Chapter-4 (of book) &
Chapter-3 (of SITU syllabus)

Saathi

Date __/__/____ Knowledge repre^{n} issues

→ In this chapter, we look in more detail at the ways of representing knowledge, so that it becomes clear and particular knowledge repre^{n} models allow for more specific & powerful problem-solving mechanism.

→ Here, we shall study specific techniques that can be used for representing & manipulating knowledge within programs.

4.1 ☆ Representations & mappings.

→ To solve complex problems considered in AI, we need both a large amount of knowledge & some mechanism to properly use that knowledge to create soln^{n} of new problems.

→ A variety of ways of representing knowledge (facts) have been used in AI programs.

→ But before we can talk about them individually, we must consider foll. points. We are dealing with two diff. kinds of entities:-

- Facts:- Truths in some relevant world. These are things that we want to represent.
- Repre^{n} of facts in some chosen formalism:- ~~These things~~ We will manipulate these things.

→ One way to structure these entities is at two levels:-

(1) The knowledge level:- At which each facts are described.

(2) The symbol level:- At this level repre^{n} of ~~knowledge~~ objects

at knowledge level are defined in terms of symbols that can be used by programs.

→ ⊗ In our discussion, we will follow a model as shown in figure. Rather than thinking one level on top of another level, we will focus on facts, their repre^n & its two-way mappings that must exist bet^n them.

→ We call these links as "repre^n mapping". The forward repre^n mapping maps from "facts to repre^n" & backward repre^n mapping goes from "repre^n to facts."

→ One repre^n of facts is common & it deserves a natural language sentence i.e "english" sentence.

→ In this case, we have mapping func^n "from english sentence to repre^n that we are actually going to use" and "from ~~back~~ it back to sentence."

Example

→ Let's look at a simple example using mathematical logic as the repre^n formalism.

→ Sentence : Spot is a dog.

→ The fact represented by english sentence can be represented in logic as :-

$$dog(spot).$$

→ Suppose, we also have logical repre^n of fact that all dogs have tails.

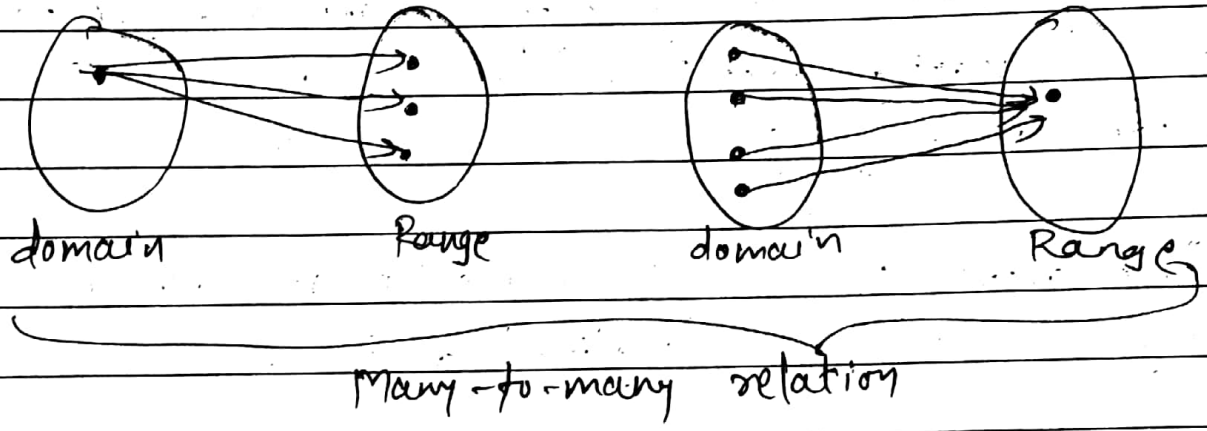
$$\forall x : dog(x) \rightarrow hastail(x)$$

Date ___/___/___

→ Using appropriate backward mapping func^{n}, we could generate the english sentence :

Spot has a tail.

→ It's imp. to keep in mind that usually the available mapping func^{n}'s are not one-to-one. They are many-to-many relations.



→ This is true for english repre^{n} of facts.

ex:

① "All dogs have tails."

② "Every dog has a tail."

same fact → "Every dog has at least one tail."

→ On the other hand,

① "All dogs have tails."
- ① → "Every dog has at least one tail."
- ② → "Each dog has several tails."

② "Every dog has a tail."
- ① → "Every dog has at least one tail."
- ② → "There is a tail that every dog has."

