AIES Notes: Knowledge Representation & Reasoning [Lecture 8]

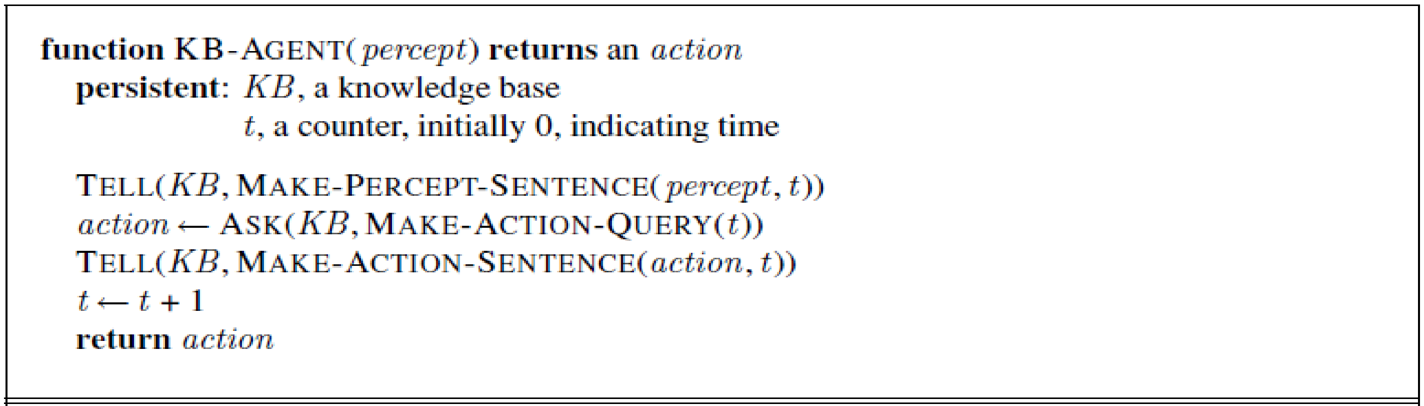
In which we design agents that can form representations of a complex world, use a process of inference to derive new representations about the world, and use these new representations to deduce what to do.

Reasoning, Representations, Knowledge-based Agents

Humans, it seems, know things; and what they know helps them do things. These are not empty statements. They make strong claims about how the intelligence of humans is achieved—not by purely reflex mechanisms but by processes of **reasoning** that operate on internal **representations** of knowledge. In AI, this approach to intelligence is embodied in **knowledge-based agents**.

Knowledge-Based Agents

The central component of a knowledge-based agent is its **knowledge base**, or **KB**. A knowledge base is a set of **sentences**. (Here "sentence" is used as a technical term. It is related but not identical to the sentences of English and other natural languages.) Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world. Sometimes we dignify a sentence with the name **axiom**, when the sentence is taken as given without being derived from other sentences.



```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 7.1 A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

Tell-Ask Procedure

Each time the agent program is called, it does three things.

- First, it **TELLs** the knowledge base what it perceives.
- Second, it **ASKs** the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on.
- Third, the agent program **TELLs** the knowledge base which action was chosen, and the agent executes the action.

Percept-Sentence, Action-Query, Action-Sentence

The details of the representation language are hidden inside three functions that implement the interface between the sensors and actuators on one side and the core representation and reasoning system on the other. MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time. MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time. Finally, MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed. The details of the inference mechanisms are hidden inside TELL and ASK. Later sections will reveal these details.

Declarative v/s Procedural

A knowledge-based agent can be built simply by TELLing it what it needs to know. Starting with an empty knowledge base, the agent designer can TELL sentences one by one until the agent knows how to operate in its environment. This is called the **declarative** approach to system building. In contrast, the **procedural** approach encodes desired behaviors directly as program code.

LOGIC

Knowledge bases consist of sentences. These sentences are expressed according to the **syntax** of the representation language, which specifies all the sentences that are well formed. The notion of syntax is clear enough in ordinary arithmetic: "x + y = 4" is a well-formed sentence, whereas "x4y+ =" is not.

A logic must also define the **semantics or meaning of sentences.** The semantics defines the truth of each sentence with respect to each **possible world**. For example, the semantics for arithmetic specifies that the sentence "x + y =4" is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1. In standard logics, every sentence must be either true or false in each possible world—there is no "in between."

