Knowledge Representation

- Introduction
- Logic
- Semantic Networks
- Frames
- Rules

Introduction

Assumption of (traditional) AI work is that:

- Knowledge may be represented as “symbol structures” (essentially, complex data structures) representing bits of knowledge (objects, facts, rules ...).
    - E.g., “red” represents colour red.
    - “car1” represents my car.
    - color(car1, red) represents fact that my car is of red color.
- Intelligent behaviour can be achieved through manipulation of symbol structures

Knowledge representation languages

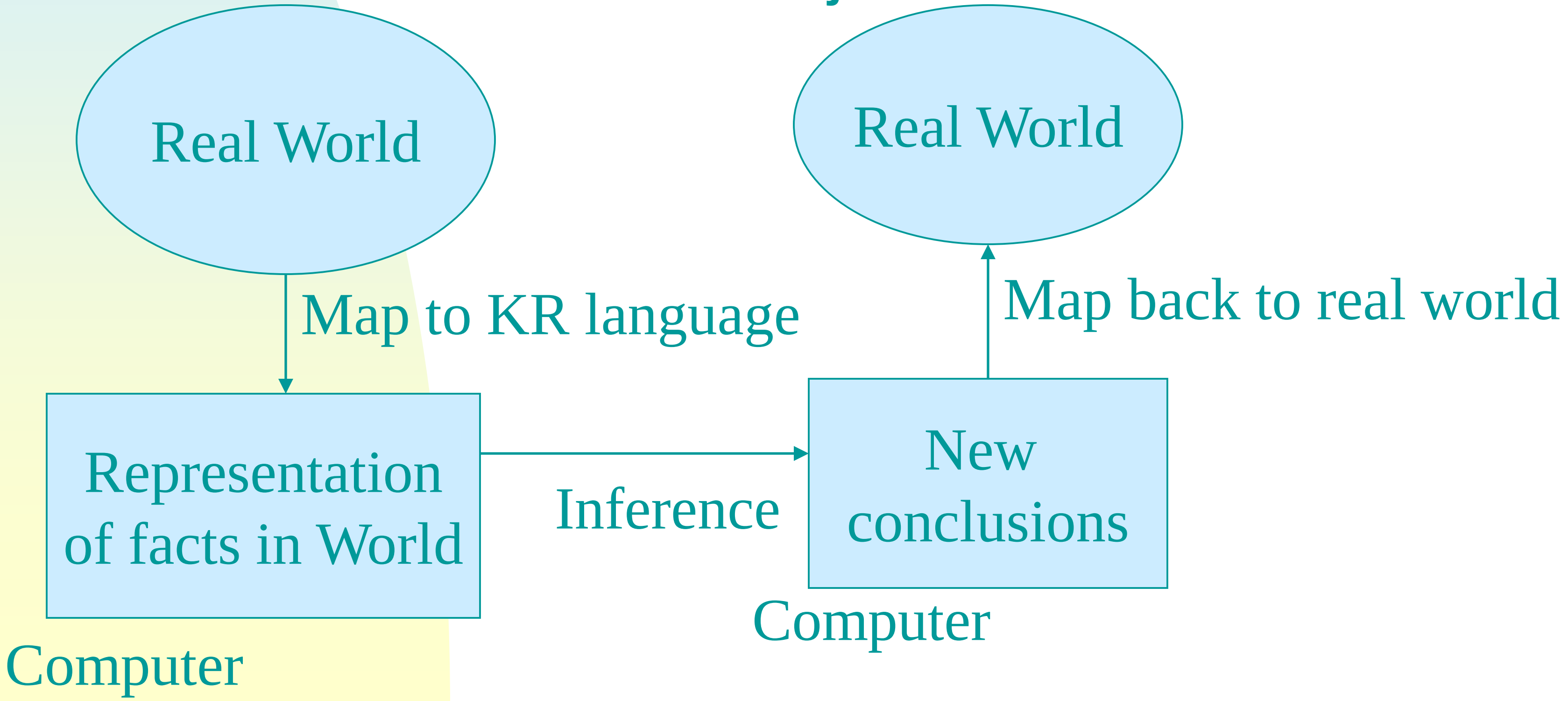
- Knowledge representation languages have been designed to facilitate this.
- Rather than use general C++/Java data structures, use special purpose formalisms.
- A KR language should allow you to:
  - represent adequately the knowledge you need for your problem (*representational adequacy*)
  - do it in a clear, precise and "natural" way.
  - allow you to reason on that knowledge, drawing new conclusions.

