Chapter 8

Preserving and Applying Human Expertise: Knowledge-Based Systems

Chapter Objectives

- Recall: KB systems are adept at preserving captured and/or discovered knowledge for later sharing and/or application.
- Introduce the student to the internal operation of knowledge-based systems, including:
  - Knowledge representation
  - Automated reasoning
- Introduce the art of knowledge engineering - how to develop knowledge-based systems
  - the tools
  - the techniques.

Chapter Objectives

- Today it's become much easier to learn about knowledge-based systems, because so many AI knowledge representation techniques from the Lisp Machine days have now been completely embraced and extended by mainstream CS & IT:
  - Object-oriented representations (frames, classes, UML)
  - Class inheritance hierarchies
  - Logical databases (relational and object-relational)
  - Integrated development environments (IDE)
- What you may still find less familiar:
  - Inference engines and shells
  - Knowledge engineering
  - Forward vs backward reasoning
  - Default-based reasoning using frames

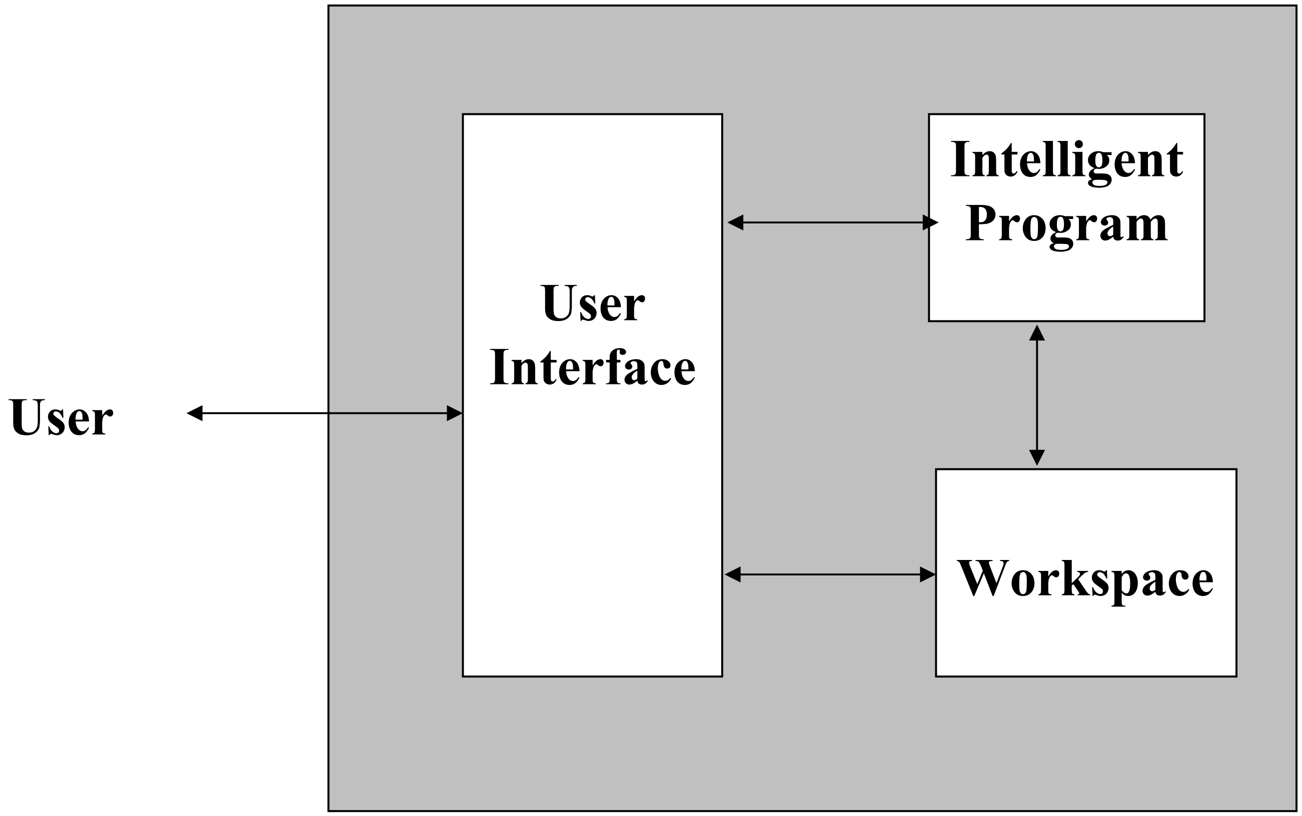
Section 8.2 Objectives

- Introduces a knowledge-based system from the points of view of those that work with them:
  - The user
  - The knowledge engineer
- Introduce the different components of a KBS
  - The inference engine
  - The knowledge base
  - The user interface
  - The fact base
  - The development environment

Knowledge-Based System: User's View

- From end-user's perspective, KB system has three components:
  - Intelligent program
  - User interface
  - Problem-specific database ("workspace")

Knowledge-Based System: User's View



Knowledge-Based System: User's View

- Intelligent program
  - A black box, to the end user
  - Encapsulates most of the knowledge, including possibly knowledge representations of rules, frames, defaults, is-a and has-a hierarchies, etc.
  - Typically rarely necessary for end users to access, and often dangerous from a security standpoint

Knowledge-Based System: User’s View

- User interface
    - May enable the intelligent program to pose questions to the user about the problem at hand
    - May provide explanations about why the intelligent program is asking particular questions
    - May allow the user to query the intelligent program as to why or how a particular decision was made
    - Displays results
    - May provide graphic representations of the results (e.g., decision tree paths, parts of the instantiated class hierarchy, etc.)
    - May allow the user to save or print results
    - …

Knowledge-Based System: User's View

- Problem-specific database ("*workspace*")
  - "Database" in this context refers not to some flat or relational database, but rather a logical representation of known facts (beginning with axioms, from which inferences are drawn)
  - The working space where the system reads any *inputs* and writes its *outputs* (in logic form)
  - The *inputs* consist of all information provided either automatically or by the user via the UI
  - The *outputs* consist of all conclusions the intelligent program is able to drive, including both the final solution required by the user, as well as intermediate conclusions that act as stepping-stones in the path to the final conclusion