Model

When we need to be precise, we use the term **model** in place of "**possible world**." Whereas possible worlds might be thought of as (potentially) real environments that the agent might or might not be in, **models are mathematical abstractions**, each of which simply fixes the truth or falsehood of every relevant sentence. Informally, we may think of a possible world as, for example, having x men and y women sitting at a table playing bridge, and the sentence x + y =4 is true when there are four people in total. Formally, the **possible models are just all possible assignments of real numbers to the variables x and y**. Each such assignment fixes the truth of any sentence of arithmetic whose variables are x and y. If a sentence α is true in **model m**, we say that m satisfies α or sometimes m is a model of α. We use the notation **M(α)** to mean the set of **all models of α**.

Now that we have a notion of truth, we are ready to talk about logical reasoning. This involves the relation of logical entailment between sentences—the idea that a sentence follows logically from another sentence. In mathematical notation, we write α |= β. to mean that the sentence α entails the sentence β. The formal definition of entailment is this: α |= β if and only if, in every model in which α is true, β is also true. Using the notation just introduced, we can write α |= β if and only if M(α) ⊆ M(β).

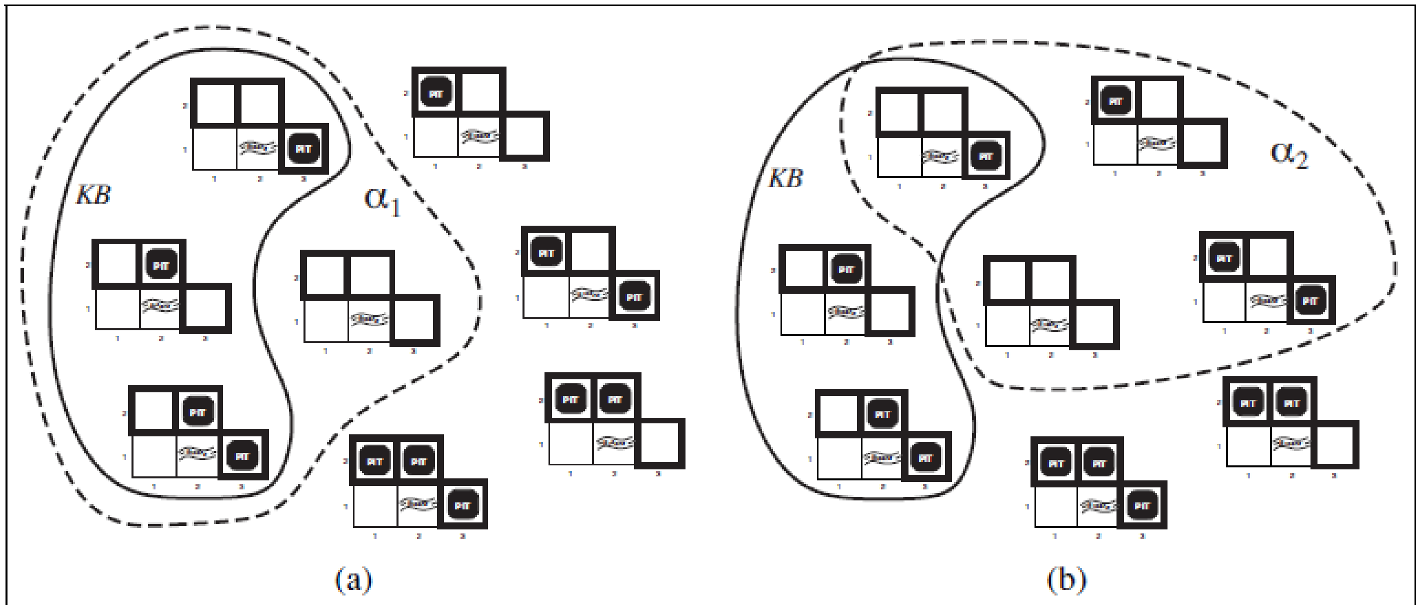


Figure 7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).