Representational adequacy

- Consider the following facts:
  - John believes no-one likes brussel sprouts.
  - Most children believe in Santa.
  - John will have to finish his assignment before he can start working on his project.
- How do we represent these formally in a way that can be manipulated in a computer program?
- Some notations/languages only allow you to represent certain things.
  - Time, beliefs, uncertainty, all hard to represent.

Well-defined syntax/semantics

- Knowledge representation languages should have precise syntax and semantics.
- You must know exactly what an expression means in terms of objects in the real world.



Well defined syntax/semantics

- Suppose we have decided that “red1” refers to a dark red colour, “car1” is my car, car2 is another..
- Syntax of language will tell you which of following is legal: `red1(car1), red1 car1, car1(red1), red1(car1 & car2)?`
- Semantics of language tells you exactly what an expression means - e.g., `Pred(Arg)` means that the property refered to by `Pred` applies to the object refered to by `Arg`. E.g., property “dark red” applies to my car.

A natural representation scheme?

- Also helpful if our representation scheme is quite intuitive and natural for human readers!
- Could represent the fact that my car is red using the notation:
  - "`xyzzy ! Zing`"
  - where `xyzzy` refers to redness, `Zing` refers to by car, and ! used in some way to assign properties.
- But this wouldn't be very helpful..

Inferential Adequacy

- Representing knowledge not very interesting unless you can use it to make inferences:
  - Draw new conclusions from existing facts.
    - “If its raining John never goes out” + “It’s raining today” so..
  - Come up with solutions to complex problems, using the represented knowledge.
- Inferential adequacy refers to how easy it is to draw inferences using represented knowledge.
- Representing everything as natural language strings has good representational adequacy and naturalness, but very poor inferential adequacy.

Inferential Efficiency

- You may be able, in principle, to make complex deductions given knowledge represented in a sophisticated language.
- But it may be just too inefficient.
- Generally the more complex the possible deductions, the less efficient will be the reasoner.
- Need representation and inference system sufficient for the task, without being hopelessly inefficient.

Requirements for KR language: Summary

- Representational Adequacy
- Clear syntax/semantics
- Inferential adequacy
- Inferential efficiency
- Naturalness

In practice no one language is perfect, and different languages are suitable for different problems.

Main KR Approaches

- Logic
- Frames/Semantic Networks/Objects
- Rule-based systems

Logic as a Knowledge Representation Language

- A Logic is a formal language, with precisely defined syntax and semantics, which supports sound inference. Independent of domain of application.
- Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference.
  - propositional logic, predicate logic, temporal logic, description logic..
- But representing some things in logic may not be very natural, and inferences may not be efficient. More specialised languages may be better..

Propositional logic

- In general a logic is defined by
  - syntax: what expressions are allowed in the language.
  - Semantics: what they mean, in terms of a mapping to real world
  - proof theory: how we can draw new conclusions from existing statements in the logic.
- Propositional logic is the simplest..

