

Materials: Projectables of

1. Winston (2nd ed) pp. 267, 268, 269
2. Luger Figure 7.2
3. Luger Figure 7.5
4. Luger p. 236, Figure 7.8
5. Luger Figure 7.15
6. Luger Figure 7.22

Handout: An ISA hierarchy implemented in Prolog; Demonstration to run

I. Introduction

- -----

- A. Earlier in the course, we looked at the Predicate Calculus as an example of a scheme for representing knowledge. However, the very flexibility of the Predicate Calculus turns out to be one of its major weaknesses.

Example: How do I represent the statement "Garfield is orange"? Two possibilities are:

```
color(garfield, orange)
orange(garfield)
```

1. The first representation would make it easy to answer the question "what color is Garfield?".
 2. However, constructing the second representation does not require that I know that (in this context), "orange" is a color. It can thus be constructed from the original sentence, without making use of additional information about the meaning of "orange".
 3. However, the second representation could be understood as representing many different statements - including the statement "Garfield is an orange"!
- B. Today, we focus on mechanisms for representing ASSOCIATIONS between entities. That is the essence of the issue in the example we just looked at - what is the nature of the association between "Garfield" and "orange"?
- C. In fact, one approach to way to approach questions of semantics (meaning) is through an associationist theory of meaning. In an associationist theory, the meaning of a word is wrapped up in its association with other words.
1. As the book pointed out, dictionaries generally define words this way.
 2. We've probably all been in a conversation where someone has used a word we did not understand - but were able to figure out at least something of its meaning from the context in which it was used.
- D. We will look at a number of approaches to representing various forms of associations that have been developed in the history of AI. (We will take them in a slightly different order from that in the book.)

1. Inclusion (ISA) Hierarchies
 2. Frames
 3. Semantic Networks
 4. Scripts
 5. Conceptual graphs.
- E. A few general comments.
1. With any of these schemes, a "lower level" system (e.g. Prolog or LISP or Predicate Calculus) is used for representing the basic information. However, we will focus on the overall structure, not the low-level details of how a structure could be represented.
 2. We are not, by any means, exhausting the range of possible knowledge representation schemes. More can be said about any of these schemes, and there are others as well that could be considered.
 3. Of course, we must not think that, for any given problem, we are limited to just one representation scheme. Many problems can best be solved by using a combination of schemes.

II. Inclusion (ISA) Hierarchies

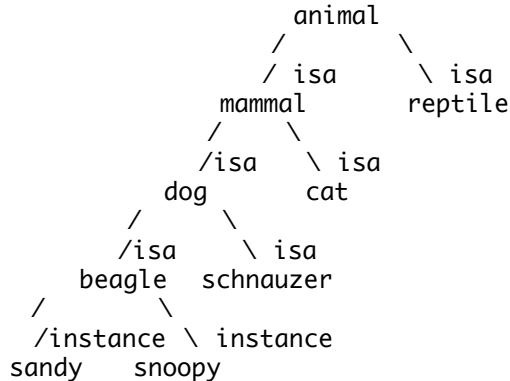
-- -----

- A. One common characteristic of much of human knowledge is that it is hierarchical in nature. Humans are, by nature, classifiers of knowledge into categories, subcategories, etc. As a result, what one knows about a given object includes not only the specific things one knows about the specific object, but also things one can infer from the categories to which it belongs.
1. Example: My first dog, Sandy, was a beagle.
 - a. Without ever having met her, you can probably infer some things about her, such as her size, general coloration, etc. If you are knowledgeable about beagles, you may also be able to infer that she was very tolerant of children and liked to roam all over the neighborhood whenever she managed to sneak out of the house. Both of these are general characteristics of beagles which also were specific characteristics of Sandy.
 - b. From the fact that Sandy was a dog, you can also infer some things about her, such as the fact that she had four legs, her attitude toward cats, etc.
 2. Actually, the above inferences made use of two similar but distinct sort of relationships:
 - a. The relationship between Sandy - a specific beagle - and beagle - a class of animal.
 - b. The relationship between beagle - a class of animal, and dog, a more inclusive class of animal of which the class beagle is a subclass.

