Master of Technology in Knowledge Engineering

KE4102: Intelligent Systems & Techniques for Business Analytics: Rule-Based Systems

Charles Pang
Institute of Systems Science
National University of Singapore
E-mail: charlespang@nus.edu.sg

Lecture Contents

- Introduction to Rule-based Systems
  - Forward Chaining
  - Backward Chaining
- RBS development method
  - Knowledge Acquisition
  - knowledge modeling
  - Task Modeling
  - Design Modeling

Rule-based Systems

- The most commonly used technique in AI Systems
- In the early years, RBS were known as “Expert Systems”
- Earliest rule-based systems: MYCIN, DENDRAL, R1/XCON, PUFF
- Uses “rules” as a knowledge representation to store problem-solving knowledge
- Inference strategy is known as “chaining”

Rule-based System Example

- Pet selection system would have some rules like this:
  - Rule1: IF size=small THEN pet=rabbit
  - Rule2: IF size=mid THEN pet=cat
  - Rule3: IF size=large THEN pet=dog
  - Rule4: IF working=long_hours THEN size=small or mid
  - Rule5: IF care=easy THEN pet=rabbit or cat
  - …..

- Structure:
  - IF *premise* THEN *conclusion*

Rule-based Reasoning

- Rule-based reasoning is a model of human problem solving:
- Deductive reasoning
  - **IF** fur_allergy= true **THEN** pets= no
  - **IF** working= long_hours **THEN** size=small or mid
  - ✓ *Analytic form of reasoning where the conclusion is guaranteed if the premise is true*
- Abductive reasoning
  - **IF** pets= no **THEN** fur_allergy= true
  - **IF** person_enter= male_toilet **THEN** gender= male
  - ✓ *Synthetic form of reasoning which may not be true*

Rules

- Rules in an RBS are relatively independent
  - Each rule represents a single chunk of knowledge
  - ***IF pet_size=medium THEN recommend=cats or small dogs***
- Rules are based on a priori knowledge or heuristics
  - Rules are derived from domain experts who uses experiential knowledge and "rules-of-thumb" more than textbooks
  - ***IF personality=emotional THEN recommend=hamster***
- Rules can incorporate uncertainties
  - Real life business situations are plagued with uncertainties that make decision-making difficult
  - ***IF work=long_hours THEN recommend=dog (30%)***

Rule-Based System Overview

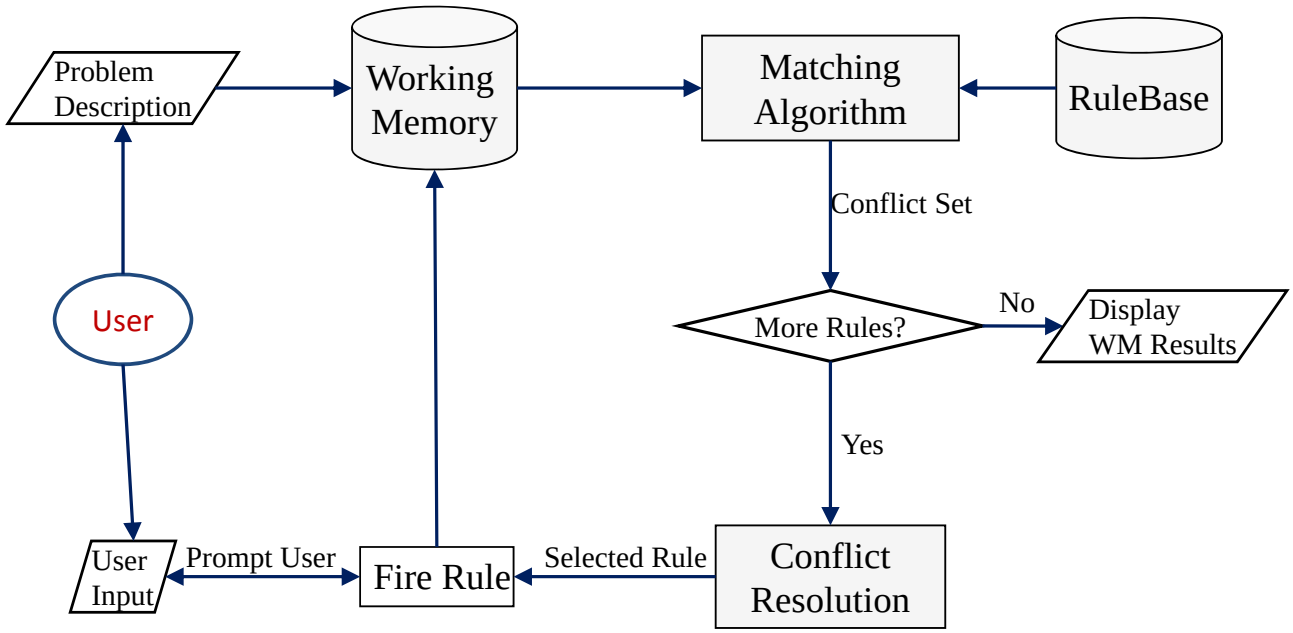
A RBS comprises 3 components:

- A set of production rules (Rule Base)
  - A rule represents a single chunk of problem-solving knowledge
- A working memory (WM)
  - Contains Rules and Data (current state of program execution)
- A Rule engine
  - A computational system that implements the control strategy and applies (fire) the rules
  - The patterns in WM are matched against the conditions of the rules
  - Matched rules are called the conflict set
  - The control strategy determines the order in which the rules are fired and resolves any rule conflicts
  - Uses the Recognise-Act cycle

Recognize-Act cycle

1. Match the condition in the RULES with the FACTS in the WM to identify the set of applicable rules
2. If there is more than one rule that can be applied then use a conflict resolution strategy to choose which one to apply. If no rules are applicable, stop execution
3. Apply the selected rule. This may result in adding new facts to the WM or deleting old facts
4. Check if the terminating goal condition is fulfilled. If it is, stop otherwise return to 1.

Recognise-Act Control Cycle



Forward Chaining Strategy

- A forward chaining system begins with a set of facts (in the WM) and applying rules to generate new facts until the desired goal is reached.
- Rules whose premise (IF..) is known to be true are fired, and their conclusions (THEN..) are declared true.
- This process continues until no more rules can be fired. The system then reports its conclusions.

Forward Chaining Example

Rule 1:

IF	the patient has a sore throat
AND	we suspect a Bacterial infection
THEN	we believe the Illness is a strep throat

Rule 2:

IF	the patient's temperature is > 100
THEN	the patient has a fever

Rule 3:

IF	the patient has been sick for over a month
AND	the patient has a fever
THEN	we suspect a bacterial infection

Facts:	**Temperature = 104º**	**F1**
	Patient has been sick for 2 months	**F2**
	Patient has a sore throat	**F3**

Forward Chaining Example - WM

Initial state of working memory WM:[F1, F2, F3]

Rule-2 [F1]

Add F4: "patient has a fever"

Rule-3 [F2, F4]

Add F5: "suspect a bacterial infection"

Rule-1 [F3, F5]

Add F6: Illness is a strep throat

No more rules to fire – halt.

Conclusion – illness is a strep throat

Forward Chaining Algorithm

Let WM (*Working Memory*) be the list of all known facts.

REPEAT

Obtain R, the list of all rules which could be applied based on the facts in WM (*Match*)

Select S, a rule from R.