Propositional Logic: Syntax

- Symbols (e.g., letters, words) are used to represent facts about the world, e.g.,
  - “P” represents the fact “Andrew likes chocolate”
  - “Q” represents the fact “Andrew has chocolate”
- These are called *atomic propositions*
- *Logical connectives* are used to represent *and:* $\wedge$, *or:* $\vee$, *if-then:* $\Rightarrow$, *not:* $\neg$.
- Statements or sentences in the language are constructed from atomic propositions and logical connectives.
  - $P \wedge \neg Q$ “Andrew likes chocolate and he doesn’t have any.”
  - $P \Rightarrow Q$ “If Andrew likes chocolate then Andrew has chocolate”

Propositional Logic: Semantics

- What does it all mean?
- Sentences in propositional logic tell you about what is true or false.
  - $P \wedge Q$ means that both P and Q are true.
  - $P \vee Q$ means that either P or Q is true (or both)
  - $P \Rightarrow Q$ means that if P is true, so is Q.
- This is all formally defined using *truth tables.*

X Y	*X v Y*
T T	*T*
T F	*T*
F T	*T*
F F	*F*

We now know exactly what is meant in terms of the truth of the elementary propositions when we get a sentence in the language (e.g., P => Q v R).

Proof Theory

- How do we draw new conclusions from existing supplied facts?
- We can define inference rules, which are guaranteed to give true conclusions given true premises.
- For propositional logic useful one is modus ponens:

$$\boxed{\frac{A,\ A \Rightarrow B}{B}}$$

- If A is true and A=> B is true, then conclude B is true.

Proof Theory and Inference

- So, let P mean "It is raining", Q mean "I carry my umbrella".
- If we know that P is true, and P => Q is true..
- We can conclude that Q is true.
- Note that certain expressions are equivalent
  - think about P => Q and ¬ P v Q.

More complex rules of inference

- Other rules of inference can be used, e.g.,:

$$\frac{A \vee B,\ \neg B \vee C}{A \vee C}$$

- This is essentially the *resolution* rule of inference, used in Prolog.
- Consider:

$$\begin{array}{c} \text{sunny} \vee \text{raining} \\ \neg\, \text{raining} \vee \text{umbrella} \end{array}$$

- What can we conclude?

Proof

- Suppose we want to try and prove that a certain proposition is true, given some sentences that are true.
- It turns out that the resolution rule is sufficient to do this.
- We put all the sentences into a standard or “normal” form (replacing A => B with $\neg$ A v B)
- There is then a standard procedure that lets you determine if the proposition in question is true.

Predicate Logic

- Propositional logic isn’t powerful enough as a general knowledge representation language.
- Impossible to make general statements. E.g., “all students sit exams” or “if any student sits an exam they either pass or fail”.
- So we need predicate logic..

Predicate Logic

- In predicate logic the basic unit is a predicate/ argument structure called an atomic sentence:
  - likes(alison, chocolate)
  - tall(fred)
- Arguments can be any of:
  - constant symbol, such as ‘alison’
  - variable symbol, such as X
  - function expression, e.g., motherof(fred)
- So we can have:
  - likes(X, richard)
  - friends(motherof(joe), motherof(jim))

Predicate logic: Syntax

- These atomic sentences can be combined using logic connectives
  - likes(john, mary) $\wedge$ tall(mary)
  - tall(john) $\Rightarrow$ nice(john)
- Sentences can also be formed using quantifiers $\forall$ (forall) and $\exists$ (there exists) to indicate how to treat variables:
  - $\forall$ X lovely(X)   Everything is lovely.
  - $\exists$ X lovely(X)   Something is lovely.
  - $\forall$ X in(X, garden) $\Rightarrow$lovely(X)   Everything in the garden is lovely.

Predicate Logic

- Can have several quantifiers, e.g.,
  - $\forall X \; \exists Y \; loves(X, Y)$
  - $\forall X \; handsome(X) \Rightarrow \exists Y \; loves(Y, X)$
- So we can represent things like:
  - All men are mortal.
  - No one likes brussel sprouts.
  - Everyone taking AI3 will pass their exams.
  - Every race has a winner.
  - John likes everyone who is tall.
  - John doesn’t like anyone who likes brussel sprouts.
  - There is something small and slimy on the table.