- c. If classes such as beagle, dog, are thought of as sets, then the first relationship is, in fact, the set element relationship between an object and a set, and the second relationship is, in fact, the subset relationship between sets.
- d. In English we use the same phrase (e.g. "is a") to express both - i.e. we say "Sandy is beagle" and "a beagle is a dog". However, in formal representation, it is helpful to distinguish the two somehow.
 - i. Various writers make use of pairs of distinct terms - e.g.
 - Rich and Knight: instance and isa
 - Tanimoto: ISIN and ISA
 - Charniak/McDermott: INST and ISA
 - Winston ISA and AKO (a-kind-of).
 - ii. Luger does not discuss the topic of "isa" hierarchies per se, and the system of notation he does focus on (conceptual graphs) actually represents these two types of information in two very distinct ways.
 - iii. I will use Rich/Knight's nomenclature for the conceptual discussion here, but then will use a distinct notation in the Prolog example I will present.

B. In AI, we can model hierarchically-classified knowledge by a tree-like structure in which the leaf nodes represent individual objects and the internal nodes represent classes of objects

1. Ex:



- 2. In such a structure, the individual nodes could be represented by using one of the other knowledge-representation schemes (e.g. predicate calculus facts or frames.) The instance and isa links could be represented by using slots in the frame (which will actually be filled with a pointer to the parent), or by using separate instance and isa predicate-calculus facts.
 - a. In contrast to most tree representations in which the parent contains pointers to the children, in isa hierarchies the most useful sort of pointer tends to be a pointer from a child to its parent. Thus, sandy would have an instance link to beagle, which in turn would have an isa link to dog, etc.

Example: suppose we used a three-place pet predicate (name, owner, instance) and a three-place class predicate (name, properties, isa) to represent the above. Then we would have facts like:

```
pet(sandy, bjork, beagle)
pet(snoopy, schultz, beagle)
...
class(beagle, [roams, good_with_kids], dog)
...
```

- b. If we used separate predicate calculus facts, then the hierarchy above could be represented by:

```
instance(sandy, beagle)
instance(snoopy, beagle)
isa(beagle, dog)
isa(schnauzer, dog)
...
```

- c. Sometimes, it is useful to store the hierarchy to facilitate moving both up and down - e.g. not only might we want to know that a beagle is a kind of dog, but also we might want to generate a list of all kinds of dogs, or even of all individual dogs.

i. Using predicate calculus instance and isa facts, we already have everything we need to be able to do this.

ii. If we used frames with instance/isa slots to store the basic hierarchy, then in addition class nodes might store a list of child pointers under a slot name such as includes. For example, the includes list for dog would be (beagle schnauzer).

3. Notice that it is possible to easily answer certain questions about objects by traversing the hierarchy. For example:

a. Is sandy a mammal? - Answered yes by traversing instance/isa links from sandy to mammal.

b. Is sandy a cat? - Here, we could only conclude that we have no reason to believe that she is, because traversing the chain of instance/isa links from sandy never encounters a cat node. (We do NOT, however, know for certain that sandy is NOT a cat. Notice the distinction between not knowing that she is a cat and knowing that she is not a cat.)

C. Inheritance of Properties

1. A structure such as we have been developing resembles an OO class hierarchy, though here the hierarchy is one of categorization, not one of data structure.
2. One of the major advantages of an isa hierarchy is that it allows an object to inherit attributes from the classes of which it is a member. (This is similar to the notion of inheritance in OO, but here we are inheriting properties - not fields and methods - The existence of an isa link from - say - dog to - say - mammal does not say that the fields of a dog object are a superset of the fields of a mammal object, as in OO, but rather that a dog inherits the PROPERTIES of a mammal.)