Execute (fire) S - this means carrying out actions specified in the conclusion of S; usually adding facts to, or deleting them from WM

Remove S from R

UNTIL R = [ ]

Problem-solving using Forward Chaining

- Works well when the problem naturally begins by gathering information and then seeing what can be inferred from it, e.g. If I have chicken, tomato paste, paprika then I should cook *chicken cacciatori*
- Good approach for certain types of problem-solving, such as
  - Planning: eg. check the fridge for ingredients, and suggest a recipe
  - monitoring & control: eg. If engine is heating up, turn up cooling fan
  - Advisory: eg. If you have good credit, good bank balance then grant you loan

Backward Chaining

- Forward Chaining is used to draw new conclusions from existing data.
- In some problems the conclusion is known to you (e.g. “engine cannot start”) and your task is to find the cause of it. These are diagnostic-type problems, and you are tracing backwards to find the root cause.

Backward Chaining Strategy

- A backward chaining inference engine starts with a goal, or hypothesis (in the WM)
- It works through the rules trying to match the goal with the action clauses (THEN part) of a rule.
- When a match is found, the condition clauses (IF part) of the matching rule become "sub-goals".
- The cycle is repeated until a verifiable set of condition clauses is found- no more rules can be are applied

Backward Chaining Rules

- In principle we can use the same set of rules for both forward and backward chaining. However, in practice we may choose to write the rules slightly differently if we are going to be using them for backward chaining.
- In backward chaining we are concerned with matching the conclusion of a rule against some goal that we are trying to prove. So the 'then' part of the rule is usually not expressed as an action to take (e.g., add/delete), but as a state which will be true if the premises are true.

Backward Chaining Example

Rule 1:
IF the patient has a sore throat
AND we suspect a Bacterial infection
THEN we believe the illness is a strep throat

Rule 2:
IF the patient's temperature is > 100
THEN the patient has a fever

Rule 3:
IF the patient has been sick for over a month
AND the patient has a fever
THEN we suspect a bacterial infection

Temperature = 104°
Patient has been sick for 2 months
Patient has a sore throat
Is the illness a strep throat?

Backward Chaining Example

- Initial state of working memory WM:[Hypothesis]

Rule-1 [Hypothesis]

Add F4: patient has a sore throat? *{goal already proven}*

Add F5: we suspect bacterial infection? *{new goal to pursue}*

Rule-3 [F5]

Add F6: patient has been sick for over a week? *{goal already proven}*

Add F7: patient has a fever? *{new goal to pursue}*

Rule-2 [F7]

Add F8: patient's temperature is > 37.4? *{goal already proven}*

No more rules to fire – halt.

Conclusion – yes, illness is a strep throat

Problem-solving using Backward Chaining

- Works well when the problem naturally begins by forming a hypothesis and then seeing if it can be proven
  - "I believe that the patient has a strep throat."
- Focused on a given goal and searches only that part of the KB that is relevant to the current problem
- Good approach for tasks like trouble-shooting, diagnostics

Conflict Set

- There are situations where more than 1 rule can fire based on the same set of facts. This creates a conflicting set of rules
- A decision has to be made to resolve the conflict and determine which rule should be fired first
- The type of conflict resolution technique adopted will affect the solution, and how quickly it is found

Conflict Set Example

Rule 1: If credit-rating = FAIR

THEN Grant-Loan = MAYBE

Rule 2: If credit-rating = FAIR AND

customer-score = GOOD

THEN Grant-Loan = YES

facts in WM:

Fact-1: credit-rating = FAIR

Fact-2: customer-score=GOOD

Conflict set is: <R1, Fact-1>, <R2, Fact-1, Fact-2>

Conflict Resolution

- In forward chaining, the order in which the rule fires depends on facts in WM, not the order of rules.
- Selecting which rule to fire:
  - Attach priority to rules, and select the rule with the highest priority (Salience)
  - Order facts by the length of time they have been in working memory, and select the most recent. (Recency)
  - Select rules which required the highest number of facts (Specificity).

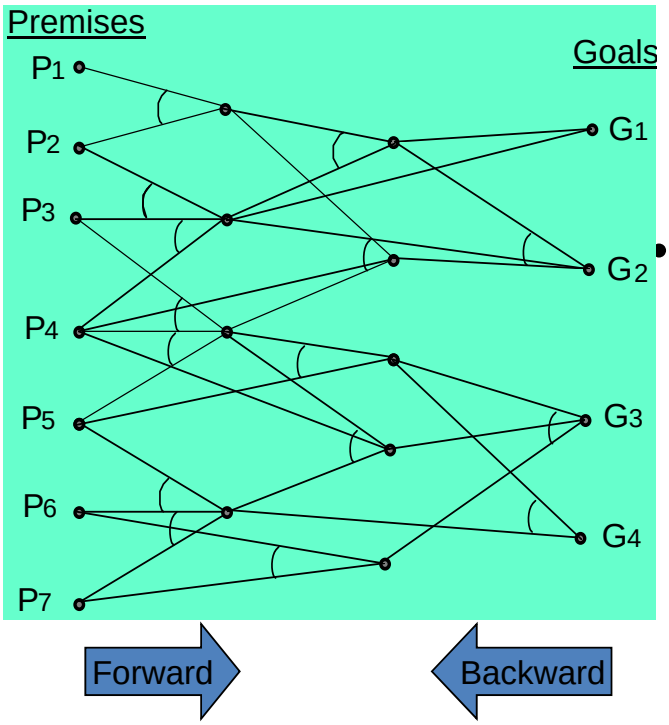
Forward Chaining vs Backward Chaining

Forward Chaining	***Backward Chaining***
planning, monitoring, control	Diagnosis, Trouble-shooting
Present to future Antecedent to consequent	Present to past Consequent to antecedent
Data driven, bottom-up reasoning	Goal-driven, top-down reasoning
Work forward to find what solutions that follow from the facts	Work backwards to find facts that support a given hypothesis
Facilitates Breadth-First-Search	Facilitates Depth-First-Search
Does not facilitate Explanation	Facilitates Explanation
CLIPS	PROLOG

Forward Chaining vs Backward Chaining

- It is certainly possible to do diagnostics in forward chaining system and planning in backward chaining.
- Explanation is facilitated in BWD because the system can easily explain exactly what goal it is trying to satisfy. In FWD, explanation is not so easily facilitated because the subgoals are not explicitly known until discovered.
- In BWD chaining, the system will commonly elicit evidence from the user to aid in proving or disproving hypothesis. This contrast with FWD system in which all the relevant facts are usually known in advance.

Forward Chaining vs Backward Chaining



- FWD works best if the tree is wide and not very deep – Breadth First Search.
- BWD works best if the tree is narrow and deep – Depth First Search.