Semantics

- There is a precise meaning to expressions in predicate logic.
- Like in propositional logic, it is all about determining whether something is true or false.
- $\forall$ X P(X) means that P(X) must be true for every object X in the *domain of interest*.
- $\exists$ X P(X) means that P(X) must be true for at least one object X in the domain of interest.
- So if we have a domain of interest consisting of just two people, john and mary, and we know that `tall(mary)` and `tall(john)` are true, we can say that $\forall$ `X tall(X)` is true.

Proof and inference

- Again we can define inference rules allowing us to say that if certain things are true, certain other things are sure to be true, e.g.
- $\forall$ X P(X) $\Rightarrow$Q(X)
  P(something)
  ------------------ (so we can conclude)
  Q(something)
- This involves matching P(X) against P(something) and binding the variable X to the symbol something.

Proof and Inference

- What can we conclude from the following?
  - $\forall$ X tall(X) $\Rightarrow$ strong(X)
  - tall(john)
  - $\forall$ X strong(X) $\Rightarrow$ loves(mary, X)

Prolog and Logic

- Prolog is based on predicate logic, but with slightly different syntax.
  - a(X) :- b(X), c(X). *Equivalent to*
  - $\forall$ X a(X) $\Leftarrow$ b(X) $\wedge$ c(X) ***Or equivalently***
  - $\forall$ X b(X) $\wedge$ c(X)$\Rightarrow$ a(X)
- Prolog has a built in proof/inference procedure, that lets you determine what is true given some initial set of facts. Proof method called “resolution”.

Other Logics

- Predicate logic not powerful enough to represent and reason on things like time, beliefs, possibility.
  - "He may do X"
  - He will do X.
  - I believe he should do X.
- Specialised logics exist to support reasoning on this kind of knowledge.

Other Knowledge Representation Methods

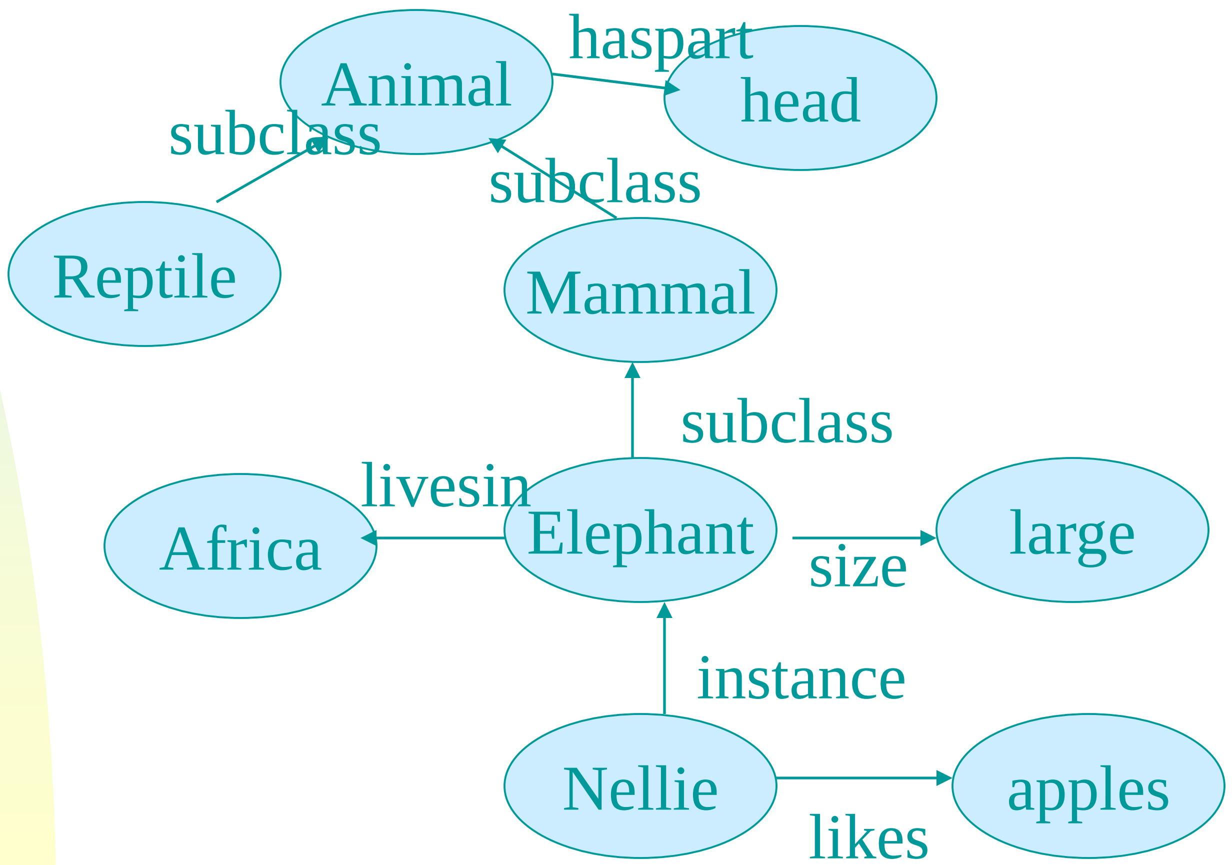
- Logic isn't the only method of representing knowledge.
- There are other methods which are less general, but more natural, and arguably easier to work with:
  - Semantic Nets
  - Frames
  - Objects
  - Rules
- To some extent modern OOP has superceded the first two, with the ability to represent knowledge in the object structures of your programming language.