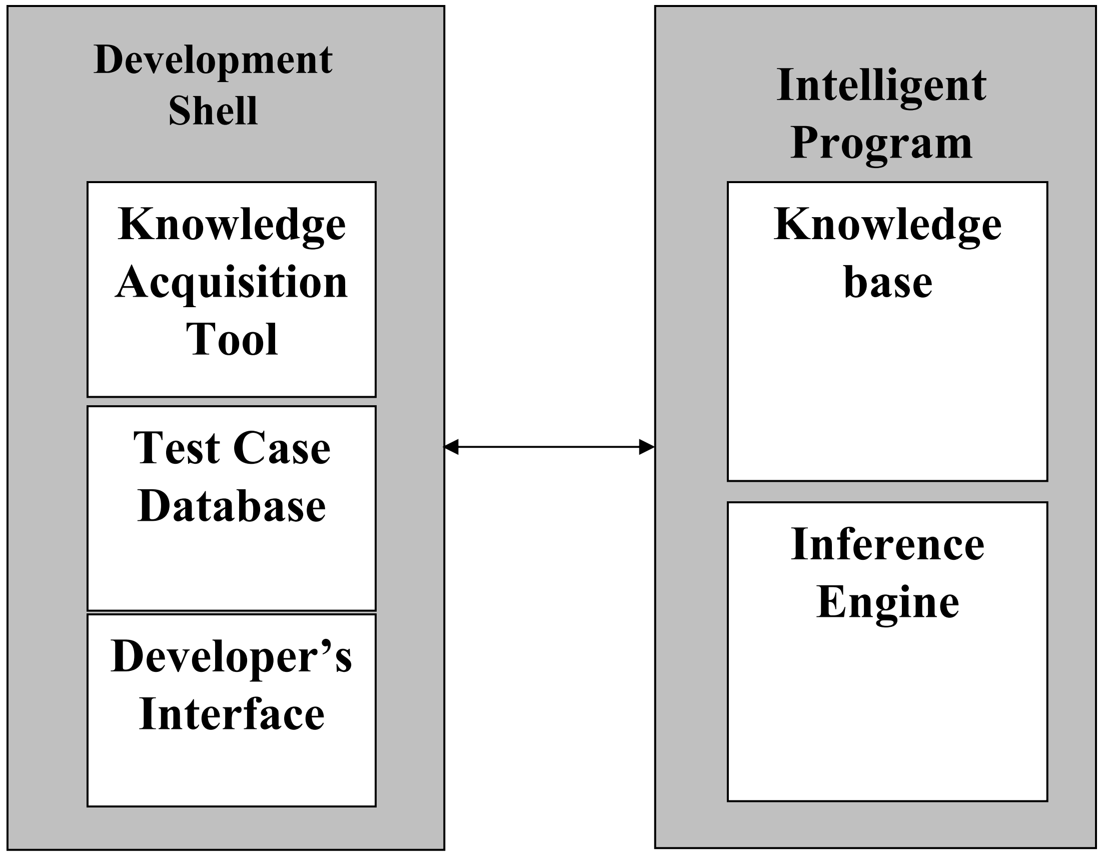
Knowledge-Based System: Developer’s View

- In fact, two kinds of “developers”:
  - Developer of the core platforms or languages
    - develop “shells”, inference engines, UIs, etc.
    - not usually necessary; just use one or more of the many existing development environments
  - Knowledge engineer (KE)
    - use the development environment (e.g., rule language, object-oriented language, etc.) to “program” a KB system
    - traditionally, personally interview domain experts to elicit knowledge
    - modern automated tools can assist, and supplement face-to-face interviews
- Unless we say otherwise, when we say “developer” we mean “knowledge engineer”

Knowledge-Based System: Developer's View

- From developer's perspective, KB system has two main components:
  - Intelligent program
  - Development environment

Knowledge-Based System: Developer's View



Knowledge-Based System: Developer's View

- Intelligent program
  - Unlike the end user, the KE can open the black box.
  - Recall from previous lecture these key differences of KB systems vs. conventional software:
    - The use of highly specific domain knowledge.
    - The heuristic nature of the knowledge employed, instead of exact.
    - <u>The separation of the knowledge from how it is used.</u>
  - Inside the black box are:
    - <u>Knowledge base</u> (the knowledge; declarative)
    - <u>Inference engine</u> (how it is used; procedural)

Knowledge-Based System: Developer's View

- Intelligent program: the knowledge base (KB)
  - Arguably the most important component of a KB system
  - Contains the entire relevant, domain-specific, problem-solving knowledge gathered and integrated by KEs from the various available sources
  - I.e.: contains the knowledge to be managed!

Knowledge-Based System: Developer's View

- How to represent the knowledge in a KB depends on its nature:
  - Functional
    - in the realm of traditional mathematical functions
    - addressed by conventional programming techniques
  - Heuristic
    - most KB applications occur in domains where conventional computational problem-solving approaches either don't exist or don't work well
    - thus the knowledge consists of heuristics – rules of thumb and shortcuts learned and developed by experts over years of practical problem solving experience
    - often naturally expressed as rules (IF-THEN statements)
      - e.g., if the house plan includes a swimming pool, then add $100K to price
  - Structured
    - where knowledge expresses the association of several components (parts) that, when composed, form a larger assembly (whole)
    - often naturally expressed as frames (objects)

