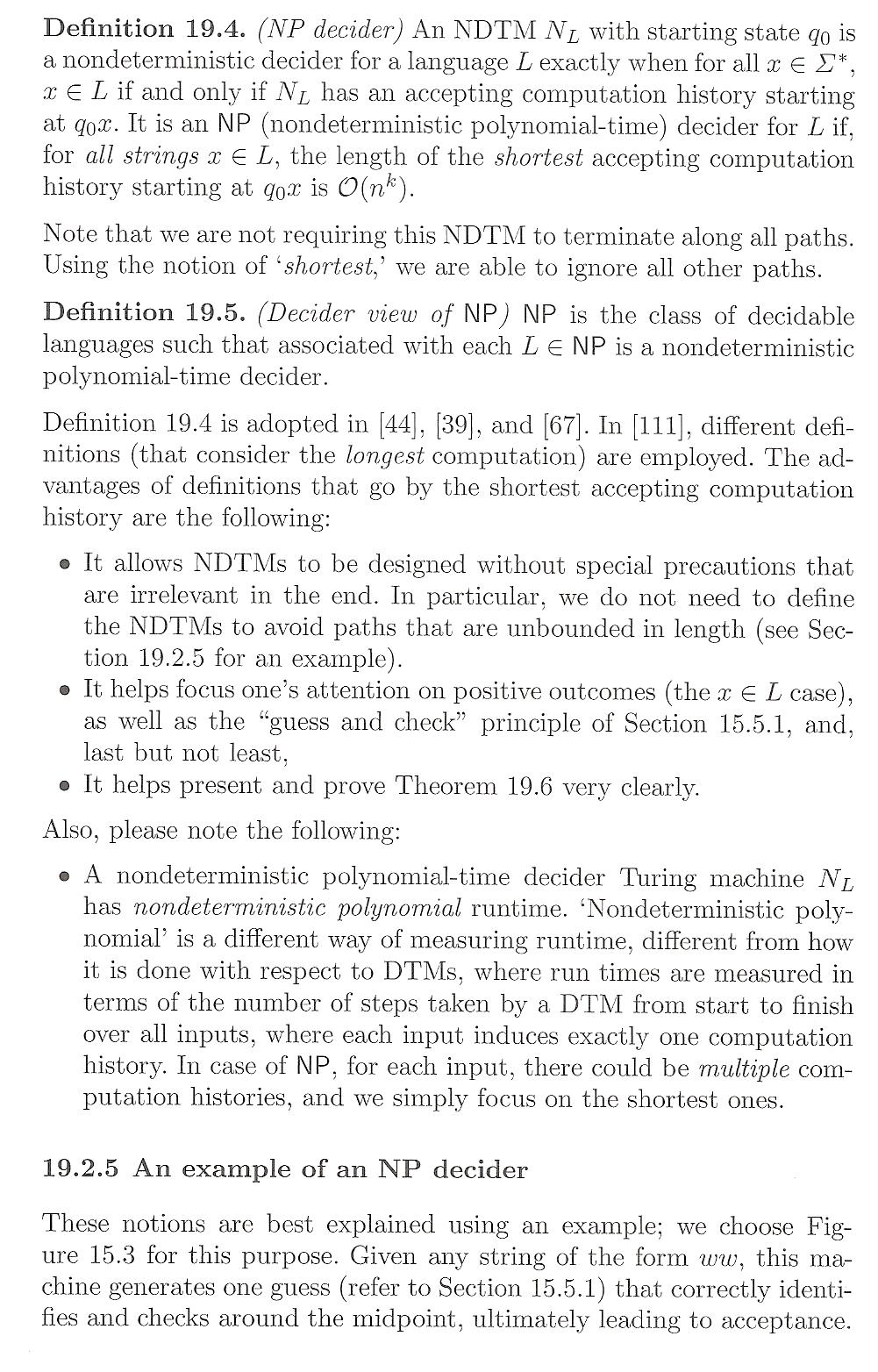
Here, c is called a certificate, evidence.