Semantic Nets etc..

- Semantic nets, frames and objects all allow you to define relations between objects, including class relations (X isa Y).
- Only restricted inference supported by the methods - that based on *inheritance.*
- *So.. Fido is a dog, dogs have 4 legs, so Fido has 4 legs.*

Semantic Networks

- Knowledge represented as a network or *graph*



Semantic Networks

- By traversing network we can find:
  - That Nellie has a head (by inheritance)
  - That certain concepts related in certain ways (e.g., apples and elephants).
- BUT: Meaning of semantic networks was not always well defined.
  - Are all Elephants big, or just typical elephants?
  - Do all Elephants live in the “same” Africa?
  - Do all animals have the same head?
- For machine processing these things must be defined.

Frames

- Frames were the next development, allowing more convenient “packaging” of facts about an object. Frames look much like modern classes, without the methods:

```
mammal:
  subclass: animal

elephant:
  subclass: mammal
  size: large
  haspart: trunk

Nellie:
  instance: elephant
  likes: apples
```

- We use the terms “slots” and “slot values”

Frames

- Frames often allowed you to say which things were just typical of a class, and which were definitional, so couldn't be overridden. Using an asterix to denote typical values:

```
Elephant:
   subclass: mammal
   haspart: trunk
   * colour: grey
   * size: large
```

- Frames also allowed multiple inheritance (Nellie is an Elephant and is a circus animal). Introduces problems in inheritance.

Frames and procedures

- Frames often allowed slots to contain procedures.
- So.. Size slot could contain code to run to calculate the size of an animal from other data.
- Sometimes divided into “if-needed” procedures, run when value needed, and “if-added” procedures, run when a value is added (to update rest of data, or inform user).
- So.. Similar, but not quite like modern object-oriented languages.

Semantic Networks (etc) and Logic

- How do we precisely define the semantics of a frame system or semantic network?
- Modern trend is to have special knowledge representation languages which look a bit like frames to users, but which:
  - use logic to define what relations mean
  - don’t provide the full power of predicate logic, but a subset that allows efficient inference. (May not want more than inheritance).

