e. **Axioms for statement logic** (that is, the meaning of $\vdash$ allows us to use these):

Twelve of the axioms are axioms for any Boolean algebra:

1–6	2 commutative, 2 associative, 2 distributive [see top p. 210]			
7–8	2 absorption	$\alpha \wedge \text{false} \vdash \text{false}$	$\alpha \vee \text{true} \vdash \text{true}$
9–10	2 identity	$\alpha \wedge \text{false} \vdash \alpha$	$\alpha \vee \text{true} \vdash \alpha$
11–12	2 inverse	$\alpha \wedge \neg \alpha \vdash \text{false}$	$\alpha \vee \neg \alpha \vdash \text{true}$

The two DeMorgan's laws on p. 210 can be proved as theorems from the above.

13	Modus ponens [MP]	$\alpha \Rightarrow \beta \ , \ \alpha \vdash \beta$	
14	Modus tollendo tollens [MTT]	$\alpha \Rightarrow \beta \ , \ \beta \Rightarrow \gamma \vdash \alpha \Rightarrow \gamma$	
15	Resolution	$\alpha \vee \beta \ , \ \neg\beta \vee \gamma \vdash \alpha \vee \gamma$	(*)

Resolution says: If in several conjuncts, something appears both negated and not negated, it can be eliminated.

MP and MTT can be proven as theorem from Resolution. MTT is a special case of resolution because $\alpha \Rightarrow \beta,\ \beta \Rightarrow \gamma \vdash \alpha \Rightarrow \gamma$ can be rewritten without $\Rightarrow$ as $\neg\alpha \vee \beta, \neg\beta \vee \gamma \vdash \neg\alpha \vee \gamma$, which is clearly (*) with $\neg\alpha$ playing the role of α. Then MP is a special case of MTT because from (true $\Rightarrow \beta$) we can deduce β.

16	And elimination	$\alpha \wedge \beta \vdash \alpha$
17	Or introduction	$\alpha \vdash \alpha \vee \beta$

Or introduction is important because none of 1–16 allow a new letter to be introduced. If all we have are rules like 1–16, we cannot find all true statements (the logic is not complete). But Resolution as the only search tool does offer at least **refutation completeness.** As p. 214 says, this means that for any given sentence, resolution can either prove or disprove it. Resolution cannot enumerate all true sentences, whereas a complete logic engine could, because **statement calculus is complete.** [1]

f. Axioms for the Wumpus world pictured on p. 198.

Let Bij mean a breeze at (i,j) Pij mean a pit at (i,j)
Wij mean a wumpus at (i, j) Sij mean a stench at (i,j)

Here are some axioms for the wumpus world of p. 198 before the agent moves one square to the right (p. 208). Set aside the Wumpus and the stench for a moment.

	R1	$\neg$ P11	There's no pit where the agent is (via sensors)
	R2	B11 $\Leftrightarrow$ P12 $\vee$ P21	Breezy square are near pits (6 such rules for Fig. 7.2 via geometry)
`	R3	B21 $\Leftrightarrow$ P11 $\vee$P22 $\vee$ P31	" " " " " " " "
	R4	$\neg$ B11	There's no breeze where the agent is (via sensors)

Now the agent moves one square to the right, adding one other axiom to the Knowledge Base [KB]

R5 B21 There's a breeze where the agent is (via sensors)

It is consistent to add axiom R5 assuming that the agent's sensors are always working perfectly.

KB = R1 $\wedge$ R2 $\wedge$ R3 $\wedge$ R4 $\wedge$ R5.

We don't model the agent's location as a part of the statement calculus. The agent's state includes information about the agent's position, but R1–R5 are just state information about the world. We skip using logic to track the location and facing direction of the agent because it cannot easily be done in statement logic. [As p. 227—which is not a part of your reading—says.]

g. Resolution.
How would we prove that we are not in a pit? Prove ¬P12?

Informally: B11 is false by R4 therefore by R2 and DeMorgan's laws, both P12 and P21 are false

Formally: We proceed by resolution as follows. We must take the whole KB, because if we're a computer, we're not smart enough to have picked R2 and R4 out of the KB.

In System Internals and Assembly Language, we put statements in disjunctive normal form in order to find logical statements equivalent to truth tables. Now we put statements in **conjunctive normal form** (CNF) to apply resolution as a proof technique. We can put statements with n statement symbols into CNF in time O(n).