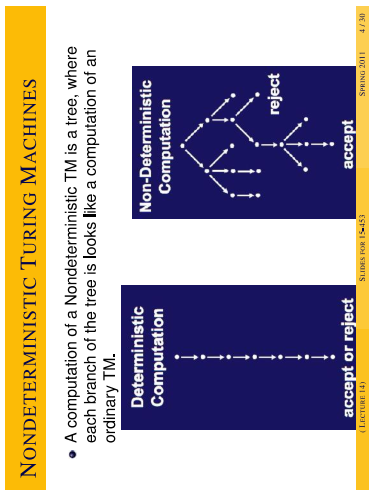
Example of a language in NP follows.

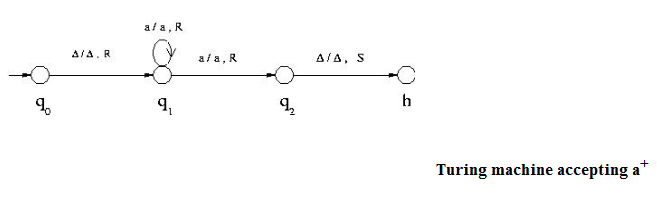








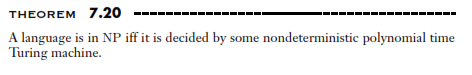




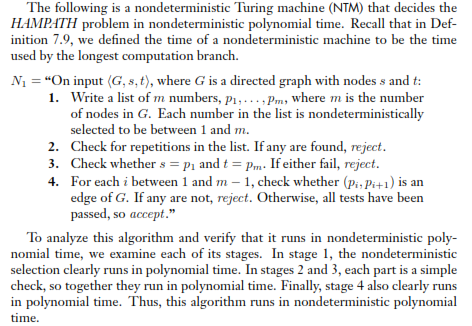
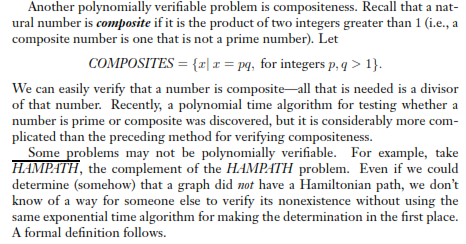
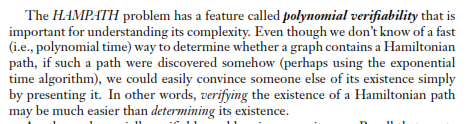
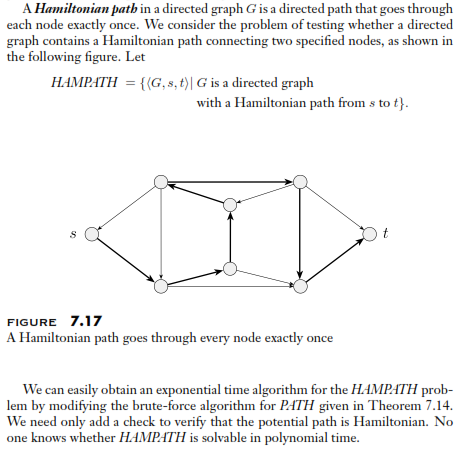
Example Gopalakrishnan 285, see also my paper Ruse 2012 with another solution for the same example; many solutions are possible (and have been done before Ruse). This example shows much better the advantages of nondeterminism. In the example for a+ we could’ve taken a+ = a a\* instead of a\* a; the former TM would be deterministic.

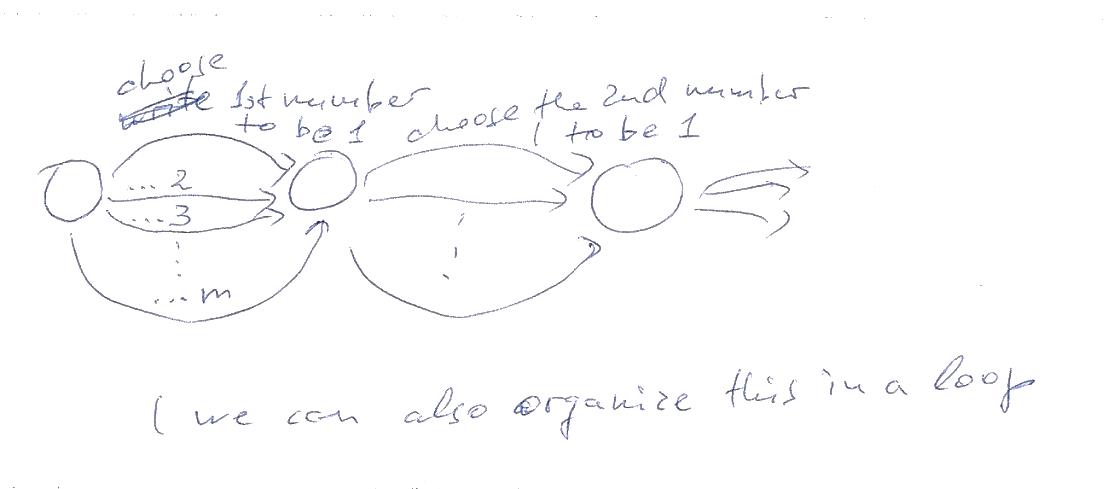
An example of a NP language using this definition: the language {ww} whose nondeterministic TM – decider was discussed.