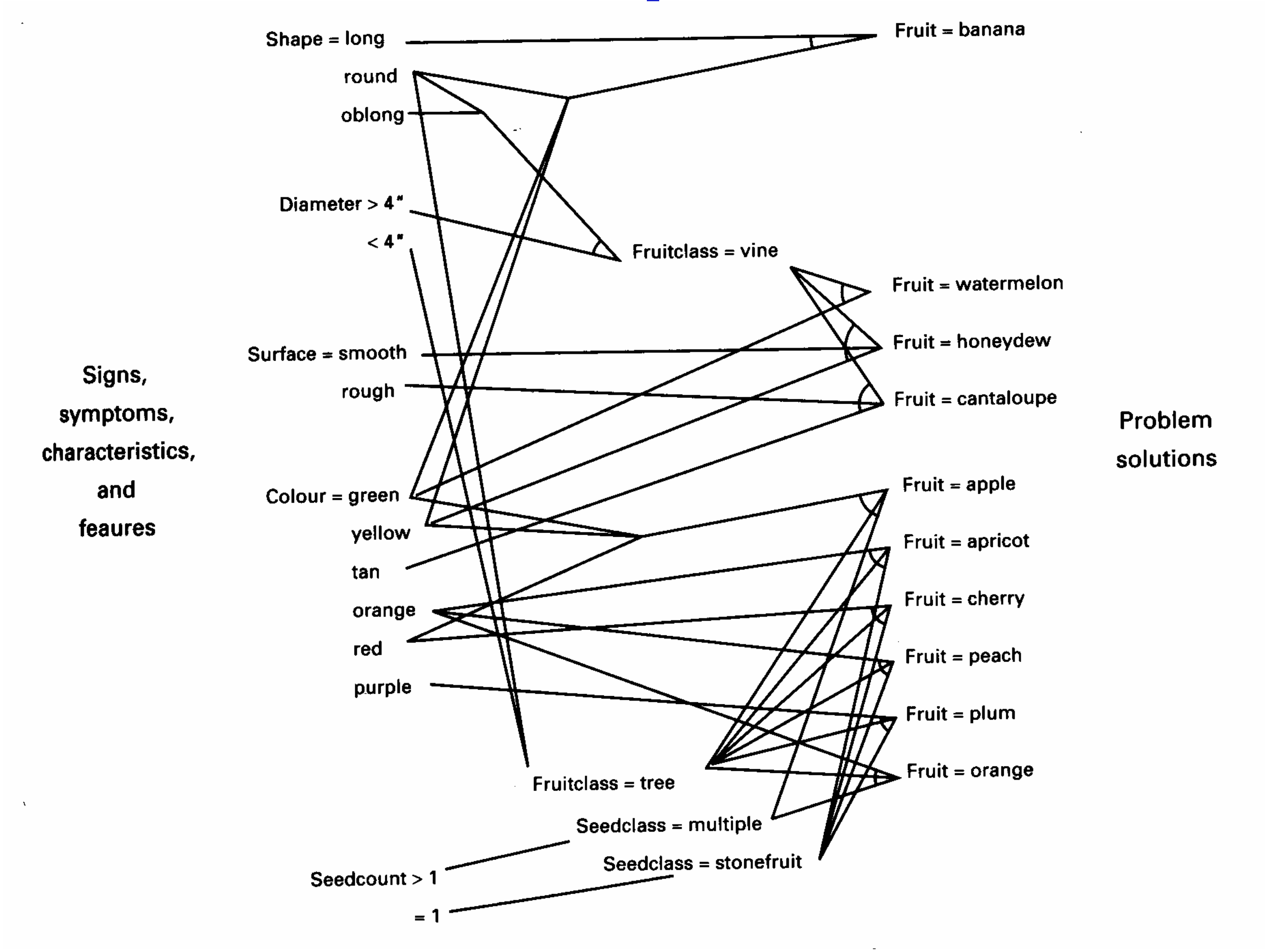
Knowledge-Based System: Developer's View

- Intelligent program: the inference engine
  - The *interpreter* of the knowledge stored in the KB
  - Examines the contents of the knowledge base and the input data accumulated about the current problem
  - Exercises the knowledge to derive the conclusions or answer the user's question

Knowledge-Based System: Developer's View

- Intelligent program: the inference engine
- The knowledge base can be thought of as a huge graph (see next slide w/example of a heuristic system to classify fruits)
  - Nodes on left side: represent all the signs, symptoms, characteristics, and features used as inputs when attempting to solve a relevant problem
  - Nodes on right side: represent all possible solutions to the problem
  - Nodes in middle: represent intermediate conclusions, which the system derives in the progression of deriving the final solution(s)
  - Edges: represent the knowledge used by the expert during problem solving (elicited from experts and stored in the KB)
    - node on its left end: an <u>antecedent</u> of an IF-THEN rule
    - node on its right end: the <u>consequent</u> of the same IF-THEN rule
  - Short arcs linking some edges: represent a logical AND of the antecedents on the edges' left ends, in the IF-THEN rule

Knowledge-Based System: Developer's View



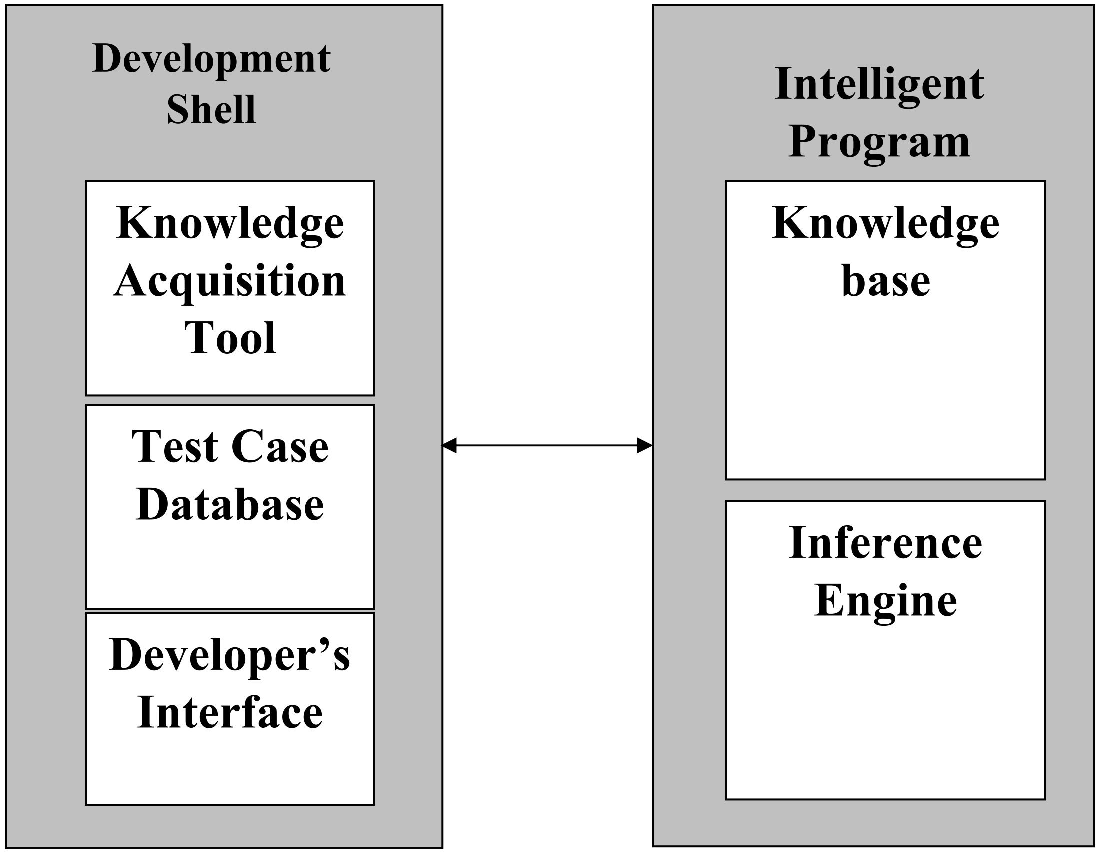
Knowledge-Based System: Developer's View