- a. We have already seen one example of inheritance - Sandy inherits membership in the class dog from the class beagle of which she is a member, and inherits membership in the class mammal from the class dog.
 - b. But we can infer more than this. Suppose we associated the attribute "has four legs" with the class dog. Sandy would, of course, inherit this attribute as well.
 - c. By using inheritance, we can we can avoid redundant storage of information by associating attributes with the largest possible class. For example, in the above hierarchy, we could store:
 - i. The property "has warm-blood" with mammal - which would result in its being inherited by all dogs, cats etc.
 - ii. The property "has four-legs" with dog and also with cat. (We can't store it with mammal since some mammals do not have four-legs - e.g. whales, humans.)
 - iii. The property "owned-by bjork" with Sandy - this is unique to her and does not belong to any bigger class.
 etc.
2. Not all classification schemes are hierarchical like the example we have been working with, of course.
- a. For example, I am a husband, father, college professor, etc.
 - i. Certain things can be inferred about me from each of these relationships. I inherit attributes from each. This is called MULTIPLE INHERITANCE.
 - ii. However, these classes certainly cannot be fit into a simple hierarchical scheme (all are subclasses of human but none is a subclass of another.)
 - b. Such situations - which occur often in the real world - are handled by isa hierarchies that are graph structures, rather than trees.
 - c. The major difference between this sort of structure and the simple tree-like hierarchy is that either
 - i. There will be multiple instance or isa facts for a given individual or class
 - ii. Or, if pointers are used to maintain the hierarchy, then the instance and isa slots of objects and classes will contain lists, rather than single values.
 - d. The value of multiple inheritance in OO is debated; however, multiple inheritance of classifications seems to be hard to avoid.
3. You will have an opportunity to work with inheritance of properties in a later project.

4. Though inheritance is simple in principle, there is one sticky issue we must address. Sometimes a property is shared by almost all, but not fully all of the members of a class.
 - a. For example, almost all birds fly, but penguins and ostriches do not. We could handle this by attaching the "can fly" attribute to each individual variety of bird, save penguin and ostrich. However, this is bad for two reasons:
 - i. It requires a lot of redundant storage of information.
 - ii. If we are introduced to a new, previously unknown species of bird, we would not be able to make the reasonable assumption that it probably can fly.
 - b. A similar problem can arise using multiple inheritance if values inherited from two different super classes conflict. For example, Tanimoto cites the case of a duck decoy which is both a duck (of sorts) and a wooden object. From the fact that it is a duck - and thus a bird - one might infer that it has a head, bill etc (which is ok), but also that it has a heart (which is not ok, since it conflicts with what is known from the fact that it is a wooden object.)
 - c. Problems like this can be handled by associating the general attribute with the superclass, and exception indicators with selected subclasses. For example, we could associate the attribute "can fly" with the class bird, and an overriding property "cannot fly" or not "can-fly" with penguin and ostrich. (This is called CANCELLATION OF INHERITANCE).
 - d. To further assist with situations like this, we might distinguish between an absolute value for a slot and a default value.
 - i. For example, for mammals being warm-blooded is always true.
 - ii. Having four legs might be a default value for a mammal, which can be over-ridden in certain specific cases.

D. DEMONSTRATION: an isa hierarchy implemented in Prolog (HANDOUT)

1. For the purposes of this program, I have chosen to use "isa" to represent both the "element of" and the "subset of" relationship, and have instead represented individuals by terms of the form individual(...) - e.g. individual(snoopy) and classes by terms of the class(...) - e.g. class(beagle). This allows me to avoid writing separate inference rules for the two kinds of relationship.
2. Go over knowledge base
3. Go over inference engine
4. Demo: infer(property(individual(garfield), owner(X))).
 infer(property(individual(garfield), eat(X))).
 infer(property(individual(nermal), purr)).
 infer(property(individual(garfield), purr)).
 infer(property(individual(tweety), fly)).
 infer(property(individual(chilly_willy), fly)).
 Various examples with alli and allc.

III. Frames

- A. We have seen that one of the things intelligent beings like us do with the entities in our world is we organize them into categories. For each category of entity, we tend to want to know certain things.
 - 1. Example: someone tells us about a car he/she just purchased. We want to know things like manufacturer, model, year, and color.
 - 2. Example: someone tells us about his/her pet dog. We want to know things like its breed etc.

- B. Object-orientation has become a widely used paradigm for modelling objects in the real world, based on the use of object structures with fields for recording different pieces of information.
 - 1. e.g.

```
class Auto
{
    String make;
    String model;
    ...
}
```
 - 2. When structures such as these are used in AI programs, they are often called FRAMES OF CONTEXT or just FRAMES.
 - a. Historically, frames in AI developed in parallel with OO in software engineering. An AI frame, then, has many similarities to an object in OO - though AI sometimes uses a more flexible representation (e.g. a network of nodes) rather than the more rigid object structure of most OO languages.