Date __/__/____

→ So, I mean to say, when we try to convert English sentence into some other repre", such as "logical proposition", we must first decide, what fact the sentences represent and then convert those facts into new repre".

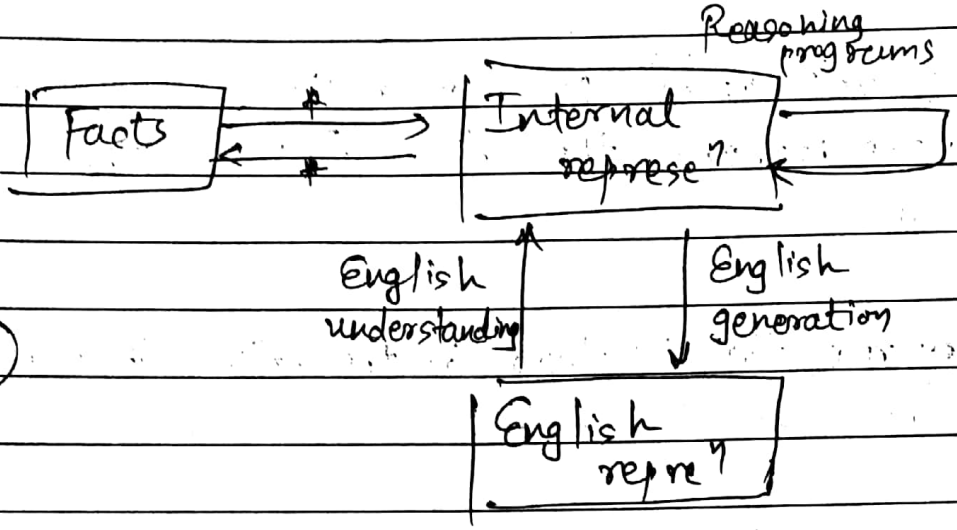


Fig-1

→ The starred links are key components for design of any knowledge-based program, since ~~we need to understand~~ internal repre" of fact plays a good role in program.

→ AI program also manipulates internal repre" of facts which is given. That's why "internal repre"" is that much mandatory.

→ Sometimes, a good repre" makes opera" of reasoning program not only correct but easy.

→ Fig-② shows an expanded view of the starred part of fig. ①. The dotted line across the top represents the abstract reasoning process. The solid line represents the concrete reasoning process that a particular program performs.

→ This program successfully models the abstract process to the extent that, when backward repre^n mapping is applied to program's o/p, the appropriate final facts are actually generated.

→ If program's opera^n or one of the repre^n mapping is not "ok" to the problem, then final facts will not be the desired ones.

→ The key role that is played by the nature of repre^n mapping is visible from this fig.

→ If good mapping can't be defined for a problem, then no matter how good the program is to solve the problem, it will not be able to produce answer that correspond to real answer to the problem.

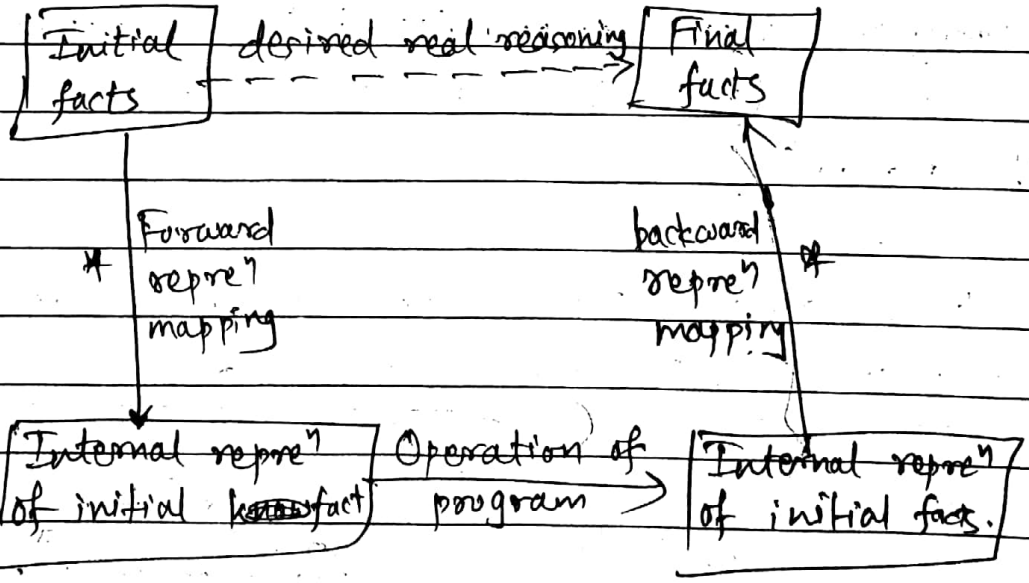


fig. 6

→ For ex, in data type design, it is expected that the mapping that we called the backward repre^n mapping is a func^n (i.e. every repre^n corresponds to only one fact) and that is onto (i.e. there's at least one repre^n for every fact).