- The job of the inference engine can be thought of as:
    - exploring this huge graph (which typically can be exponentially large)
    - finding path(s) connecting the nodes on the right to the left
- Note: this graph is just a conceptual aid
    - Logic representations are used within the actual system implementation

Knowledge-Based System: Developer's View

- Development environment
  - Like any IDE, assists the developer with structuring, debugging, modifying, and expanding the "program", which in this case is knowledge gathered from experts.
  - Inside are:
    - Knowledge acquisition tool
    - Test case database
    - Developer's interface

Knowledge-Based System: Developer's View



Knowledge-Based System: Developer's View

- Development environment: Knowledge acquisition tool
    - Assists the KE in the construction of the KB
    - Simplest form:
        - A KB editor, providing a view of the knowledge (rules, frames, etc) and offering editing functions
    - Sophisticated form can add a wide range of features:
        - Multiple browsers for the KB, in both text and graphical forms, hierarchical as well as linear
        - Debuggers and checkers to assist KE in locating "bugs" (logical inconsistencies, typos, etc)
        - Tools that compare existing knowledge to new knowledge, to attempt to guess what the KE really means in case the KE is not as precise as required (analogy: word completion)
        - Version control mechanisms to provide fine-grained bookkeeping
        - …

Knowledge-Based System: Developer's View

- Development environment: Test case database
  - The KE often makes potentially significant (and dangerous!) changes to the KB.
    - Deleting existing knowledge could accidentally eliminate important relationships, preventing needed inferences
    - Modifying existing knowledge could also accidentally change important relationships
    - Adding new knowledge could accidentally introduce contradictions with existing knowledge, or even reprioritize inferences such that the desired conclusions are no longer drawn
  - Any of these could compromise the accuracy/integrity of the knowledge base – checks are essential.
  - Many KB systems include a test case database
    - Sample problems that have been successfully executed on the system in the past
    - Whenever the KB is edited, the test cases are all re-executed to verify that these benchmark cases are still solved correctly.

Knowledge-Based System: Developer's View

- Development environment: Developer's interface
  - Usually, an augmented version of the UI seen by the end user
  - Allows the KE to exercise the KB as it is modified and retested
  - Permits the KE to see exactly how the system can operate when delivered to the end user

Section 8.3 Objectives

- Introduce the various means of representing knowledge:
    - Rules
    - Frames

Knowledge Representation: Rules

- Heuristic knowledge is often nicely represented as rules.
- IF-THEN rules are highly intuitive for both KEs and experts to think about, and to understand once implemented.
- If you ask skilled experts about how they solve problems or how they reached a certain conclusion, they typically respond with knowledge expressed in a rule format
    - e.g., Well, I noticed that A, B, and C were present in this problem, and these three facts imply that D is true.
    - i.e., **IF** A, B, C **THEN** D
    - i.e., $A \wedge B \wedge C \Rightarrow D$
    - i.e.,

$$\begin{tikzcd}
A \arrow[rd] & \\
B \arrow[r] & D \\
C \arrow[ru] &
\end{tikzcd}$$

- A, B, C are the *antecedents* (or *premises* or *conditions/situations*)
- D is the *consequent* (or *conclusion* or *action*)
- Note: a big drawback of rules is that KEs have trouble maintaining consistency in a rule base when the number of interacting rules becomes too large (say, in the thousands)

Rules can be used to express a wide range of associations

- Can express situations and actions that must be taken in those situations (response plans)
    - If dark clouds are rolling in from the west, the wind is increasing, and lightning strikes are occurring, then you should seek cover in a building
    - If you are driving a car and an emergency vehicle approaches, then you should slow down and pull to the side of the road to allow the emergency vehicle to pass
    - If baking a cake, test for completion by inserting a toothpick in the cake's center. If it emerges clean, the cake is ready to be taken out of the oven.

Rules can be used to express a wide range of associations

- Can express premises and conclusions that can be drawn from those premises (diagnosis/explanation)
  - If your body temperature is above 37°C, then you have a fever.
  - If the outside temperature is below freezing, the gas gauge on your car does not register empty, and the engine turns over but will not start, then it is highly likely that you have a frozen gas line.
  - If the loan applicant's salary is greater than $50,000, and the number of outstanding loans is less than two, then the applicant will pay the loan and is considered "good risk."

Rules can be used to express a wide range of associations

- Can express antecedents and their consequences (cause-and-effect)
  - If you don’t read the textbook or attend class, then you will flunk the exam.
  - If the tub’s drain is clogged and the water is left running, then the floor will become wet.
  - If the electric bill is past due and your credit rating is bad, then your electricity will be cut off.

Inference chains

- With pattern matching, rule-based systems use automated reasoning methods to progress logically from data to conclusions.
- Process of problem-solving in KB systems is to create a series of inferences that form a “path” between the problem definition and its solution.
- Such a series of inferences is called an inference chain.