Implementing simple Frame systems

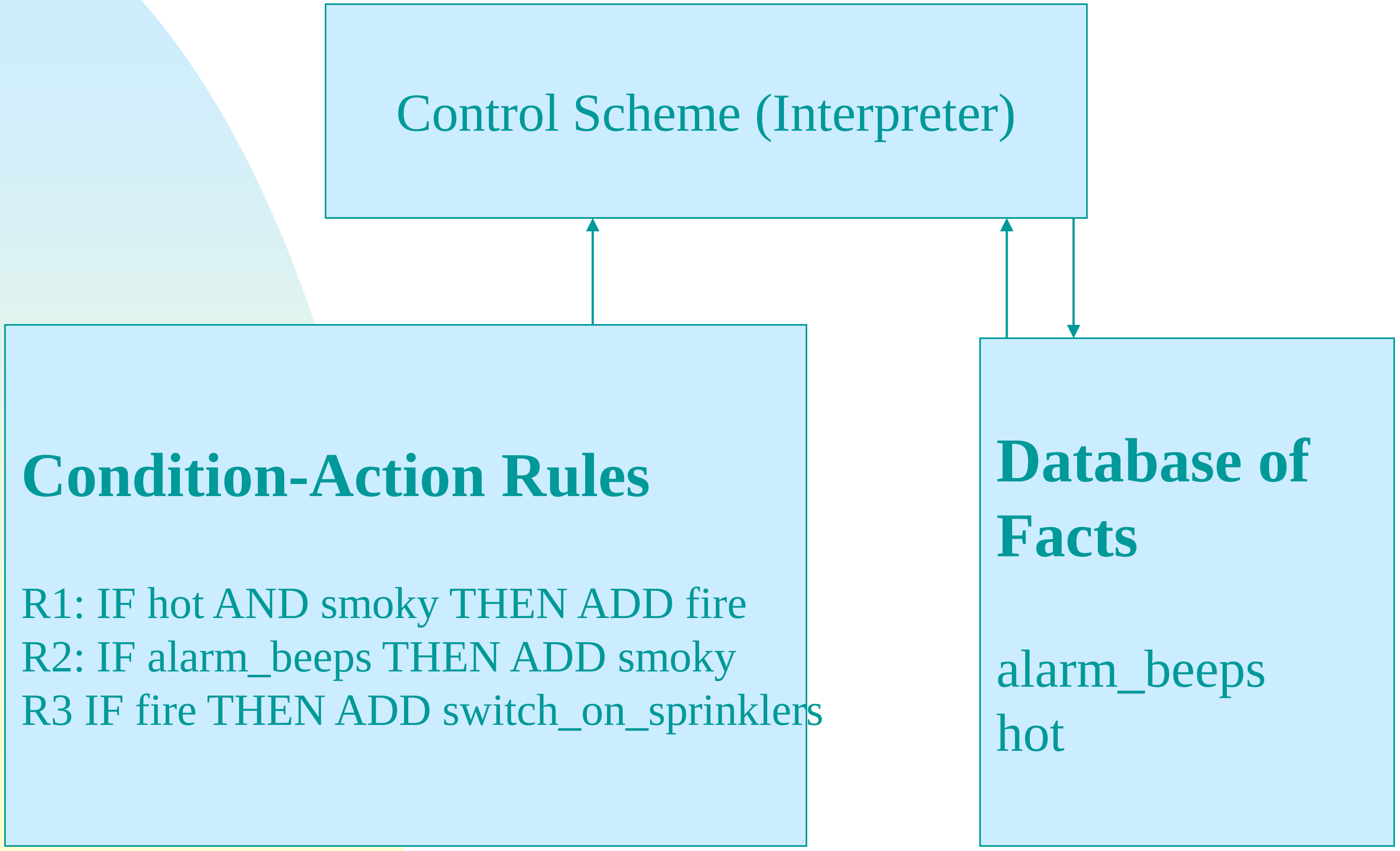
- Sometimes, even when using a logic-based language, it is useful to be able to define inheritance rules, and group object attributes together in a frame-like structure.
- So we could have..:

  -
    ```
    slot-value(elephant, size, large)
    instance(nellie, elephant)
    value(Obj, Slot, V) :-
      instance(Obj, Class),
      slot-value(Class, Slot, V).
    ```

Rule-Based Systems

- In Logic (and frames) we represent knowledge in a declarative, static way - as some facts and rules that are true.
- Rules in logic say what is TRUE given some conditions.
- Rule-based systems are based on rules that say what to DO, given various conditions.
  - IF <this is the case> THEN <do this>
- A special *interpreter* controls when rules are invoked.

Rule-based system architecture


Control Scheme (Interpreter)
Condition-Action Rules
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3 IF fire THEN ADD switch_on_sprinklers
Database of Facts
alarm_beeps
hot

Rules and Logic

- Simple examples are very similar to rules in logic.
- However, in rule based systems we consider:
  - Other kinds of actions (apart from adding facts).
  - Degrees of certainty associated with facts.
  - Various different control schemes (not necessarily related to idea of logical proof).
- Less concern about precise semantics and sound inference.

Control schemes

- Two main kinds of rule-based systems: forward chaining and backward chaining.
- Forward chaining starts with the facts, and sees what rules apply (and hence what should be done) given the facts.