RBS Development Methodology

- Knowledge Acquisition
- Knowledge Modeling
- Task Modeling
- Design Modeling
- Development
- Verification & Validation

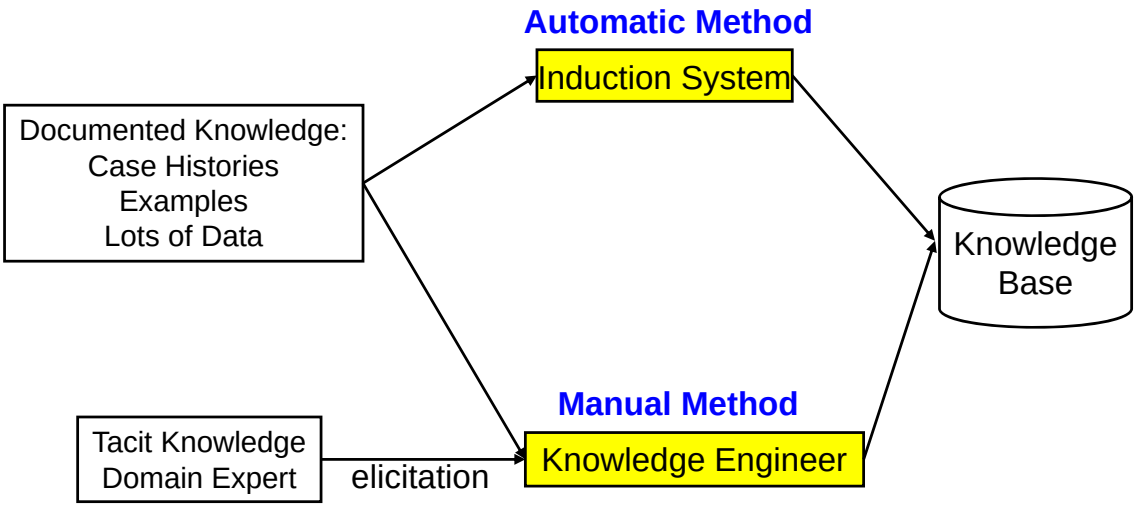
Knowledge Acquisition

- Knowledge Acquisition is the transfer and transformation of problem solving knowledge into a form that can be used to build Intelligent Systems
- Knowledge Acquisition is ***Requirements Engineering***
- Knowledge acquisition is sometimes called *knowledge capture* or *knowledge elicitation*
- Personnel involved : Subject Matter Expert (SME) and Knowledge Engineer (KE)
- knowledge acquisition can be accomplished in many ways

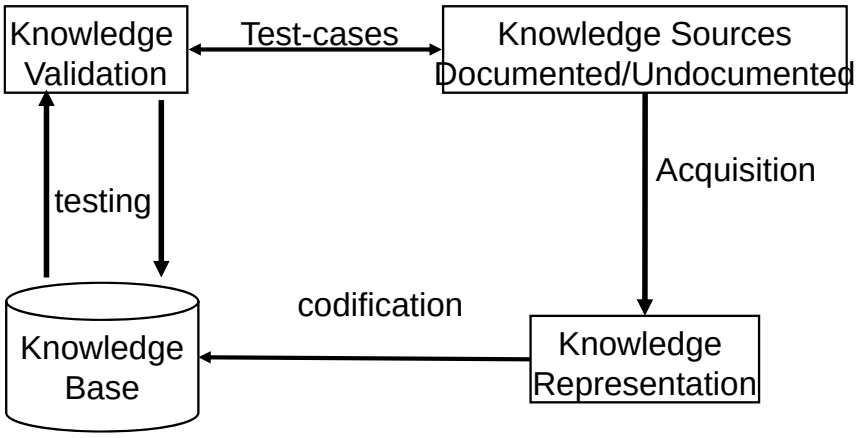
Knowledge Sources for Acquisition

- Two types of knowledge can be used to build RBS:
  - Documented sources
    - Collection from printed sources
    - Machine learning (Decision Tree, Clustering, Classification, NN, etc)
  - Undocumented sources:
    - Tacit knowledge that can only be captured by elicitation from human experts

Knowledge Acquisition Methods



Knowledge Acquisition Cycle



- Knowledge capture requires many refinement cycles

Knowledge Elicitation

- **Elicitation** is acquisition of ***TACIT knowledge*** from a subject matter expert
- The Knowledge elicitation approach:
  - Capture knowledge using interview techniques
  - Interpret/analyse the data
  - Build knowledge models
  - Use the knowledge models to guide further elicitation
  - Verify and Validate the knowledge
  - Stop when the knowledge model is complete

Knowledge Elicitation bottleneck

- Knowledge Elicitation is difficult and often causes a bottleneck in the development cycle of Knowledge-based system. Hence, the term *"Knowledge Elicitation bottleneck"*.
- Bottleneck issues can happen because of :
  - Subject Matter Expert
  - Knowledge Engineer
  - Problem-solving Knowledge

Bottleneck: Domain Expert

- Common problems with human experts:
  - Uncooperative – feels threatened
  - Not interested – no value to them
  - Vulnerability – weakness is exposed
  - Unavailable – too busy
  - Wrong level - too senior
  - Knowledge is intuitive - cannot be codified

Bottleneck: Knowledge Engineer

- Lack of relevant training
- Lack of experience
  - Elicitation techniques are hard to master
- Misconception that questions implies answers
- Misinterpretation of what is heard
- Personal bias - deep rooted principles
- Relationship/Interaction issues with experts

Bottleneck: Problem-solving Knowledge itself

- Problem solving knowledge can be Tacit:
  - Heuristics
  - Experiential
  - Structured
  - Compiled
- Types of knowledge
  - Conceptual versus Procedural
  - Explicit versus Tacit

Interview Technique

- Most widely used technique for capturing tacit knowledge
- An effective technique for verifying and validating captured knowledge
- Allows first-hand observation of expert's problem solving behavior and process
- Most people enjoyed being interviewed ☺
- Structured and Unstructured Interviews

* Different approaches are used in structured and unstructured interviews

Interview Process

- More Beneficial interviewing expert at their workplace
- Make sure the meeting place is quiet and free from interruptions
- Before the interview:
  - Do background research on the domain area
  - Background check on the domain expert
  - Design and phrase your questions
  - Email questions to domain expert
  - Acquire and prepare the tools for the interview
- During the interview:
  - Introductions & Social preliminaries
  - State purpose of interview
  - Give a brief on the roles and responsibilities
  - Be courteous, Listen closely and avoid arguments
  - Investigate each topic in detail
  - Evaluate session outcome
- Observe confidentiality

Knowledge Modeling

- After you have acquired some domain knowledge from the experts and other sources, how do we present a comprehensive view of this knowledge?
- **Knowledge Models**
  - A Knowledge model is a ***structured representation*** of knowledge that allows us to better understand the domain and the processes involved in decision-making
  - Models provide rich descriptions of domain knowledge that is ***independent*** of any particular implementation formalism.
  - Models also serve as a basis for communication between experts, knowledge engineers and end-users

Knowledge Models

- Concept Dictionary
- Concept Tree
- Composition Tree
- Attribute Tress
- Decision Tree
- Cause Tree
- Goal Reduction

Concept Dictionary

- A concept dictionary contains the list of all relevant concepts that are used in the problem domain to solve the problem.
- The dictionary provides a detailed explanation of the concept and can include any information that is useful for a good understanding of the concept.
- It can be similar to a glossary.
- It does not have any particular format