So our agent in the Wumpus world has the following KB when written in CNF. See Figure 7.13.

KB=(¬B11∨P12∨B11) ∧ (P12∨P21∨¬P21) ∧ (¬B11∨P21∨B11) ∧ (P12∨P21∨¬P12) ∧ ¬P21 ∧ ¬P12
t t t t

Now resolve ¬P12 against KB: Assume the opposite, P12. Take KB ∧ P12, which reduces to ¬P21 ∧ ¬P12 ∧ P12, which in turn reduces to false (what your book calls the "empty clause"). Hence, our assumption of P12 was wrong, so ¬P12 has been proved by contradiction.

The program `aima.logic.propositional.algorithms.PLResolution.java` does resolution.

h. Here's an example of Resolution to solve a query in Prolog.

Let a = People are wise.	Let KB be	a.	[People are wise.]
b = People are rich.		b :- a.	[If people are wise, people are rich.]
d = People are happy.		d :- b.	[If people are rich, people are happy.]
	Let a query be	?-d.	[Are people happy?]

Here's what Prolog does behind the scenes: Write the KB in CNF, and conjoin the negation of the "goal" or "query": $a \wedge (\neg a \vee b) \wedge (\neg b \vee d) \wedge \neg d$. Then by resolution we get false, so the negation of the goal is false, so the goal is true, or "Yes", as Prolog would say.

i. Homework hints for Chapter 7 teamwork assignment Exercises: §§ 7.8, 7.5, 7.9, 7.3b, 7.3d.

Exercise 7.3b asks whether you can think of 3 sentences which are known to be true or false without knowing whether some of the literals of the sentences are true or false, presumably because the truth or falsity of the separate literals does not matter. No program need be written. This is easy.

For Exercise 7.3d, write a Java driver program to call on a method `pltrue` which in turn calls on methods that allow you to populate a knowledge base and then test the truth of a sentence. See `TTEntailsDemo.java` for how to set up the knowledge base with the `tell` method and `askWithTTEntails` for how to test for truth. No modification of existing programs is needed. Big hint: In order to show that `TTEntails` properly "sometimes judge[s] truth from partial models," to help you think of three examples of sentence, you will have to notice that `ttCheckAll` is called recursively twice, connected with Java's `&&` operator, which does not evaluate the second argument if the first is known to be false.

---

[1] Are there theorem provers that work in general for predicate calculus? Yes!
[The footnote is at e above, but read this after you read h above, where Prolog is introduced.]

Prolog only handles so-called **Horn clauses,** which is to say clauses of the form "α *if* a-conjunction-of-positive-literals>." Here is an example:

a :- b, c, d, e.

A Horn clause is equivalently a disjunction of literals where at most one is positive. Here's why:
Remember that r ⇒ means ¬ r ∨s. Thus a :- b, c, d, e. translates to ¬ (b ∧ c ∧ d ∧ e) ∨ a.
Now repeatedly apply DeMorgan laws to get ¬ b ∨ ¬ c ∨ ¬ d ∨ ¬ e ∨ a.

Where to find general theorem provers? **http://en.wikipedia.org/wiki/Automated_theorem_proving**
Others below are listed in Appendix C of a book I recently ordered for our library and read: Patrick Blackburn and Johan Bos's *Representation and Inference for Natural Language* (Stanford, CA: Center for the Study of Language and Information, 2005). A good project for AI would be to demonstrate one of these theorem provers to our class.

Theorem Provers. Several of the following are not currently under development, but there are links to more recent stuff from here.

Bliksem	http://www.mpi-sb.mpg.de/~nivelle/software/bliksem
FDPLL	http://www.uni-koblenz.de/~peter/FDPLL
Gandalf	http://www.ttu.ee/it/gandalf
Otter	**http://www-unix.mcs.anl.gov/AR/otter**
Scott	**http://users.rsise.anu.edu.au/~jks/scott.html**
Spass	http://spass.mpi-sb.mpg.de/index.html
Vampire	**http://www.cs.man.ac.uk/%7Eriazanoa/Vampire**
zChaff	http://www.princeton.edu/~chaff/zchaff.html [cf. Gandalf]
Mace	http://www.cs.unm.edu/~mccune
Paradox	**http://www.math.chalmers.se/~koen/paradox**
Satchmo	http://www.pms.informatik.uni-muenchen.de/software