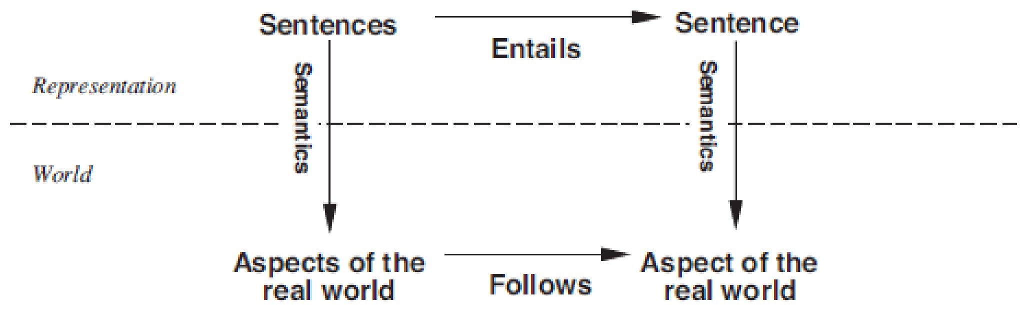


Figure 7.6 Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

PROPOSITIONAL LOGIC: A VERY SIMPLE LOGIC

The **syntax** of propositional logic defines the allowable sentences. The **atomic sentences** consist of a single proposition symbol. Each such symbol stands for a **proposition that can be true or false**.

Complex sentences are constructed from simpler sentences, using parentheses and logical connectives. There are five connectives in common use:

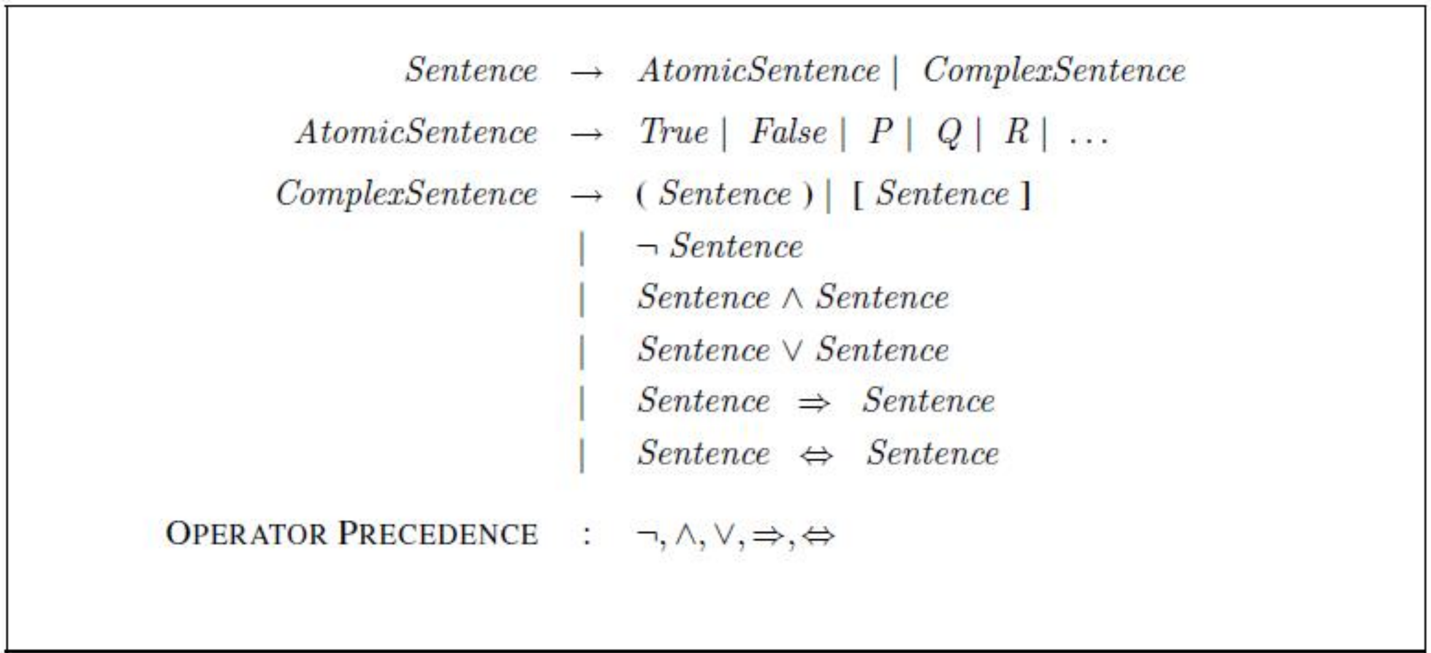
¬ (not). A sentence such as ¬W1,3 is called the negation of W1,3. A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal).

∧ (and). A sentence whose main connective is ∧, such as W1,3 ∧ P3,1, is called a conjunction; its parts are the conjuncts. (The ∧ looks like an "A" for "And.")

∨ (or). A sentence using ∨, such as (W1,3∧P3,1)∨W2,2, is a disjunction of the disjuncts (W1,3 ∧ P3,1) and W2,2. (Historically, the ∨ comes from the Latin "vel," which means "or." For most people, it is easier to remember ∨ as an upside-down ∧.)