→ Unfortunately, in many AI domains, it's not possible to have such a repre^n mapping always. So, we have to live with one that gives less ideal results.

→ But, programmers generally tries to find concrete implementations of abstract concepts.

☆ Approaches to knowledge repre^n (9)

→ A good system for the repre^n of knowledge in a particular domain should possess following four properties:-

(1) Representational adequacy :- The ability to represent all the kinds of knowledge that are needed in the domain.

(2) Inferential adequacy :- ~~To use~~ The ability to use the representation structure in such a way so that we can derive new structure corresponding to new knowledge inferred (obtained) from old.

(3) Inferential efficiency :- The ability to focus the attention of the inference mechanism in the most promising direction.

(4) Acquisitional efficiency :- The ability to acquire new info. easily. The simplest case involves direct insertion, by a person, of new knowledge into the DB.

→ Unfortunately, there's no single system ~~that~~ found which optimizes all the capabilities for all kind of knowledge.

→ Therfore, multiple techniques for knowledge repre^n exists.

Date ___/___/_____

Many programs rely on more than one technique.

Technique-1 Simple relational knowledge

→ The simplest way to represent declarative facts is a set of relations likewise used in DB systems. Fig. shows an example of such a relational system.

Fig. (3) Simple relational knowledge & example facts

Player	Height	Weight	Bats-throws
Hank Aaron	6-0	160	Right-right
Willie Mays	5-10	170	Right-right
Babe Ruth	6-2	215	Left-left
Ted Williams	6-3	205	Left-right

→ The reason that this repre" is simple, since it provides very weak inferential capabilities.

→ But, knowledge represented in this form may serve as an i/p to more powerful inference mechanism.

→ For ex. from above fig., it's not possible to answer simple question "Who is the heaviest player?" But if a procedure for finding the heaviest player is there, then these facts (knowledge) will be useful for procedure to compute an answer.

Procedure and query

→ Instead of that, if we are provided with a set of rules for deciding which hitter (batsman) is best against

a given pitcher (bowler), then this same relation can provide at least some of the info. required by those rules.

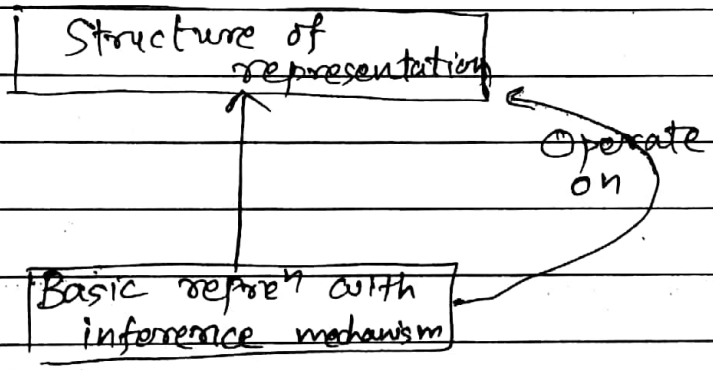
Technique

Inheritable knowledge

→ The relational knowledge of fig. ③ corresponds to a set of attr.s & associated values that together describe the objects of the knowledge base.

→ Knowledge about objects, their attr.s & their values need not be as simple as shown in our example.