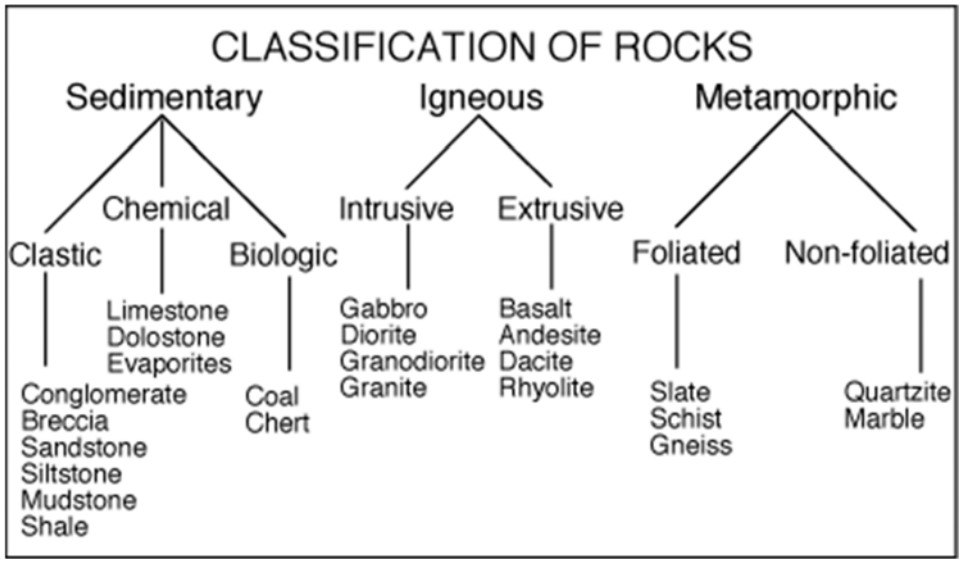
Concept Dictionary- Example

Concept	Synonyms/Abbr	Meaning
Esophagus	Gullet oesophagus	Sometimes known as the gullet. Muscular tube through which food passes from pharynx to the stomach
Duodenum		The first section of the small intestine
Peptic ulcer	PUD	Area of the gastrointestinal tract that is extremely painful. Mucosal erosions equal to or greater than 0.5cm.
Hyperacidity	Acid dyspepsia, Amalpitta	A condition of excreting more than the normal amount of hydrochloric acid in the stomach

Concept Tree

- Tree that shows concepts and the classes and sub-classes.
- All relationships must be “is a”.
- Check the tree by looking at the lowest and highest nodes and asking “is <sub-concept> a type of <concept>”
- Nodes should have clear & complete names.
- Helps the knowledge engineer understand the terminology and breadth of the domain.
- An essential knowledge model for most KA projects

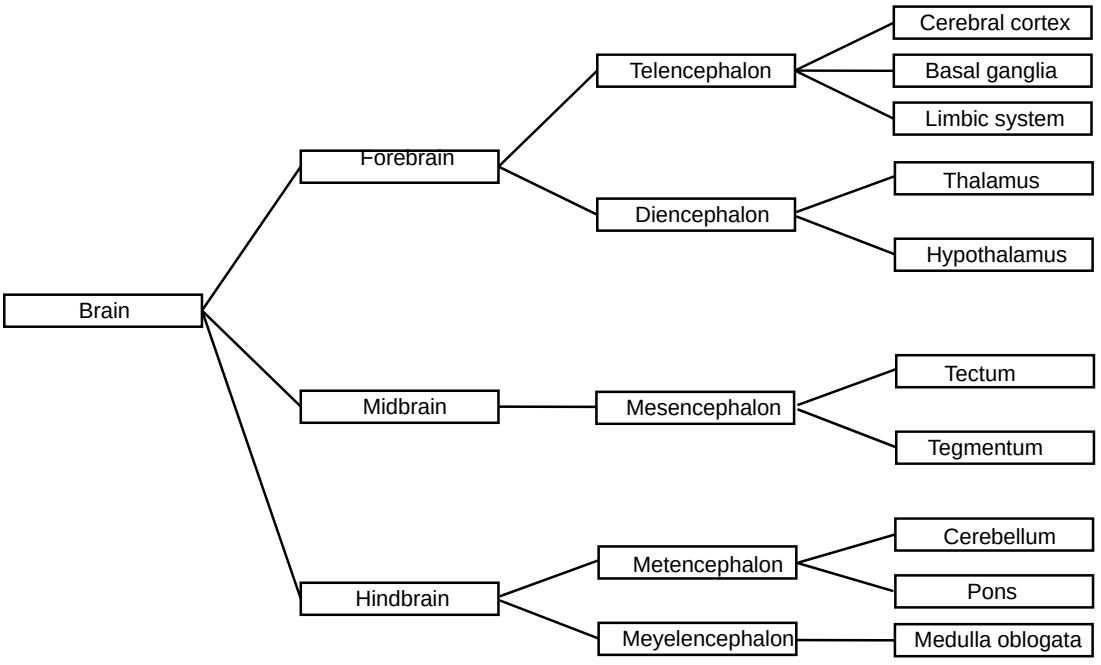
Concept Tree - Example



Composition Tree

- Tree showing the way a concept breaks-down into its constituent parts.
- All relationships must be “part of”.
- A useful way of understanding such things as
  - Products (parts of a machinery)
  - Organisations (your organisation chart)
  - Documents (the table of contents)

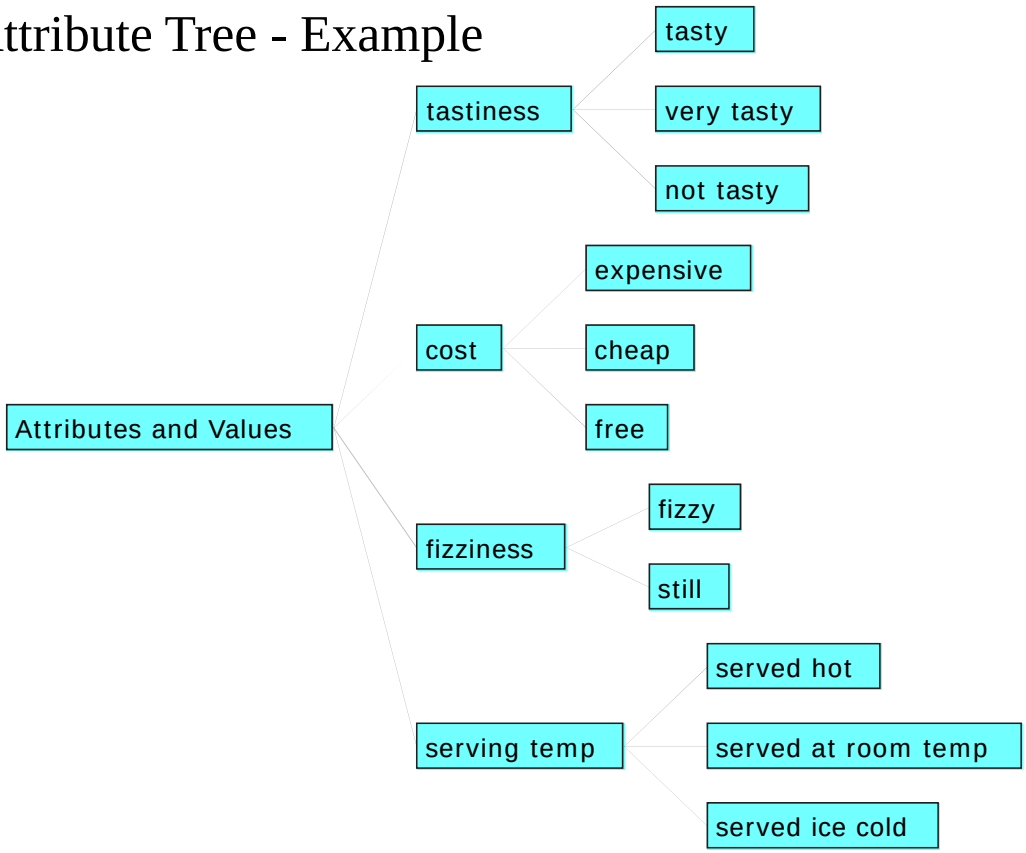
Composition Tree - Example



Attribute Tree

- Tree showing the attributes and values in a domain.
- All adjectival values are shown as sub-nodes of the appropriate attribute.
- Numerical values are not usually shown.
- Useful way of capturing detailed concept knowledge, i.e. the properties of concepts.