⇒ (implies). A sentence such as (W1,3∧P3,1) ⇒ ¬W2,2 is called an implication (or conditional). Its premise or antecedent is (W1,3 ∧P3,1), and its conclusion or consequent is ¬W2,2. Implications are also known as rules or if–then statements. The implication symbol is sometimes written in other books as ⊃ or →.

⇔ (if and only if). The sentence W1,3 ⇔ ¬W2,2 is a biconditional. Some other books write this as ≡.



$$\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} & \rightarrow & (\ \textit{Sentence}\ ) \mid [\ \textit{Sentence}\ ] \\
 & \mid & \neg\ \textit{Sentence} \\
 & \mid & \textit{Sentence} \land \textit{Sentence} \\
 & \mid & \textit{Sentence} \lor \textit{Sentence} \\
 & \mid & \textit{Sentence} \ \Rightarrow\ \textit{Sentence} \\
 & \mid & \textit{Sentence} \ \Leftrightarrow\ \textit{Sentence} \\
 & & \\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}$$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Having specified the syntax of propositional logic, we now specify its **semantics**. The **semantics** defines the rules for determining the truth of a sentence with respect to a particular model. In propositional logic, a model simply fixes the **truth value**—true or false—for every proposition symbol.

A Simple Inference Procedure

$$
\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{commutativity of } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{associativity of } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{De Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{distributivity of } \vee \text{ over } \wedge
\end{array}
$$

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

PROPOSITIONAL THEOREM PROVING

Before we plunge into the details of theorem-proving algorithms, we will need some additional concepts related to entailment.

The first concept is logical equivalence: two sentences α and β are **logically equivalent** if they are true in the same set of models. We write this as α ≡ β.

The second concept we will need is **validity**. A sentence is valid if it is true in all models. For example, the sentence P ∨ ¬P is valid. Valid sentences are also known as **tautologies**—they are necessarily true. Because the sentence True is true in all models, every valid sentence is logically equivalent to True. What good are valid sentences? From our definition of entailment, we can derive the deduction theorem, which was known to the ancient Greeks:

For any sentences α and β, α |= β if and only if the sentence (α ⇒ β) is valid.

The final concept we will need is **satisfiability**. A sentence is satisfiable if it is true in, or satisfied by, some model. For example, the knowledge base given earlier, (R1 ∧ R2 ∧ R3 ∧ R4 ∧ R5), is satisfiable because there are three models in which it is true. Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence. The problem of determining the satisfiability of sentences SAT in propositional logic—**the SAT problem**—was the first problem proved to be NP-complete. Many problems in computer science are really satisfiability problems.

Inference and proofs

Inference rules can be applied to derive a **proof—a chain of conclusions that leads to the desired goal**.

SUMMARY

• Intelligent agents need knowledge about the world in order to reach good decisions.

• Knowledge is contained in agents in the form of sentences in a knowledge representation language that are stored in a knowledge base.

• A knowledge-based agent is composed of a knowledge base and an inference mechanism. It operates by storing sentences about the world in its knowledge base, using the inference mechanism to infer new sentences, and using these sentences to decide what action to take.

• A representation language is defined by its syntax, which specifies the structure of sentences, and its semantics, which defines the truth of each sentence in each possible world or model.

• The relationship of entailment between sentences is crucial to our understanding of reasoning. A sentence α entails another sentence β if β is true in all worlds where α is true. Equivalent definitions include the validity of the sentence $\alpha \Rightarrow \beta$ and the un-satisfiability of the sentence $\alpha \land \neg\beta$.

• Inference is the process of deriving new sentences from old ones. Sound inference algorithms derive only sentences that are entailed; complete algorithms derive all sentences that are entailed.

• Propositional logic is a simple language consisting of proposition symbols and logical connectives. It can handle propositions that are known true, known false, or completely unknown.

• The set of possible models, given a fixed propositional vocabulary, is finite, so entailment can be checked by enumerating models. Efficient model-checking inference algorithms for propositional logic include backtracking and local search methods and can often solve large problems quickly.

• Inference rules are patterns of sound inference that can be used to find proofs. The resolution rule yields a complete inference algorithm for knowledge bases that are expressed in conjunctive normal form. Forward chaining and backward chaining are very natural reasoning algorithms for knowledge bases in Horn form.