- Backward chaining (much like Prolog) starts with something to find out, and looks for rules that will help in answering it.

Forward chaining

- In a forward chaining system:
  - Facts are held in a *working memory*
  - *Condition-action* rules represent actions to take when specified facts occur in working memory. IF condition THEN action.
  - Typically the actions involve adding or deleting facts from working memory.

Forward chaining

- Control cycle called *recognize-act* cycle.
- Repeat:
  - Find all rules which have satisfied conditions given facts in working memory.
  - Choose one, using *conflict resolution strategies*.
  - Perform actions in conclusion, probably modifying working memory.
- Until no rules can fire, or “halt” symbol added to working memory.

Example

- Simple “fire” example given earlier:
- Working memory initially contains:
  - alarm_beeps
  - hot
- Following the algorithm: First cycle..
  - Find all rules with satisfied conditions : R2
  - Choose one: R2
  - Perform actions: ADD smoky.
  - Working memory now contains
    - alarm_beeps, hot, smoky

Example continued

- Next cycle:
  - Find all rules with conditions satisfied : R1
  - Choose one and apply action: ADD fire
  - Working memory now contains alarm_beeps, hot, smoky, fire.
- Then
  - Rules with conditions satisfied: R3
  - apply action: ADD switch_on_sprinklers.

Forward chaining applications

- Forward chaining systems have been used as:
  - a model of human reasoning
  - basis for expert systems - various *expert system shells* based on this model, such as CLIPS.
- Practical forward chaining systems support *pattern matching* (Prolog)
- Example CLIPS rule:

