7.3

a. There is a `pl true` in the Python code, and a version of `ask` in the Lisp code that serves the same purpose. The Java code did not have this function as of May 2003, but it should be added soon.)

b. The sentences $\mathbf{True}$, $P \vee \neg P$, and $P \wedge \neg P$ can all be determined to be true or false in a partial model that does not specify the truth value for P.

c. It is possible to create two sentences, each with k variables that are not instantiated in the partial model, such that one of them is true for all 2^k possible values of the variables, while the other sentence is false for one of the 2^k values. This shows that in general one must consider all 2^k possibilities. Enumerating them takes exponential time.

d. The python implementation of `pl true` returns true if any disjunct of a disjunction is true, and false if any conjunct of a conjunction is false. It will do this even if other disjuncts/conjuncts contains uninstantiated variables. Thus, in the partial model where P is true, $P \vee Q$ returns true, and $\neg P \wedge Q$ returns false. But the truth values of $Q \vee \neg Q$, $Q \vee \mathbf{True}$, and $Q \wedge \neg Q$ are not detected.

e. Our version of `tt entails` already uses this modified `pl true`. It would be slower if it did not.

7.7.

These can be computed by counting the rows in a truth table that come out true. Remember to count the propositions that are not mentioned; if a sentence mentions only A and B, then we multiply the number of models for $\{A,B\}$ by 2^2 to account for C and D.

a. 6
b. 12
c. 4

7.8.

A binary logical connective is defined by a truth table with 4 rows. Each of the four rows may be true or false, so there are $2^4 = 16$ possible truth tables, and thus 16 possible connectives. Six of these are trivial ones that ignore one or both inputs; they correspond to $\mathbf{True}$, $\mathbf{False}$, P, Q, $\neg P$ and $\neg Q$. Four of them we have already studied: $\wedge, \vee, \Rightarrow, \Leftrightarrow$.

The remaining six are potentially useful. One of them is reverse implication ($\Leftarrow$ instead of $\Rightarrow$), and the other five are the negations of $\wedge, \vee, \Rightarrow, \Leftrightarrow$ and $\Leftarrow$. (The first two of these are sometimes called *nand* and *nor*.)

7.10

We use the `validity` function from `logic/prop.lisp` to determine the validity of each sentence:

```
> (validity "Smoke => Smoke")
VALID
```

```
> (validity "Smoke => Fire")
SATISFIABLE
> (validity "(Smoke => Fire) => (~Smoke => ~Fire)")
SATISFIABLE
> (validity "Smoke | Fire | ~Fire")
VALID
> (validity "((Smoke ^ Heat) => Fire) <=> ((Smoke => Fire) | (Heat => Fire))")
VALID
> (validity "(Smoke => Fire) => ((Smoke ^ Heat) => Fire)")
VALID
> (validity "Big | Dumb | (Big => Dumb)")
VALID
> (validity "(Big ^ Dumb) | ~Dumb")
SATISFIABLE
```

Many people are fooled by (e) and (g) because they think of implication as being causation, or something close to it. Thus, in (e), they feel that it is the combination of Smoke and Heat that leads to Fire, and thus there is no reason why one or the other alone should lead to fire. Similarly, in (g), they feel that there is no necessary causal relation between Big and Dumb, so the sentence should be satisfiable, but not valid. However, this reasoning is incorrect, because implication is *not* causation—implication is just a kind of disjunction (in the sense that $P \Rightarrow Q$ is the same as $\neg P \vee Q$). So $\mathbf{Big} \vee \mathbf{Dumb} \vee (\mathbf{Big} \Rightarrow \mathbf{Dumb})$ is equivalent to $\mathbf{Big} \vee \mathbf{Dumb} \vee \neg\mathbf{Big} \vee \mathbf{Dumb}$, which is equivalent to $\mathbf{Big} \vee \neg\mathbf{Big} \vee \mathbf{Dumb}$, which is true whether $\mathbf{Big}$ is true or false, and is therefore valid.