**Theorem**: The verifier view of NP and the decider view are equivalent. [Gopalakrishnan 352, Sipser 294]

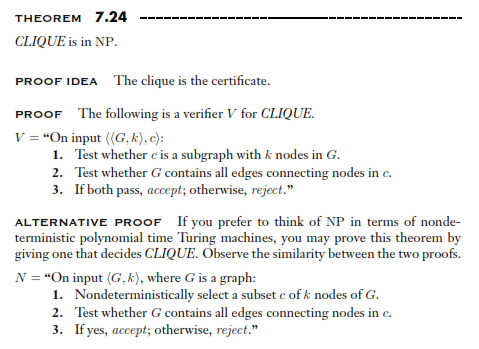
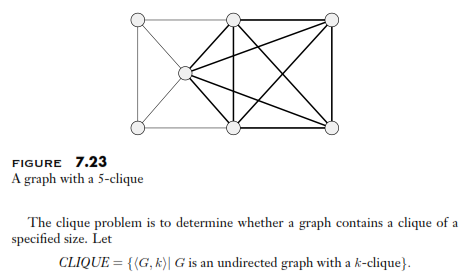


Other problems – Hamiltonian path [Sipser 292-]





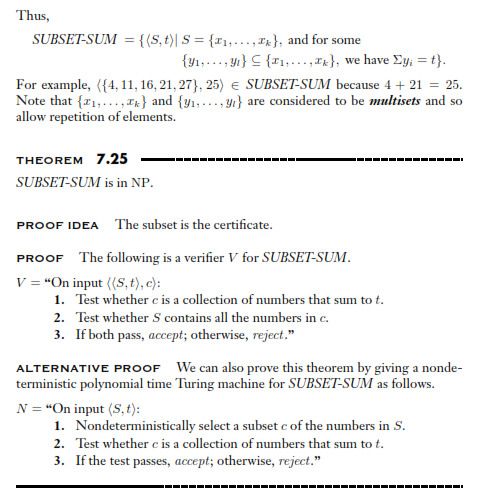
Step 1



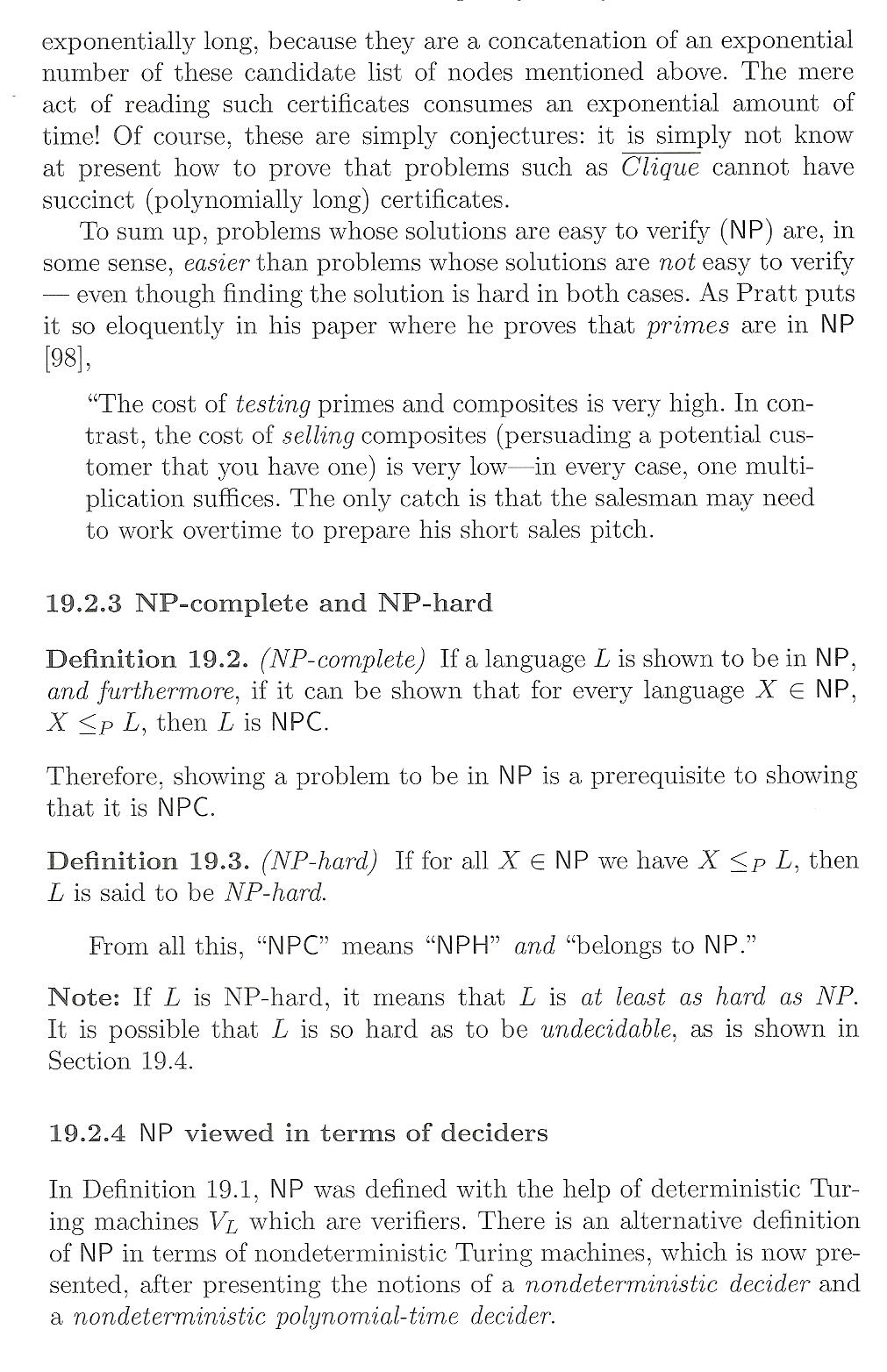
The above example emphasizes the relationship between the two definitions of NP!

Note that if we prove that a problem is in NP that does not mean that it is not in P – we simply don’t know a polynomial-time algorithm for this problem. One might be able to find such an algorithm in the future.