• Logical state estimation involves maintaining a logical sentence that describes the set of possible states consistent with the observation history. Each update step requires inference using the transition model of the environment, which is built from successor state axioms that specify how each fluent changes.

• Decisions within a logical agent can be made by SAT solving: finding possible models specifying future action sequences that reach the goal. This approach works only for fully observable or sensorless environments.

• Propositional logic does not scale to environments of unbounded size because it lacks the expressive power to deal concisely with time, space, and universal patterns of relationships among objects.

First-Order Logic Practice Questions & Answers

These are practice questions to get you used to writing in first order logic.

Convert to English

1. ∀x IsABunny(x) ⇒ IsCute(x)

All bunnies are cute

2. ∀x IsAStudent(x) ∧ IsTakingAI(x) ⇒ IsCool(x)

Everyone student who is taking AI is cool

3. ∀x IsABunny(x) ∧ IsAStudent(x) ∧ IsTakingAI(x) ⇒ IsCute(x) ∧ IsCool(x)

Every bunny who is a student taking AI is cute and cool

4. ∀x EatsRamen(x) ⇒ IsHomeless(x) ∨ IsAGradStudent(x)

Everyone who eats ramen is either homeless or a graduate student

5. ∃s ∀h IsAStudent(s) ∧ IsTaking(s,AI) ∧ HomeworkFor(h,AI) ∧ ¬Hates(s,h)

There is at least one student who doesn't hate (any of) the AI homework

6. ∀a (IsACat(a) ⇒ Rules(a)) ∧ (IsADog(a) ⇒ Drools(a))

Cats rule and dogs drool

Convert to FOL

1. There are no mushrooms that are poisonous and purple

∀x Mushroom(x) ⇒ ¬(Poisonous(x) ∧ Purple(x))

2. There is a mushroom that is purple and poisonous

∃x Mushroom(x) ∧ Poisonous(x) ∧ Purple(x)

3. There is a bunny who is a cute

∃x IsABunny(x) ∧ IsCute(x)

4. Elder gods do not like Hello Kitty

∀x IsAnElderGod(x) ⇒ ¬Likes(x, HelloKitty)

5. Sister-in-law (your spouse's sister)

∀x,y SisterInLaw(x,y) ⇒ ∃z Female(x) ∧ Spouse(y,z) ∧ Siblings(x,z)

6. There is only one Elvis (do not use ∃!)

∃x IsElvis(x) ∧ ∀y IsELvis(y) ⇒ x=y

7. Every child who has a Chinpokomon card is cool

∀x,y Child(x) ∧ ChinpokomonCard(y) ∧ Owns(x,y) ⇒ Cool(x)

8. Some students took AI1 in Spring 2008 (using Take(person, course, semester))

∃x Student(x) ∧ Takes(x, AI1, Spring2008)

9. Every student who takes AI1 passes it

∀x,y Student(x) ∧ Semester(y) ∧ Takes(x, AI1, y) ⇒ Passes(x, AI1, y)

Is this correctly written?

1. Peter has at least two children.

∃x,y ParentOf(Peter, x) ∧ ParentOf(Peter, y)

No.

Consider the array of kids [Alice, Bob, Charles]. Peter is the father of Alice and no one else. Now consider the equivalent C (pseudo)code

2. Someone at Stanford is smart

∃x At(x, Stanford) ∧ isSmart(x)

Yes, this one is fine.

Convert to English

3.∀x At(x, Berkeley) ⇒ isSmart(x)

Everyone at Berkeley is smart.

If you're wondering why i used Berkeley it's because this is one of the rare occasions that i used someone else's example, in this case Russel and Norvig. You can guess what school one of the authors is from.

4.∀x At(x, Berkeley) ∧ isSmart(x)

Everyone is at Berkeley. Everyone is smart.

This is probably not what you wanted to write. It's also a very common mistake. Remember – use ⇒ with ∀, not ∧.

5.∃x At(x, Stanford) ⇒ isSmart(x)

Contrary to what many people think, this statement does not say that there is a person at Stanford who is smart. It says that if there is at least one person who, if he is at Stanford, is also smart. And if he isn't, who knows. While it's possible the author meant this, in most cases this is an error. Remember – use ∧ with ∃, not ⇒.

Applying implication elimination, you could also write: ∃x ¬At(x, Stanford) ∨ isSmart(x)

which means there exists at least one person who either doesn't go to Stanford or who is smart (whether he's at Stanford or not).