Inference chains (Example)

Rule base

- Rule 1
  - IF the ambient temperature is above 90°F
  - THEN the weather is hot
- Rule 2
  - IF the relative humidity is greater than 65%
  - THEN the atmosphere is humid
- Rule 3
  - IF the weather is hot and the atmosphere is humid
  - THEN thunderstorms are likely to develop

Inference chains (Example)

Inputs

- Fact 1
  - the ambient temperature is 92°F
- Fact 2
  - the relative humidity is 70%

Inference chains (Example)

From:

- Fact 1
  - the ambient temperature is 92°F
- Rule 1
  - IF the ambient temperature is above 90°F
  - THEN the weather is hot

The inference engine can deduce:

- Fact 3
  - the weather is hot

Inference chains (Example)

From:

- Fact 1
  - the relative humidity is 70%
- Rule 2
  - IF the relative humidity is greater than 65%
  - THEN the atmosphere is humid

The inference engine can deduce:

- Fact 4
  - the atmosphere is humid

Inference chains (Example)

From:

- Fact 3
  - the weather is hot
- Fact 4
  - the atmosphere is humid
- Rule 3
  - IF the weather is hot and the atmosphere is humid
  - THEN thunderstorms are likely to develop

The inference engine can deduce:

- Fact 5
  - thunderstorms are likely to develop

Uncertainty

- Note the use of “likely” in “thunderstorms are likely to develop”
- Ideally we’d like a more precise prediction than “likely”
- Modern systems provide mechanisms for reasoning under uncertainty
  - Probability (especially Bayesian methods)
  - Fuzzy logic

Knowledge Representation: Frames

- Rules do not provide a natural representation for:
  - grouping facts into associated clusters
  - associating relevant procedural knowledge with some fact or group of facts
- Frames were developed by Marvin Minsky [1975] for this purpose, in his landmark paper “A Framework for Representing Knowledge”
- Frames formed the basis of objects and object-oriented programming (OOP)
  - In fact, OOP began as a branch of AI in the 1980s (with Smalltalk and Lisp’s CLOS)
- Frames naturally represent structured knowledge
  - Just as rules naturally represent heuristic knowledge
- Also quite intuitive, but in a different way than rules
  - Not quite as natural as rules for the lay person, but typically quite easy for the technically educated

Frame

- A frame provides the structure or framework for representing structured knowledge
- Using a structure consisting of a frame <u>name</u> and a set of <u>attribute-value</u> pairs, a frame represents a stereotypical situation (i.e., an object, concept, or process) from the real world

Figure 8.4

Frame: **MUSTANG**
Manufacturer: Ford
Country of Manufacture: USA
Model: Mustang GT
Number of wheels: 4
Number of doors: 2
Year: 2002
Engine Size: 4.6L-V8
Transmission: Standard
Reliability: Medium
Body Style: Convertible
Color: Red
Miles-per-gallon: 19.7
Serial Number: 12345A67890B
Owner: Avelino

Functional attributes (Methods)

- Functional attributes are attributes whose values are executable functions
  - Also known as methods ("member functions" in C++)
  - E.g., if we wish to know the reliability of 2002 Mustangs based on the fleets' experience, this number could change constantly. A method could provide more reliable up-to-date numbers than storing a fixed value.

Default values

- A default value can be associated with each attribute.
  - (In C++ or Java, this is like providing default values in the constructors.)

Attributes, in more detail

- Attributes are also known as slots ("members" in C++)
- Values are also known as fillers
- Each attribute/slot/member can be filled by a value/filler
- In addition, certain metaproperties can be associated with any attribute/slot/member (these were originally called "facets" and the filler was also considered one of the facets):
  - Range: the range of possible values for this slot
  - Legal-values: the set of discrete possible values the slot can take (like enums in C++)
  - Default: the value to assume if none is explicitly stated
  - If-needed: Procedure(s) for determining the actual value
  - If-added: Procedures to execute when a value is specified for the slot
  - If-changed: Procedures to execute if the value of the slot is changed
- The last three, which evolved into methods, are also called daemons

Inheritance

- To avoid including, for example, the number of wheels in every frame for every car, inheritance was invented.
    - All attributes that do not change (the general attributes) are moved up to a higher level frame.
    - The more specific frames inherit that information.

Figure 8.5

Generic **AUTOMOBILE** Frame
    Specialization–of: VEHICLE
    Generalization–of: (STATION–WAGON COUPE SEDAN CONVERTIBLE)

```
Manufacturer:
          Value:
          Legal-values: (FORD MAZDA BMW SAAB GM CHRYSLER)
          Default: FORD
Country–of–Manufacture:
          Value:
          Legal-values: (USA JAPAN GERMANY SWEDEN)
          If-Needed: (GET-ORIGIN)
          Default: USA
Model:
          Value:
          Legal-values: (TAURUS FOCUS MUSTANG CROWN-VICTORIA))
Color:
          Value:
          Legal-values: (BLACK WHITE RED PERSIAN–AQUA)
          If–Needed: (EXAMINE–TITLE or CONSULT–DEALER or LOOK–
                  AT–AUTOMOBILE)
Reliability:
          Value:
          Legal-values: (HIGH MEDIUM LOW)
          Default: MEDIUM

Miles–Per–Gallon:
          Value:
          Range: (0 – 100)
Year:
          Value:
          Range: (1940 – 1990)
          If–Changed: (ERROR: Value cannot be modified)
Owner:
          Value:
          If–Added: (APPLY–FOR–TITLE and OBTAIN–TAG and PAY–
                  SALES–TAX)
```

Figure 8.6

- By exploiting inheritance, notice how simple it can become to define new frames.

Generic COUPE Frame
    Specialization–of: AUTOMOBILE
    Generalization–of: (SUZIE-SMITH'S-AUTOMOBILE, JOHN-DOE'S-AUTOMOBILE AVELINO'S-AUTOMOBILE)
    Doors:
        Value: 2