In this lecture, we will use the traditional AI terminology, though many concepts also appear in OO under different names.
 - b. A frame is a structure describing some object. Different pieces of information about the object are represented as values of the various SLOTS in the frame.
 - c. The general frame structure for a class of objects is called a SCHEMA (plural SCHEMATA if there are several different classes of objects to model), and the particular frames for particular objects are called INSTANCES of the schema.
 - d. Example: Cars might be described by a car schema with slots for make, model, year, and color.
 - e. In keeping with a historical representation for frames in LISP, we sometimes give slots names beginning with a colon - e.g

:make, :model, :year, :color.

- C. As we have discussed frames thus far, they do not seem to be too different from the familiar concept of objects in OO languages (or record structures in data processing.)

However, there are a few characteristics of frames in AI programs that are not typically found in conventional object/record structures (though they could certainly be modelled using these structures.)

1. It is useful for frames to be linked together by instance and isa links to form inclusion hierarchies. Likewise, it is often useful for frames of different types to be linked together by other kinds of links as well that represent relationships between the objects represented.
 - a. While such links occur in OO and ordinary data processing, a distinctive of AI programs is that different objects of the same type may be linked to other objects in different ways.
 - b. Sometimes the term "frame" actually refers to a structure of nodes linked together, rather than to a single node.
2. One thing that is often characteristic of knowledge in AI programs is that it is partial. For some particular auto, I may know the make, model, and color, but not the year.
 - a. In this case, the :year slot might contain an indication that the value is unknown.
 - b. When a value is put in a slot, we say that the slot is FILLED IN. This may occur at the time the frame is created, or later as further information about the object becomes known.
 - c. Several mechanisms are commonly used in AI programs to deal with incomplete information:
 - i. Default values for slots - values to be assumed to hold for a particular instance if the slot has not been filled in.
 - ii. If-needed procedures for slots - an if-needed procedure is a procedure to be invoked to attempt to infer the value for a slot (based on information in other slots) if the information is needed at some point. (Of course, such a procedure might fail to produce a value, too, if the information it needs is missing.)
 - iii. If-added procedures for slots - an if-added procedure is a procedure to be invoked when a formerly empty slot is filled in. This procedure may fill in other slots based on the new knowledge.
 - iv. Such procedures are sometimes referred to as "daemons" [where the term in this context does not refer to fallen angels, but rather picks up a Unixism referring to background programs that do not run under the direct control of a user].
 - d. Example: suppose we were to create a schema for a student-frame, describing a college student. One slot in this frame might be his/her grade-point average (:gpa). This slot would initially be vacant when a student frame is created, since students don't have a gpa until they've taken at least one course. Further, the value in this slot would change at the end of each semester in which the student takes at least one course.

- i. Since the gpa calculation involves a fair amount of computation, we might choose to not do it regularly. Instead, we might associate with the gpa slot an if-needed procedure that calculates the gpa when someone tries to use it. (Presumably, this procedure would then fill in the slot with the calculated value so that recomputation is not necessary the next time.) Of course, this procedure would have to return something like null if the course history information is lacking.)
 - ii. Since any change to the student's course history would change his/her gpa, we might associate with the courses-taken slot an if-added procedure that invalidates the gpa slot (sets it back to unknown) any time a change is made to the courses-taken slot. This will force a fresh computation of gpa the next time it is needed. (Or, at higher computational cost, we might have the if-added procedure re-compute the gpa and put the new value in the gpa slot.)
- e. Note that default values, if-needed procedures, and if-added procedures for slots are really properties of the SCHEMA, not of the actual frame itself.
- i. Some writers (e.g. Tanimoto) call these ATTACHMENTS to the slots to distinguish them from the actual slot values.
 - ii. Another term sometimes used is FACETs. A given slot may have a value facet, a default facet, an if-needed facet etc.
 - iii. Procedural attachments that are run as a side-effect of altering the value of a slot are sometimes called DEMONS.

D. An example of how frames can be used in problem-solving.

1. In his intro AI text, Patrick Henry Winston discusses the use of frames to model the structure of stereo-typed news stories, such as disaster stories. This method has been used in actual programs.
2. In this case, the frame structure, together with if-needed procedures for each slot, provides a mechanism for extracting key features from a story.

PROJECT: Winston p. 267, 268 (top part + pattern)

3. This mechanism could digest a news story and produce a summary like the following:

PROJECT: p. 268 (story) + p. 269 (figure + first summary)

4. Of course, it could also produce some strange results:

PROJECT: p. 269 (second story)

IV. Scripts

-- -----

- A. A knowledge representation scheme closely related to frames is scripts. Scripts are frame-like structures used when there is a time-order relationship between the slots.