In contrast, when we prove that a problem is NP-complete, we can claim that it is not in P unless P = NP!



Actually, the Hamiltonian path, clique, subset-sum and others are NP-complete problems [Sipser Sec. 7.5]

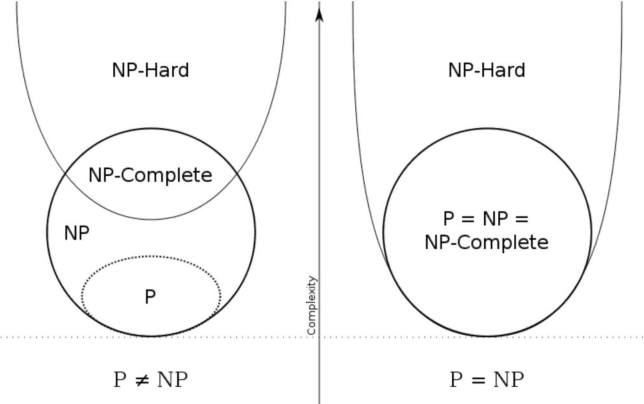


So it is outside NP – may be decidable outside NP, or even undecidable.

(plus)

In other words, NP-complete are NP-hard problems that, in addition, are in NP.

NP-hard problems may be of any type: [decision problems](http://en.wikipedia.org/wiki/Decision_problem), [search problems](http://en.wikipedia.org/wiki/Search_problem), or [optimization problems](http://en.wikipedia.org/wiki/Optimization_problem).