```
(defrule fire-alarm
    (temperature ?r1 hot)
    (environment ?r1 smoky)
   =>
    (assert (fire-in ?r1)))
```

Backward Chaining

- Same rules/facts may be processed differently, using *backward chaining* interpreter.
- This allows rather more focused style of reasoning. (Forward chaining may result in a lot of irrelevant conclusions added to working memory.)
- Start with possible hypothesis. Should I switch the sprinklers on?
- Set this as a goal to prove
  - Similar to Prolog which uses a backward chaining style of reasoning.

Backward Chaining

- Basic algorithm:
- To prove goal G:
  - If G is in the initial facts, it is proven.
  - Otherwise, find a rule which can be used to conclude G, and try to prove each of that rule’s conditions.

Backward Chaining Example

Should we switch on the sprinklers? Set as a goal.

- G1: switch_on_sprinklers
- Is it in initial facts? No. Is there a rule which adds this as a conclusion? Yes, R3
- Set condition of R3 as new goal to prove:
- G2: fire.
- Is it in initial facts? No. Rule? Yes, R1
- Set conditions as new goals: G3: hot, G4: smoky.

Example continued

- Try to prove G3: hot. In initial facts.
- Try to prove G4: smoky. Conclusion of rule so..
- G5: alarm_beeps.
- In initial facts, so all done…
- Proved hypothesis *switch_on_sprinklers*.

Expert Systems Applications

- Backward chaining systems have also been fairly widely used in expert systems.
- E.g., medical systems, where start with set of hypotheses on possible diseases - try to prove each one, asking additional questions of user when fact is unknown.

Expert Systems (Intro)

- Expert systems aimed to capture specialist human expertise which was in short supply. Eg..,
  - Medical expertise
  - Computer configuration expertise.
  - Expertise for oil exploration.
- Aim was to develop systems capturing this expertise, so the knowledge could be deployed where experts were unavailable.

Expert System Architecture

- Common approach is/was to have knowledge represented as if-then rules.
- Reasoning strategy might be forward or backward chaining.
- An early and simple system which used a backward chaining rule-based system is MYCIN.
- Helped physicians diagnose bacterial infections.

MYCIN

- MYCIN’s “knowledge base” consisted of set of IF-THEN rules, e.g.,

  IF the infection is primary-bacteremia

  AND the site of the culture is one of the sterile sites

  AND the suspected portal of entry is the gastrointestinal tract

  THEN there is suggestive evidence (0.7) that infection is bacteroid.

MYCIN: Reasoning and Problem Solving Strategy

- MYCIN could use backward chaining to find out whether a possible bacteria was to blame.
- This was augmented with “certainty factors” that allowed an assessment of the likelihood, if no one bacteria was certain.
- MYCIN’s problem solving strategy was simple:
  - For each possible bacteria:
    - Using backward chaining, try to prove that it is the case, finding the certainty.
  - Find a treatment which “covers” all the bacteria above some level of certainty.

MYCIN: Problem Solving

- When trying to prove a goal through backward chaining, system could ask user certain things.
  - Certain facts are marked as “askable”, so if they couldn’t be proved, ask the user.
- This results in following style of dialogue:
- MYCIN: Has the patient had neurosurgery?
  USER: No.
  MYCIN: Is the patient a burn patient?
  USER: No.
  …
- MYCIN: It could be Diplococcus..

Modelling Human Diagnostic Strategies.

- Problem Solving Strategy used in MYCIN only works when small number of hypotheses (e.g., bacteria).
- For hundreds of possible diseases, need a better strategy.
- Later medical diagnostic systems used an approach based on human expert reasoning.

Diagnostic Reasoning: Internist

- Internist is a medical expert system for general disease diagnosis.
- Knowledge in system consists of disease profiles, giving symptoms associated with disease and strength of association.
- E.g., ECHINOCOCCAL CYST of LIVER

Finding	Strength
Cough	*1*
Fever	*2*
Jaundice	*4*

Problem Solving in Internist

- Use initial data (symptoms) to suggest, or trigger possible diseases.
- Determine what other symptoms would be expected given these diseases.
- Gather more data to *differentiate* between these hypotheses. Either:
  - If one hypothesis most likely, try to confirm it.
  - If many possible hypotheses, try to rule some out.
  - If a few hypotheses, try to discriminate between them.

Medical Expert Systems Today

- Medical expert systems were quite effective in evaluations comparing their performance with human experts.
- Rather few however are used in practice - Hard to integrate into existing practices.
- The successful ones:
  - Support the physicians decisions, rather than doing the whole diagnosis.
  - Include many useful support materials, such as report generating tools, reference material etc.

Summary

- Predicate logic provides well defined language for knowledge rep supporting inference.
- Frames/Semantic Networks/Objects more natural, but only explicitly support inheritance, and may not have well defined semantics.
- Current trend is either to just use OO, or to use logic.

Summary

- Intelligent systems require that we have
  - ◆ Knowledge formally represented
  - ◆ New inferences/conclusions possible.
- Formal languages have been developed to support knowledge representation.
- One important one is the use of logic - very general purpose way to formally represent truths about the world, and draw sound conclusions from these.

Summary

- Rule-based systems provide way of reasoning on knowledge based on Condition-Action rules.
- Two main ways to perform reasoning: forward or backward chaining.
- Forward: start with facts; Backward: start with hypotheses
- Both can be used in expert systems

Summary: Expert Systems

- Effective systems have been developed that capture expert knowledge in areas like medicine.
- Typically combine rule-based approaches, with additional certainty/probabilistic reasoning, and some top level control of the problem solving process.