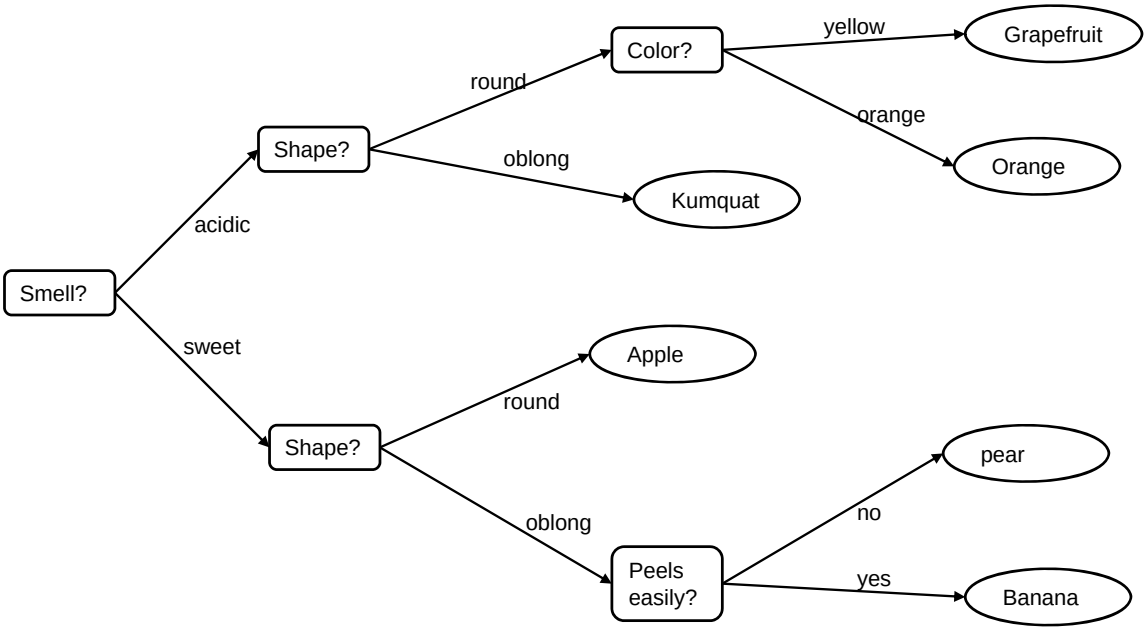
Attribute Tree - Example



Decision Tree

- Tree showing the alternative courses of action for a particular decision.
- A useful way of capturing detailed process knowledge.
- Provides a snapshot of the experts inference.

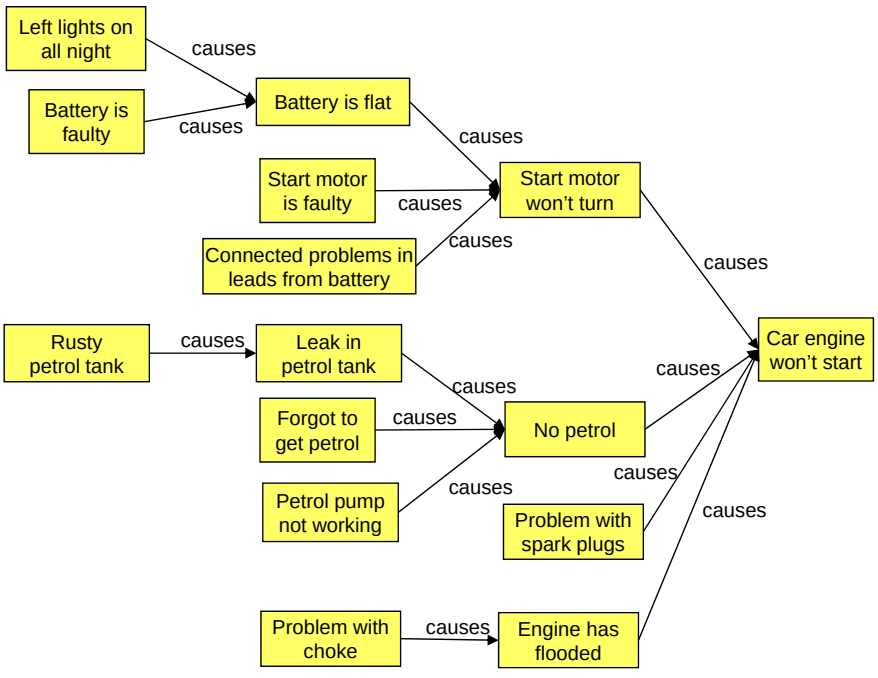
Decision Tree – Example



Cause Tree

- Type of tree showing the way a problem is caused by things, which in turn are caused by other things
- Uses only the *causes* relationship.
- Good way of representing the knowledge needed when an expert performs a diagnosis activity.

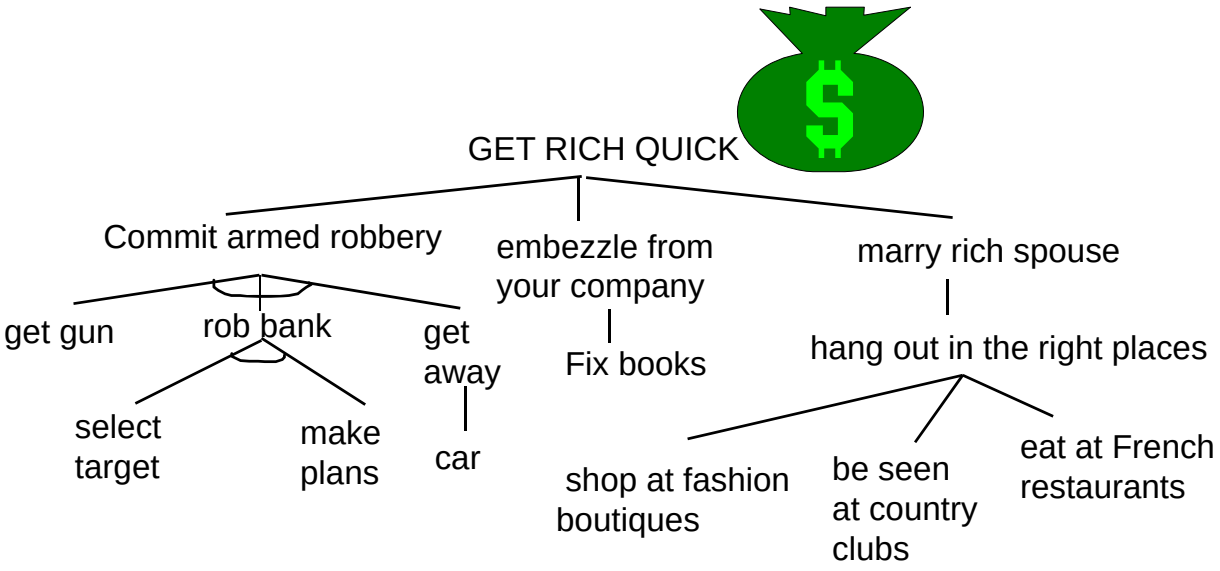
Cause Tree - Example



Goal Reduction Tree

- Indicate relations between goal & sub-goals
- Incorporates AND/OR links
  - OR nodes implicit
- Typical patterns of problem-solution behaviour can be analyzed