**decidable**

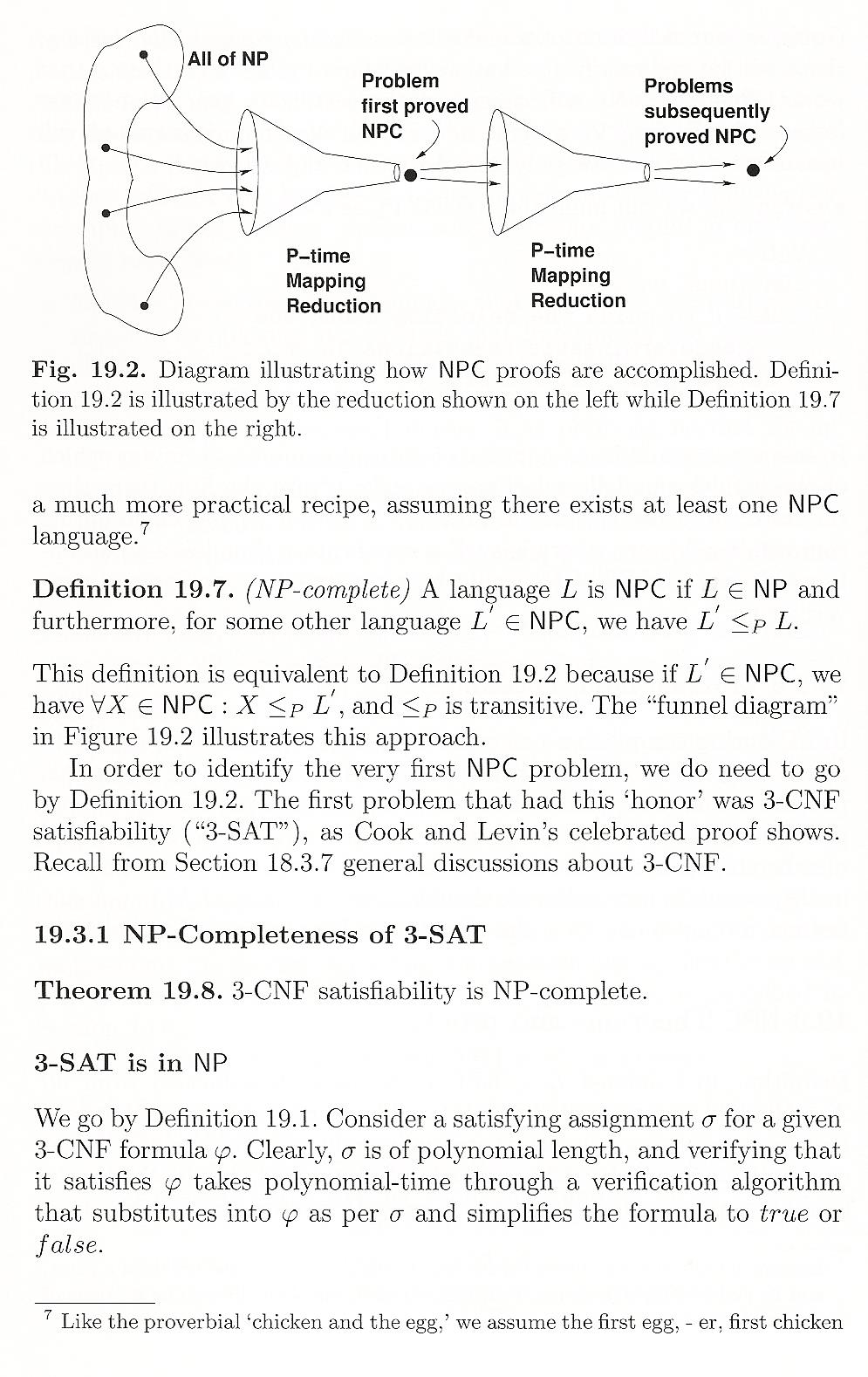
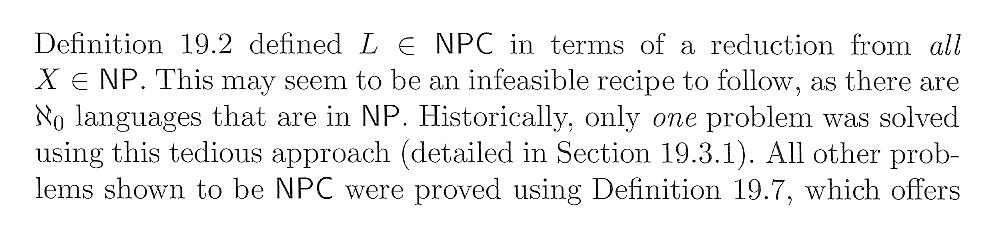
**undecidable**