1. Example: Consider the following story (from Rich/Knight):

"John went out to a restaurant last night. He ordered steak. When he paid for it, he noticed that he was running out of money. He hurried home since it had started to rain."

Based on the above story, did John eat dinner last night?

2. The fact that we routinely answer yes to such a question implies that we are using a structure of knowledge with some default information that fills in for things not explicitly stated. The story does NOT say that John actually ate dinner - but it does say that he went to a restaurant, ordered steak, and paid for it. Anyone hearing the story would assume that he ate the steak.
3. If you think about typical human story-telling, you realize that we seldom fill in all the details of a story. Rather, we tell some of the details, relying on our listeners to fill in the rest.
- a. For example, in the story we just looked at, it is not necessary to explicitly say that John ate the steak; and, in fact, we would not normally say so unless there was something unusual about the way he ate it.
- b. To see this latter point, consider the following two variants of the above story:

"John went out to a restaurant last night. He ordered steak. When the waiter brought it to the table, John ate it. Then he hurried home since it had started to rain."

"John went out to a restaurant last night. He ordered steak. When the waiter brought it to the table, John bolted it down. Then he hurried home since it had started to rain."

The first of these variants sounds a bit unnatural, since there is nothing unusual or interesting about the fact that John ate the steak. The second is not unnatural, since the fact that he bolted it down is a bit unusual - one normally savors a steak.

(By the way, did John pay for his meal in these stories? We're not told that he did, but we wouldn't think to question the fact that he did so.)

- B. Some of the more fruitful work in natural language understanding relies on structures called SCRIPTS that describe routine events such as going to a restaurant. Those who work with scripts generally assume that something similar is part of the way we understand stories.

1. These scripts contain various slots pertaining to the sequence of actions that normally occur in such an event. For example, the following is an abbreviated restaurant script:

Customer enters restaurant
 Customer sits down at a table
 Waiter brings customer a menu
 Customer orders meal
 Waiter brings order to cook
 Cook prepares food
 Waiter brings food to customer
 Customer eats food
 Waiter brings customer check
 Customer pays bill Customer leaves tip for waiter
 Customer leaves restaurant

(Note: the order of bill paying and tip leaving may go either way.)

2. It is assumed that when we listen to a story, something early in the story triggers a mental script that we use to help us understand what we are hearing. For example, in the story we just considered the opening sentence "John went out to a restaurant last night" would trigger the restaurant script. In like manner, a natural language understanding program would use keywords to trigger appropriate scripts.
3. Once a script has been selected, the program can maintain a pointer to the script which it advances as different items are encountered.
 - a. As appropriate, slots can be filled in with information provided in the narrative. For example, the narrative may provide as to where the customer sat, what he ordered, etc.
 - b. Of course, typically many items in the script will not be mentioned in a given narrative. Thus, the pointer to the script will advance by jumps from one item explicitly mentioned to the other. Items skipped over may be assumed to have occurred by default.
 Example: in our first story, the pointer would skip from Customer enters restaurant to Customer orders food to Customer leaves restaurant.

C. Example of a script-based natural language understanding program:
 W. Lehnert demo at MIT alumni day.

V. Semantic Networks

-- -----

- A. The isa hierarchies we have been working with are a form of network structure in which individual nodes (representing objects or classes of objects) are interconnected by labelled links (instance and isa.) As such, they form a very simple example of a powerful sort of representation structure called a semantic network.
- B. A semantic network is a network of nodes - representing objects or concepts - connected by labelled links.
 1. It is called a semantic network because the links in the network are labelled with semantic meanings.

2. Such networks were first developed in conjunction with efforts to represent the meaning of sentences, using an associationist theory of meaning.

C. Example: PROJECT - Luger Figure 7.2

1. We could represent the network by letting each node be a frame, with slots to hold links to other frames.
2. Alternately, we could use predicate calculus facts - e.g. the example we just looked at:

```
instance_of(frosty, snowman).
made_of(snowman, snow).
form_of(snow, water).
hardness(snow, soft).
...
```

or - alternately - we might use:

```
link(instance_of, frosty, snowman).
link(made_of, snowman, snow).
link(form_of, snow, water).
link(hardness, snow, soft).
...
```

(The latter form would support inferences in which the link label is a variable.)