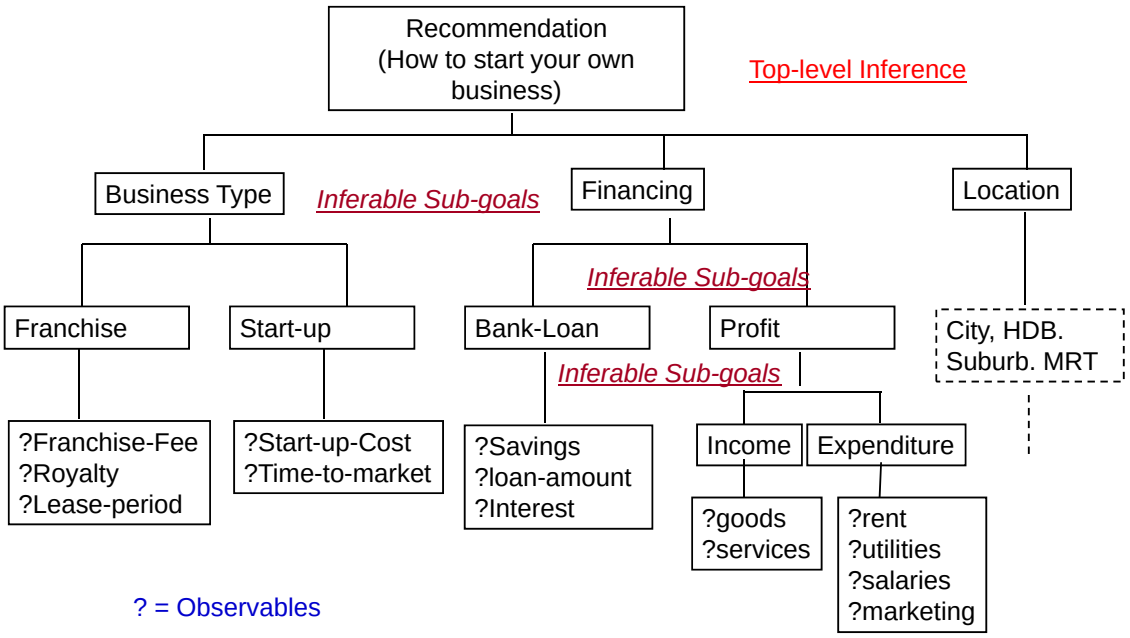
Goal Reduction Tree - Example



What is a Dependency Diagram?

- A Dependency Diagram is a type of **Knowledge Model** that expresses the problem-solving domain knowledge captured during the **Knowledge Acquisition** phase.
- A Dependency Diagram consists of problem-solving **factors** arranged in a hierarchical tree structure.
- The top-most level node is the final Decision.
- The intermediate nodes are the sub-factors.
- The leave nodes are the input-factors.
- It is called a dependency diagram because it clearly shows all the **factor dependencies** needed in a specific decision-making.
- It is sometimes referred to as an Inference Diagram because it also shows the different level of inferences in the decision-making.

Dependency (Inference) Diagram



Goal Trees vs Decision Trees

- Goal Reduction Trees are more “action” oriented while the Decision Tree is more static.
- Decision Trees have labeled links where every node has a specific question.

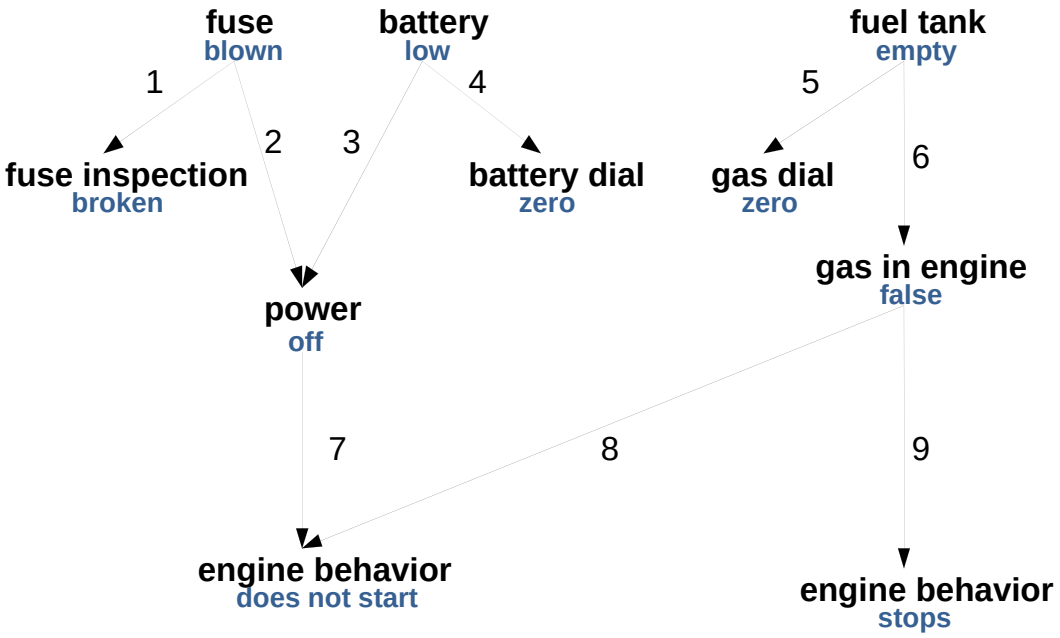
Improving the capture process

- Focus on how experts approach the problem – i.e. look beyond facts and heuristics
- Have you understood the problem domain well enough
  - Are you modeling the knowledge accurately?
  - Do you see patterns and relationships leading to the solution?
- Focus on episodic knowledge
  - Use past case situations or problem scenarios
- Plan each interview carefully:
  - Minimize the effort spent in gathering, transcribing, analyzing
  - Minimize the time spent with domain experts
  - Develop common practice and clear standards

Task Modeling

- Task is a high level description of a problem solving strategy – e.g. diagnostic, assessment
- Make use of the CommonKads **Inference Structure** diagrams
- Inference structure diagram represent a detailed design of the problem solving steps in the task
  - The Inference Structure specifies the *knowledge* and *reasoning* requirements of the proposed system.
  - Supports and distinguishes between different types of knowledge.
- CommonKads has several re-usable templates

Cause Tree

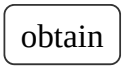


Graphical conventions

- Inference structure diagrams uses the following graphical notations:

Rectangle = Input/Ouput

Ovals = Inference

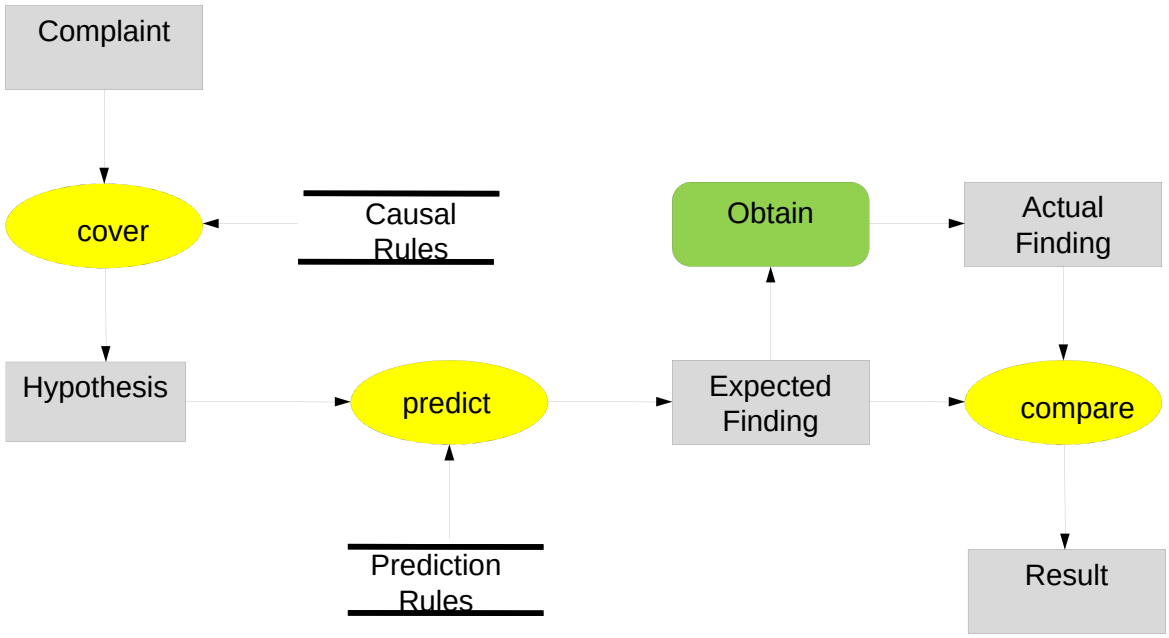


Rounded box = transfer function

2 thick horizontal lines = static knowledge (Rules)