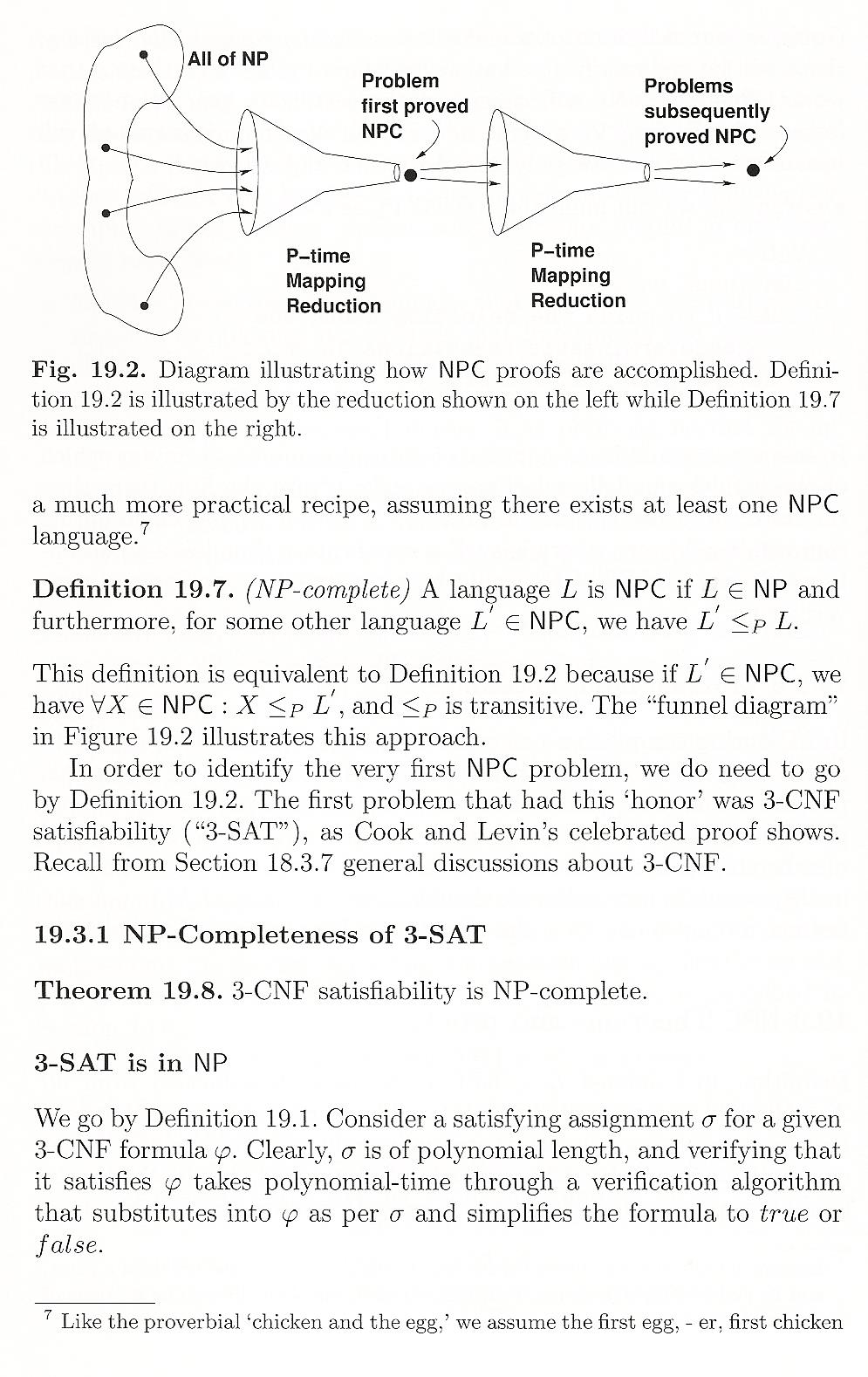
Examples of NP-hard problems that are not NP-complete follow.