→

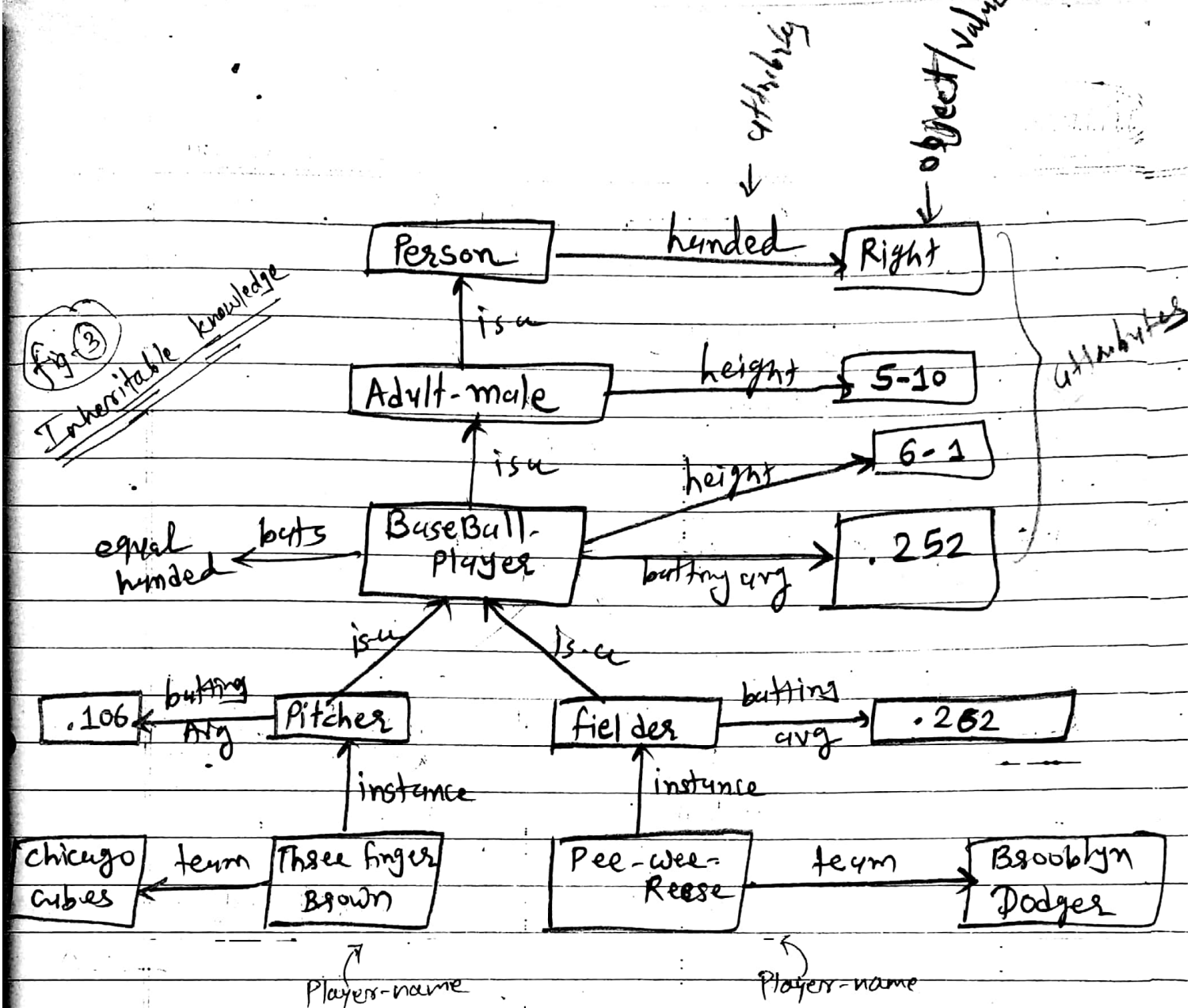


→ For above scenario to be effective, the structure must be designed to correspond to the inference mechanism that are desired.

→ One of the most useful forms of inference is "property inheritance", in which elements of specific classes inherit attr. & values from more general classes in which they are included.

→ In order to support "property inheritance", objects must be organized into classes & classes must be arranged in a generalization hierarchy.

Fig-③ Inheritable knowledge



→ Fig-③ shows some additional baseball knowledge inserted into a structure.

→ Lines represent attributes. Boxed node represent objects & values of attr. of objects. These values can also be viewed as objects with attr.s & values, so on.

→ The arrow points from an object to its value along the corresponding attr. line. This structure in fig.③ is also known as "slot-and-filter structure." It may also be called as a "semantic n/w" or collection of "frames."

Date __/__/____

→ In collection of frame, each individual frame represents the collection of attr.s & values associated with a particular node.

→ Below fig. (A) shows the node for baseball player displayed as a frame.

fig (A)

Base-ball player
is a :- Adult-Male
bats :- (EQUAL handed)
height :- 6-1
batting-average :- 0.252

Viewing a node as a frame

→ Usually the use of the term "frame-system" implies ~~so~~ that, the more structure on the attr.s & the inference mechanism that are available to apply to them than ~~does~~ "semantic N/w".