Arrows = input/output dependencies

Inference Structure (Diagnostic)



Inference

- Inference represents the lowest level of functional decomposition, and carries out primitive reasoning step.
- Inference is carried out using Static Knowledge (rules) to derive new information from its input.
- No formal specification of internal behaviour of inference is provided – this should be self-evident by simply looking at the input/output specification for the inference.

Cover

Operation	Given some effect, derive a system state that could have caused it.
Example	Cover the complaints about a car to derive potential faults.
Static Knowledge	A behavioural model of the system being diagnosed. A causal network is the most common used knowledge.
Typical Task Types	Cover is specific for diagnosis type of tasks
Control Behaviour	Cover produces multiple solutions for the same input
Computational Model	Abductive reasoning methods are used. Can range from simple to complex depending on the nature of the diagnostic method used.

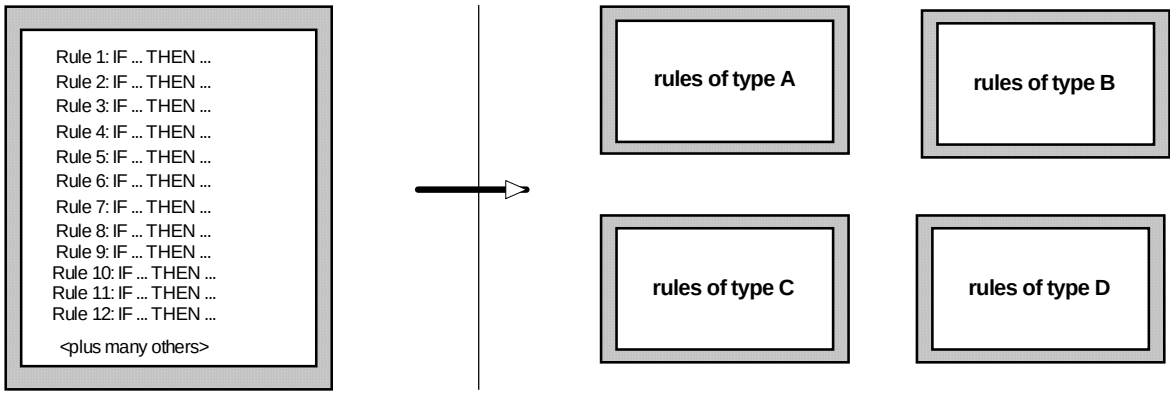
Predict

Operation	This inference generates a prediction of the system state at some point in the future when given the input such as a description of the current system state.
Example	Predict the blood pressure of a patient. Predict whether the stock will go up or down or by how much.
Static Knowledge	Domain knowledge is a model of the system behavior. Can be quantitative or qualitative.
Typical Task Types	Mostly used in analytic tasks.
Control Behaviour	Predict produces a single output.
Computational Model	Qualitative or quantitative reasoning algorithms.

Compare

Operation	Input to compare are two objects. This inference returns equal if the two objects are the similar. If not, it returns either not-equal or some numerical value indicating the difference.
Example	Comparison of two findings – one that is predicted by the system (expected results) and the one that is actually observed (via user input)
Static Knowledge	For simple cases, no domain knowledge is required – knowldege can be implemented as a lookup table. In other cases, domain knowledge may be needed to make the comparison – for example a similarity algorithm based on deep domain knowledge might be needed.
Typical Task Types	Commonly found in analytical tasks.
Control Behaviour	Compare produces a single solution.
Computational Model	Often requires only simple techniques.

Structuring a Rule-base



single flat
knowledge base

multiple rule sets
containing rules
with similar structure

Modelling with templates

- Inference structures are models that can be reused in new applications;
  - prevent "re-inventing the wheel"
  - cost/time efficient
  - decreases complexity
- CommonKADS provides a library of generic inference structures which are used as a starting point for the development of inference structures for particular projects
- The type of the task is used as the main guide for choosing a template.

Computational Thinking

- "Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent." Cuny, Snyder, Wing

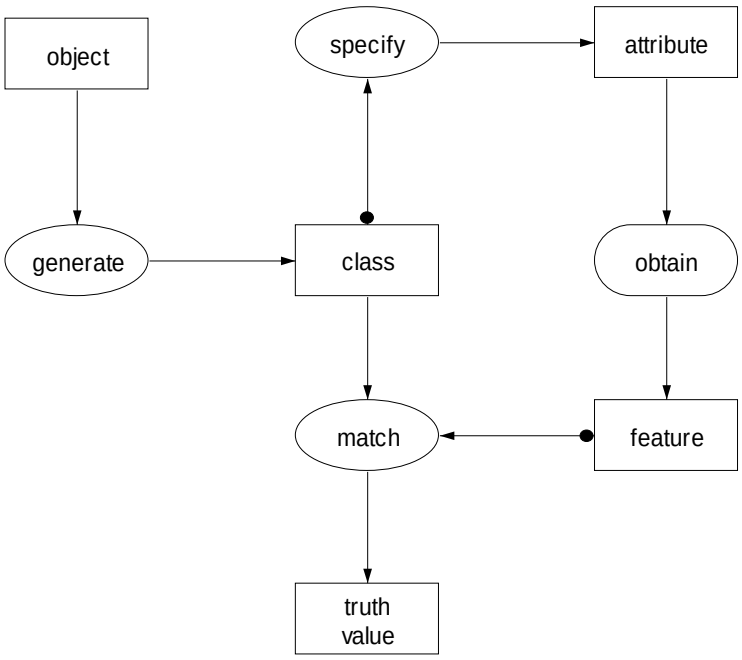
Inference Structure Templates

- Inference Structures are specific for a task type
- Supports top-down knowledge modeling
- Select a model and modify
  - Templates will have to be refined as the model may not fit the application exactly

Task types

- Analytics Tasks:
  - Classification
  - Assessment
  - Diagnosis
  - Monitoring
- Synthetic Tasks:
  - Configuration
  - Assignment
  - scheduling