There are decision problems that are NP-hard but not NP-complete, for example the [halting problem](http://en.wikipedia.org/wiki/Halting_problem). This is the problem which asks "given a program and its input, will it run forever?" That's a *yes*/*no* question, so this is a decision problem. It is easy to prove that the halting problem is *NP-hard* but not *NP-complete*. For example, the [Boolean satisfiability problem](http://en.wikipedia.org/wiki/Boolean_satisfiability_problem) can be reduced to the halting problem by transforming it to the description of a [Turing machine](http://en.wikipedia.org/wiki/Turing_machine) that tries all [truth value](http://en.wikipedia.org/wiki/Truth_value) assignments and when it finds one that satisfies the formula it halts and otherwise it goes into an infinite loop. It is also easy to see that the halting problem is not in *NP* since all problems in NP are decidable in a finite number of operations, while the halting problem, in general, is [undecidable](http://en.wikipedia.org/wiki/Undecidable_problem).

There are also NP-hard problems that are neither NP-complete nor undecidable. For instance, the language of [True quantified Boolean formulas](http://en.wikipedia.org/wiki/True_quantified_Boolean_formula) is decidable in [polynomial space](http://en.wikipedia.org/wiki/PSPACE), but not non-deterministic polynomial time (unless NP = [PSPACE](http://en.wikipedia.org/wiki/PSPACE)).

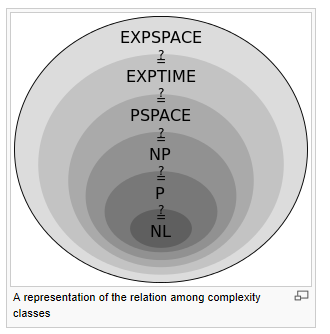
On proving NP-completeness

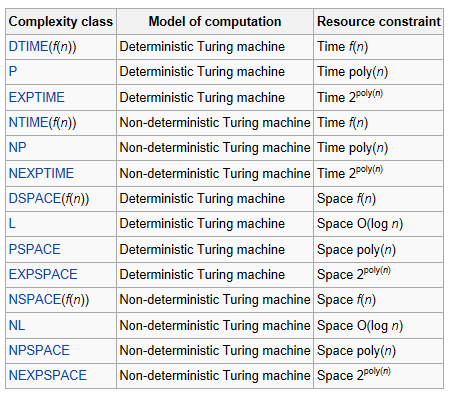




Some other complexity classes

[<http://en.wikipedia.org/wiki/Computational_complexity_theory>]





It turns out that PSPACE = NPSPACE and EXPSPACE = NEXPSPACE by [Savitch's theorem](http://en.wikipedia.org/wiki/Savitch%27s_theorem).

Other important complexity classes include [BPP](http://en.wikipedia.org/wiki/BPP_(complexity)), [ZPP](http://en.wikipedia.org/wiki/ZPP_(complexity)) and [RP](http://en.wikipedia.org/wiki/RP_(complexity)), which are defined using [probabilistic Turing machines](http://en.wikipedia.org/wiki/Probabilistic_Turing_machine); [AC](http://en.wikipedia.org/wiki/AC_(complexity)) and [NC](http://en.wikipedia.org/wiki/NC_(complexity)), which are defined using Boolean circuits and [BQP](http://en.wikipedia.org/wiki/BQP) and [QMA](http://en.wikipedia.org/wiki/QMA), which are defined using quantum Turing machines. [#P](http://en.wikipedia.org/wiki/Sharp-P) is an important complexity class of counting problems (not decision problems). Classes like [IP](http://en.wikipedia.org/wiki/IP_(complexity)) and [AM](http://en.wikipedia.org/wiki/AM_(complexity)) are defined using [Interactive proof systems](http://en.wikipedia.org/wiki/Interactive_proof_system). [ALL](http://en.wikipedia.org/wiki/ALL_(complexity)) is the class of all decision problems.

A lot also in Goldreich (both books) – incl. presentations for decision, search, optimization problems.