Figure 8.7

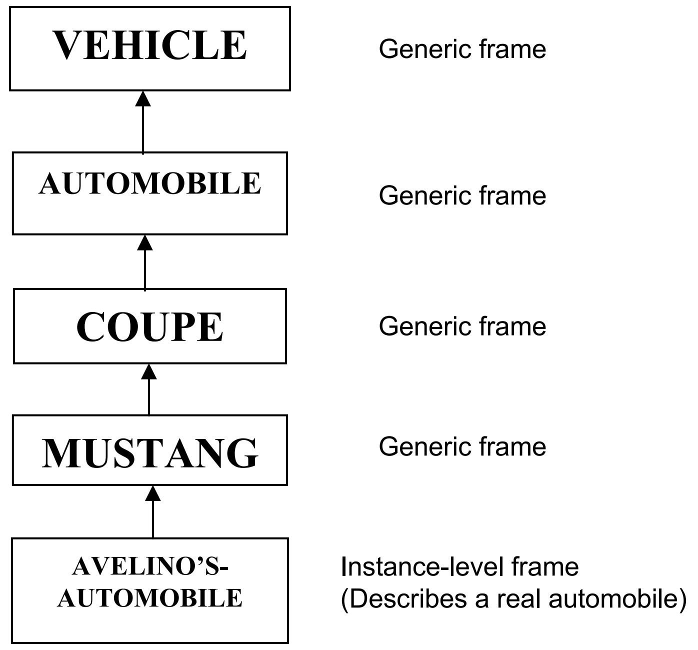

VEHICLE
Generic frame
AUTOMOBILE
Generic frame
COUPE
Generic frame
MUSTANG
Generic frame
AVELINO'S-AUTOMOBILE
Instance-level frame
(Describes a real automobile)

Figure 8.8

Generic **MUSTANG** Frame
  Specialization-of: Coupe
  Generalization-of: AVELINO'S-AUTOMOBILE
Manufacturer:
    Value: FORD
Engine:
    VALUE:
    LEGAL-VALUES: (3.0L-V6, 4.6L-V8)

Figure 8.9

AVELINO'S-AUTOMOBILE Frame
Specialization–of: Mustang
Manufacturer:
    Value: Ford
Vehicle ID No.:
    Value: 12345AJG67890
Country–Of–Manufacture:
    Value:
Color:
    Value: Red
Reliability:
    Value: Medium
Miles–Per–Gallon:
    Value: 19.7
Year:
    Value: 2002
Owner:
    Value: AVELINO
Doors:
    Value:

Section 8.4 Objectives

- Introduce the means of manipulating the knowledge found in a knowledge base.
- Reasoning with frames
- Reasoning with rules
  - Forward reasoning
  - Backward reasoning

Frame-Based Reasoning

- Modern knowledge representations differentiate between terminological and assertional knowledge [KL-ONE]
- Terminological
  - E.g.: "a red car belonging to Bob"
    - is just a term that we can use
    - does not say anything about the real world
    - in fact, there might not even exist any red car belonging to Bob in the real world
  - Reasoning: respond to queries about the definitions of various concepts (classes/categories, objects/instances)
- Assertional
  - E.g.:
    - "the car belonging to Bob is red"
    - "the red car belongs to Bob"
    - "a red car belonging to Bob does not exist"
  - Reasoning: respond to queries about the known state of the world

Frame-Based Reasoning

- Frame-based reasoning
  - Excels at defining terms (i.e., for representing terminological knowledge)
  - Is generally used as support for defining terms or storing concepts, within more complex systems that use other mechanisms for reasoning about assertional knowledge
  - Also excellent for representing cases in case-based reasoning systems (next lecture)