D. One of the key issues in developing semantic network representations is the choice of labels for the links.

1. We want the set of possible labels to be large enough to capture all possible kinds of relationship, but not so large as to be ridiculous. For example, we would probably not want a link like:

```
anthony ----- cs371
      taking_at_320_mwf_spring_06
```

2. Some of the most fruitful work has come from researchers who have sought to develop relatively small sets of STANDARDIZED link labels that are still rich enough to capture the full range of possible meanings.
3. One approach is the CASE FRAME approach, based on the grammatical notion of case. In this approach, links are labelled by case roles, such as subject, object, etc.

Example: PROJECT - Luger Figure 7.5

4. Another well known system, based on the work of Roger Schank at Yale, known as "Conceptual Dependency Theory".
 - a. This has become a very highly developed system of notation that allows English sentences to be translated into a canonical network representation.

- b. Perhaps the most widely known aspect of the theory is the idea that all actions (verbs) can be represented in terms of 11 "primitive acts". A sentence, then, can be represented as one of these primitive acts linked to appropriate constituents such as an actor, agent, recipient etc.
 - c. Schank's primitive acts are as follows:
(PROJECT P. 236 of Luger)
 - d. Using the notation scheme developed by Schank, one can represent the meaning of a sentence using a "canonical form".
PROJECT Luger Figure 7.8 - GO OVER
 - e. One can then say that two sentences have the same meaning if their canonical representation has the same form
5. As Luger noted, a representation scheme like this faces several significant problems
- a. Translating from natural language into a canonical representation is not straightforward and appears - in general - to not be computable.
 - b. There is no evidence that humans store their knowledge in a form like this.
 - c. The primitives themselves may be too inflexible. For example, how does one represent a fuzzy concept such as "tall".

VI. Conceptual Graphs

-- -----

- A. The book develops in some detail a form of semantic net representation known as a conceptual graph.
- B. The basic idea is this:
 - 1. A proposition is represented by a bi-graph composed of two types of nodes:
 - Concept nodes (shown as rectangles)
 - Relation nodes (shown as ovals)
 - a. Concept nodes only connect directly to relation nodes and vice versa
 - b. Arcs are not labelled
 - 2. Every concept node can include a type specification. A concept node may also include an individual specification - either a specific individual, or one of several possible generic notations.

Note an important syntactic difference from UML object diagrams. In a UML diagram, the notation is individual : class; in a conceptual graph it's class : individual!
 - 3. Type information is represented by a separate type lattice - i.e. "isa" relationships between types are not represented by conceptual graphs. (This is how conceptual graphs distinguish "instance of" and "isa".

3. Example: Figure 7.15 p. 249

C. Conceptual graphs support four basic kinds of operation

1. Copy
2. Restrict - replace a type by a subtype, or a generic marker by a specific marker. This is akin to unification in preparation for the next operation.
3. Join - combine two graphs that have a node in common
4. Simplify - eliminate duplicate nodes

PROJECT Luger Figure 7.22

VII. Representing Common Sense

- A. One of the challenges for any knowledge representation scheme is how to represent "common sense" or "naive" knowledge - i.e. the things that most people know as a result of having lived in the world, without necessarily having learned an explicit statement of them. In fact, it is generally much harder to find ways to represent this kind of information than it is to find ways to represent more technical expert knowledge.

EXAMPLES (from Nilsson):

"If you drop an object, it will fall. (Nowadays, a ten-year-old might also know that objects wouldn't 'fall' if dropped in an orbiting satellite."

"People don't exist before they are born."

"Fish live in water and die if they are taken out."

"People buy bread and milk in a grocery store."

"People typically sleep at night."

- B. One interesting approach to this is a project called CYC - now over 20 years old - that is being carried out under the leadership of Douglas Lenat in Austin, Texas.

1. The project, as described in an article in the August, 1990 CACM, is "a bold attempt to assemble a massive knowledge base (on the order of 10^8 axioms) spanning human consensus knowledge".

2. The developers of CYC are releasing an open version plus a much richer commercial version - see:

www.cyc.com and www.opencyc.org

The most recent version of opencyc contains 47,000 concepts and 306,000 assertions, represented in a predicate-calculus-like language called cycl

3. The CYC project is controversial. (Note, for example, that there is not even an entry under CYC in the index of our textbook).