→ In short, "frame-system" can provide more inference mechanism than "semantic N/w".

→ How these structures support inference using knowledge they contain. We can discuss it briefly here as following.

① All of the objects & most of the attr.s shown here been selected to correspond to base-ball domain, and they have no general significance.

② "is-a" and "instance" are two exceptions to this attr. "is-a" is used to show class inclusion & "instance" is used to show class membership.

instances = Class membership
is-a = Class inclusion

Date ___/___/_____

(3) "is-a" & "instance" attr. provide the basis for property inheritance as an inference technique.

→ An idealized form of property inheritance can be stated as follows:-

* Algo:- Property inheritance:-

→ To retrive a value V for attr. A of an instance object O:-

(1) Find O in the knowledge base.

(2) If there's a value for attr. A, report that value.

(3) Otherwise, see if there's a value for attr. instance. If not, then fail.

(4) Otherwise, move to the node corresponding to that value & look for a value for the attr. A. If one is found, report it.

(5) Otherwise, do until there's no value for the "is-a" attr. or until an answer is found.

(a) Get the value of "is-a" attr. & move to that node.

(b) See if there's a value for that attr. A. If there is, then report it.

→ This procedure is simplistic. It doesn't say what we should do if there's more than one value of the "instance" or "is-a" attr. But it describes the basic mechanism of inheritance.

→ We can apply this procedure to our example knowledge base to derive answers to foll. queries :-

(1) team(Pee-Wee-Reese) = Brooklyn-Dodgers :- This attr. had a value stored explicitly in the knowledge base.

(2) batting-average(Three-finger brown) = 0.106 :- Since there's no value for batting avg. stored explicitly for "Three-finger-Brown", we follow the instance attr. to "Pitcher" and extract the value stored there.

(3) height(Pee-Wee-Reese) = 6-1 :- This represents another default inference. Notice that, since we are coming on that first, and after that we come at the fact that height of base-ball player overrides a more general fact about the height of "adult" males. In short, we considered height = 6-1, bcoz we encountered that only as a first value.

(4) bats(Three-finger-Brown) = Right :- To get a value of attr. "bats" required going up using "is-a" relationship and we have reached to class "Base-ball player". But we have found that, there's no direct value for "attr." "bats" and there's a rule for computing a value. This rule is "handed" and it requires another value. So, now, it is necessary to go all the way up to "Person" and discover default value for "handedness" for people is "Right". Therefore, the result "Right" is applied for rule "bats" via-via (indirectly).

"s" ની value equal-handed છે, પણ ... rule છે, value નહિ, bcoz →

Equal handed means bowling & batting both righty or lefty that means batting અને bowling બંને માં same, પણ સામાન્યતઃ rule નહિ, perfect value જોઈએ છે.

Page No.

Date __/__/____

Sadhna

Inferential knowledge :-

→ Property inheritance is a powerful form of inference, but not always useful form. Sometimes the power of traditional logic (like ∧ and ∨) is necessary to describe inference that are needed.

→ Fig. shows two examples of the use of first-order predicate logic to represent knowledge about "base-ball."

$$\forall x : Ball(x) \wedge Fly(x) \wedge Fair(x) \dashrightarrow Infield\text{-}fly(x)$$

$$\forall x, y : Batter(x) \wedge batted(x, y) \wedge Infield\text{-}fly(x) \rightarrow Out(x)$$

→ In above fig, this knowledge is useless if there is no inference procedure that can exploit it.

→ There are many procedures, some of which reason forward from given facts to conclusion and others reason backward from desired conclusions to given facts.

→ One of the most commonly used procedure is "resolution", which exploits proof by contradiction strategy.

Procedural knowledge

→ Our previous examples of "base-ball knowledge" have concentrated on relatively static & declarative facts.

→ But another useful kind of knowledge is operational or

Date ___/___/___

procedural knowledge, that specifies what to do when.

→ Procedural knowledge can be represented in programs by many ways. The most common way is simply in programming lang. such as LISP for doing something.

→ Unfortunately, this way of representing procedural knowledge gets low score with respect to properties of inferential adequacy & acquisitional efficiency. It gets low score for inferential adequacy, bcoz it is very difficult to write program that can reason about another program's behavior.

→ The most commonly used techniques for representing procedural knowledge is the use of production rules. Fig ⑤ shows an example of a production rule that represents a piece of knowledge possessed by a "baseball players."

If: ninth inning, and score is close, and less than 2 outs, and first base is vacant, and batter is better hitter than next batter,

Then: Walk the batter.