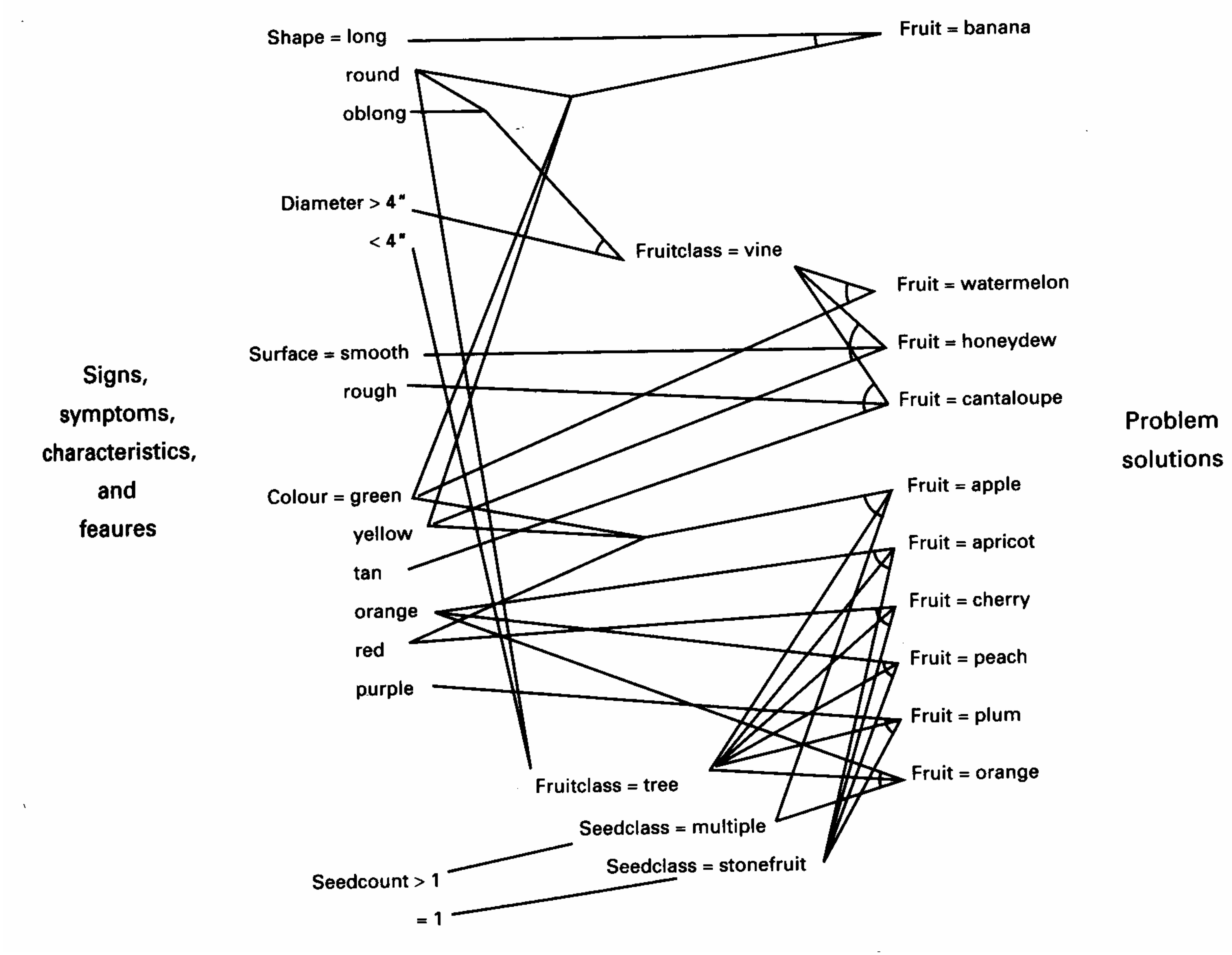
Rule-Based Reasoning

- Abstractly: the objective of a reasoning process is to derive a value (and these days, also a measure of certainty) for a conclusion.
- But there is a wide variance of ways this can be done.
  - In many situations, human find it most natural to progress from the initial data to a final answer.
    - E.g., thunderstorm example earlier.
    - Makes good sense whenever
      - relatively few input data are required, or
      - there are many possible conclusions.
  - Alternatively, when there is much data but only a small portion is relevant, considering all the data would be highly inefficient.
    - E.g., you only tell the doctor your abnormal symptoms (headache, fever) and not all the other things that are okay (neck doesn’t hurt, back feels fine, …).
    - Doctor tries to determine the most likely diagnosis based on the limited input data, and then prove the hypothesis via additional questions or tests.

Rule-Based Reasoning

- So there are two means of deriving conclusions:
  - Start with all the known data and progress toward the conclusion – data driven, forward chaining, or forward reasoning
  - Select a possible conclusion and try to prove its validity by looking for supporting evidence – goal driven, backward chaining, or backward reasoning

Rule-Based Reasoning (recall this earlier example…)



Rule-Based Reasoning

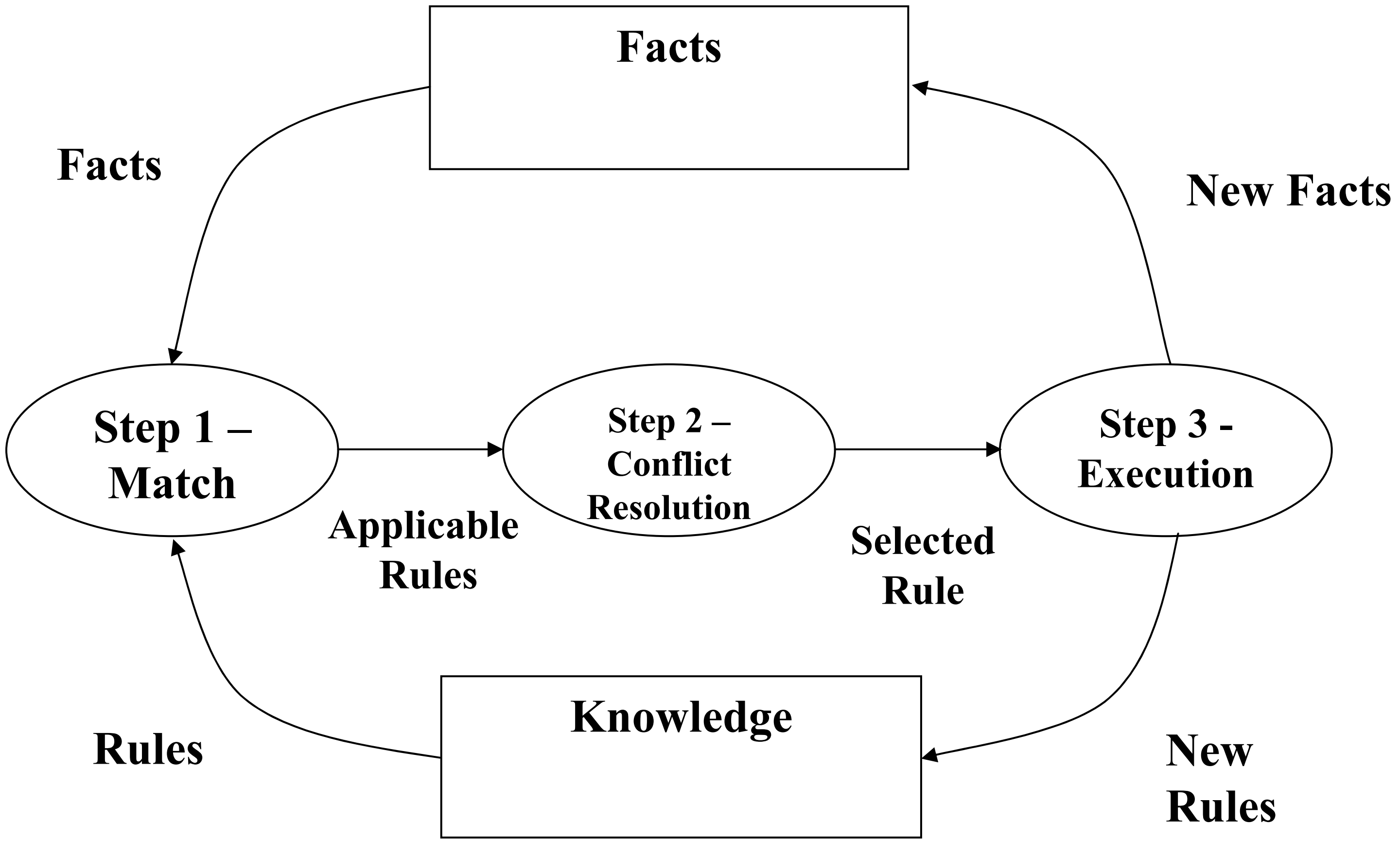
- Rule 1
  - IF Surface = smooth
    AND Fruitclass = vine
    AND Color = green
  - THEN Fruit = honeydew

- Each rule describes some characteristic of the different fruits through a series of parameters, eg:
  - Fruit
  - Fruitclass
  - Seedclass
- Parameters that represent the final answer are <u>conclusions</u> or <u>goals</u>.
  - Fruit
- The others are called <u>intermediate</u> parameters/conclusions/goals.
- A combination of parameter and value that is considered true is called a <u>fact</u>.
- The workspace (or database) contains all the facts known to the system at any given point in time during the execution of the KB system.
- By assigning the goal a value (eg, Fruit = apple), the system solves the problem of classifying a specific instance of a fruit.

Forward chaining

- Moves from the input data toward the conclusions/goals.
- Inputs are used to satisfy the premises of applicable rules, which allows them to be executed ("fired") thus setting values of other (intermediate or final) parameters, yielding newly derived facts.
- These new facts intern may cause the premises of other rules to be satisfied and fired.
- This process is called <u>rule interpretation</u>.

Forward chaining: Rule Interpretation Process



Trace of Rule-Based Execution

Execution Cycle	Applicable rules	Selected rule	Derived Fact
1	3,4	3	Fruitclass = tree
2	3,4	4	Seedclass = stonefruit
3	3,4, 11	11	Fruit = cherry
4	3, 4, 11	—	—

Forward chaining

- Best suited for problem domains involving synthesis:
  - design
  - configuration
  - planning
  - control
  - scheduling
- Domains where the data drive the solution approach
- Very well known and widely used free public-domain implementation is CLIPS.
  - Originally developed by NASA
  - Also supports limited backward chaining

Backward chaining

- Moves from a hypothesized goal backward toward the input data, looking for some way to assemble enough supporting evidence from the input data.
- E.g., start with a single top-level goal in our fruit classification example – the goal is indicated by the goal parameter <u>Fruit</u>
- Backward inference can also already begin without any inputs specified.
    - Inputs can be requested from the user or otherwise acquired, only as they become necessary to derive a value for the selected goal.
- When an answer is found, execution can stop, or it can go on to attempt to satisfy another goal.
- This process is called <u>tracing a goal</u>.
- The rule interpreter for backward reasoning differs significantly from that of forward reasoning.
    - Typically, the workspace is initially empty.

Backward chaining: Rule Interpretation Process

1. Form a list initially composed of all top-level goals defined in the system. The top-level goals are predefined by the developer.
2. Consider the first goal from the list. Gather all rules capable of deriving a value for this goal.
3. For each of these rules, in turn examine its premises:
    1. If all its premises are satisfied (ie, each premise has its specified value contained as a fact in the workspace), then execute this rule to derive its conclusions. Remove this goal from the list and return to step 2. This is done because a value has been derived for the current goal.
    2. If a premise of a rule is not satisfied because of the absence of a fact to satisfy one of its premises (i.e., one of the premise's values does not exist as a fact in the workspace), look for rules that derive the specified value for this premise. If any can be found, then consider this premise to be a subgoal, place it on the beginning of the goal list, and go back to step 2.
    3. If step b cannot find a rule to derive the specified value for the current premise, then query the user for its value and add it to the workspace as a fact. If this value satisfies the current premise, then continue with this rule's next premise. If the premise is not satisfied, then consider the next rule.
4. If all rules that can satisfy the current goal have been attempted and all have failed to derive a value, then this goal remains undetermined. Remove it from the list and go back to step 2. If the list is empty (ie, all top-level goals have been processed), then halt and announce completion.

Backward chaining: Closed World Assumption

- Rule-based systems generally follow the <u>closed-world assumption</u>:
  - If a fact specifically asserting something is not present, then assume it to be false.
- Thus backward chaining along a particular path will fail, if it requires some fact that cannot be proved from any input facts.

Backward chaining

- Goals: (Fruit)
- **System:** What is the value for Shape?
- **User:** round
- Workspace: ((Shape = round))
- Goals: (Fruitclass Fruit)
- **System:** What is the value of Diameter?
- **User:** 1 in
- Workspace: ((Shape = round) (Diameter = 1 in))
- Rule 3 premises are satisfied
- Workspace: ((Shape = round) (Diameter = 1 in) (Fruitclass = tree))
- Goals: (Fruit)
- **System:** What is the value of Color?
- **User:** red
- Workspace: ((Shape = round) (Diameter = 1 in) (Fruitclass = tree) (Color = red))
- **System:** What is the value of Seedclass?
- **User:** stonefruit
- Workspace: ((Shape = round) (Diameter = 1 in) (Fruitclass = tree) (Color = red) (Seedclass = stonefruit))
- Rule 11 premises are satisfied
- Workspace: ((Shape = round) (Diameter = 1 in) (Fruitclass = tree) (Color = red) (Seedclass = stonefruit) (Fruit = cherry))
- Goals: ()
- Halt

Backward chaining

- Best suited for <u>diagnostic</u> problems that have a relatively small number of possible conclusions
- Best known implementation is <u>Prolog</u>
  - Of course, since Prolog is a complete programming language, you can always implement a forward chainer using Prolog

Issues in Developing Knowledge-Based Systems

- Requirements are harder to define than for standard software
- Human expertise is difficult to define and even more difficult to elicit
  - Unlike standard software, requires developer to maintain close continuous contact with experts throughout the entire development process
- Desirable qualities of knowledge engineers can be more challenging than for standard software engineers
  - Must depend equally on intuition and personal qualities to be successful, eg
    - ability to absorb broad and deep knowledge in non-CS domains
    - the ability to get along well with others
  - Must still have good "programming" (logic, design, organization, documentation, debugging, and testing) skills just like other SEs

Knowledge Engineering

- Gonzalez & Dankel [1993] definition:
    - The acquisition of knowledge in some domain from one or more non-electronic sources, and its conversion into a form that can be utilized by a computer to solve problems that, typically, can only be solved by persons extensively knowledgeable in that domain.
- KM standpoint:
    - The elicitation, capture, and storage of tacit or explicit knowledge for later application.

Knowledge Engineering

- Involves more than merely translating the knowledge from human terms to a machine-readable form.
- Involves:
  - recognizing what knowledge is in use to solve a problem
  - categorizing this knowledge
  - determining the best way to represent it
- Improperly represented knowledge may ultimately doom a KB system development project!
  - Problem: the impact of a poor representation may not be immediately felt.
  - Significant effort may be wasted creating a system that, ultimately, must be completely redeveloped because of poor initial knowledge representation choices.

Tools

- Tools available for developing knowledge-based systems:
  - Rule-based shells
  - Hybrid shells
  - Special purpose shells
  - Inductive shells (considered later in this course)
  - Developing a system from scratch
- Excellent open-source as well as expensive commercial systems available

Conclusions

- The student should be familiar with:
  - Rules and how they are used to represent and exercise conditional knowledge.
  - Frames and how they are used and exercise structured knowledge.
  - The meaning of a knowledge-based system shell and the types of shells commercially available.

Chapter 8

Preserving and Applying Human Expertise: Knowledge-Based Systems