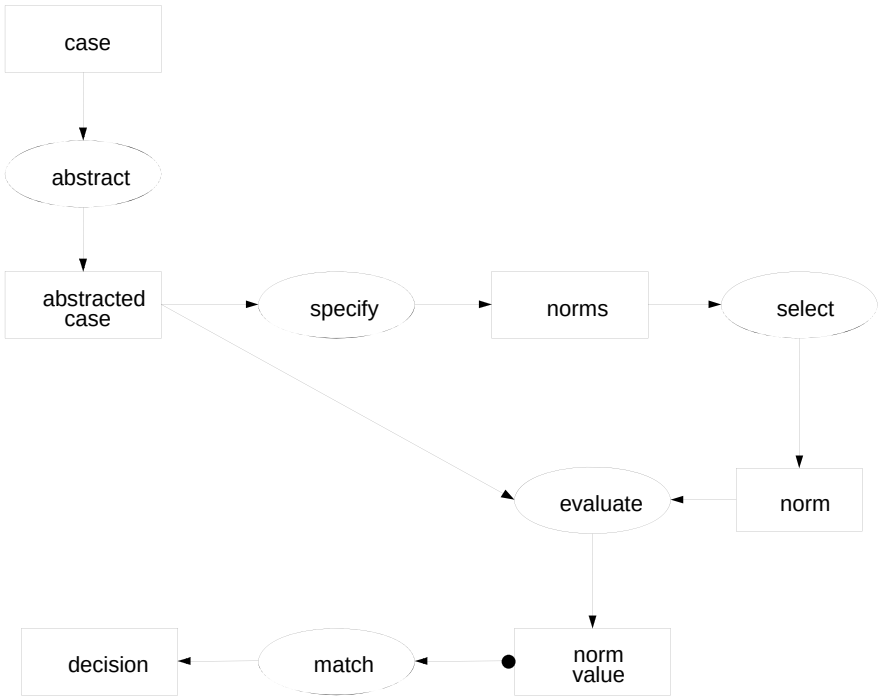
Classification



Control Method

```
while new-solution generate(object -> candidate) do
    candidate-classes := candidate union candidate-classes;
end while
while new-solution specify(candidate-classes -> attribute)
          and length candidate-classes > 1 do
    obtain(attribute -> new-feature);
    current-feature-set := new-feature union current-feature-set;
    for-each candidate in candidate-classes do
        match(candidate + current-feature-set -> truth-value);
        if truth-value = false;
        then candidate-classes := candidate-classes subtract candidate;
        end if
    end for-each
end while
```


Assessment



Control Method

while **new-solution** abstract(case-description -> abstracted-case) **do**

case-description := abstracted-case;

end while

specify(abstracted-case -> norms);

repeat

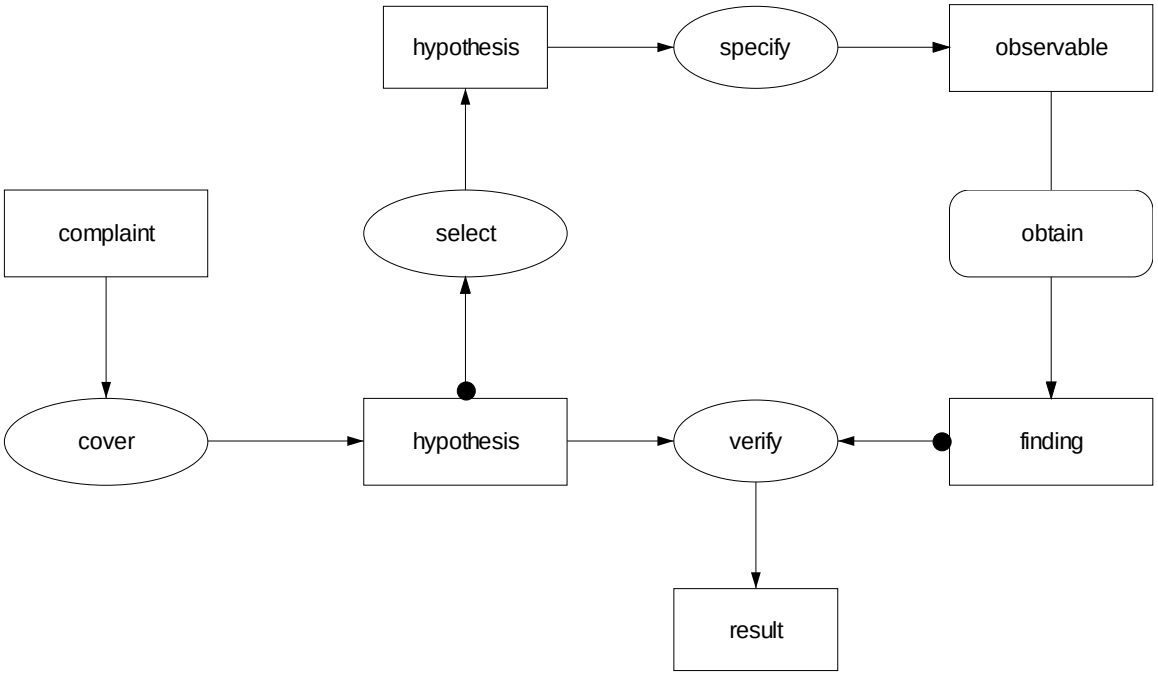
select(norms -> norm);

evaluate(abstracted-case + norm -> norm-value);

evaluation-results := norm-value **union** evaluation-results;

until **has-solution** match(evaluation-results -> decision);

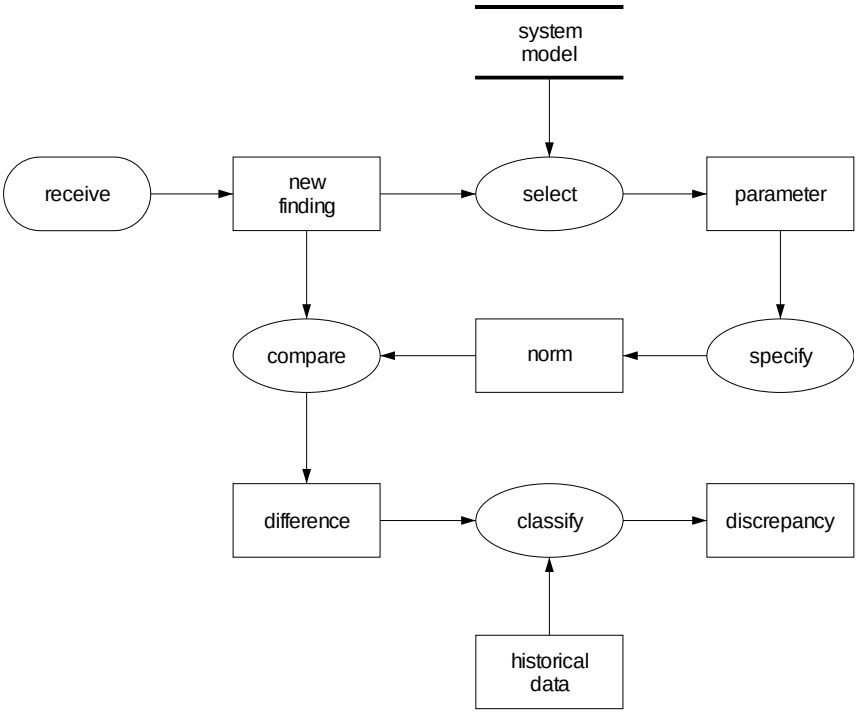
Diagnosis: inference structure



Control Method

while new-solution cover(complaint -> hypothesis) do
    differential := hypothesis **add** differential;
end while
repeat
    select(differential -> hypothesis);
    specify(hypothesis -> observable);
    obtain(observable -> finding);
    evidence := finding **add** evidence;
    foreach hypothesis **in** differential **do**
        verify(hypothesis + evidence -> result);
        if result = false **then** differential := differential **subtract** hypothesis
until length differential =< 1 **or "**no observables left"
faults := hypothesis;

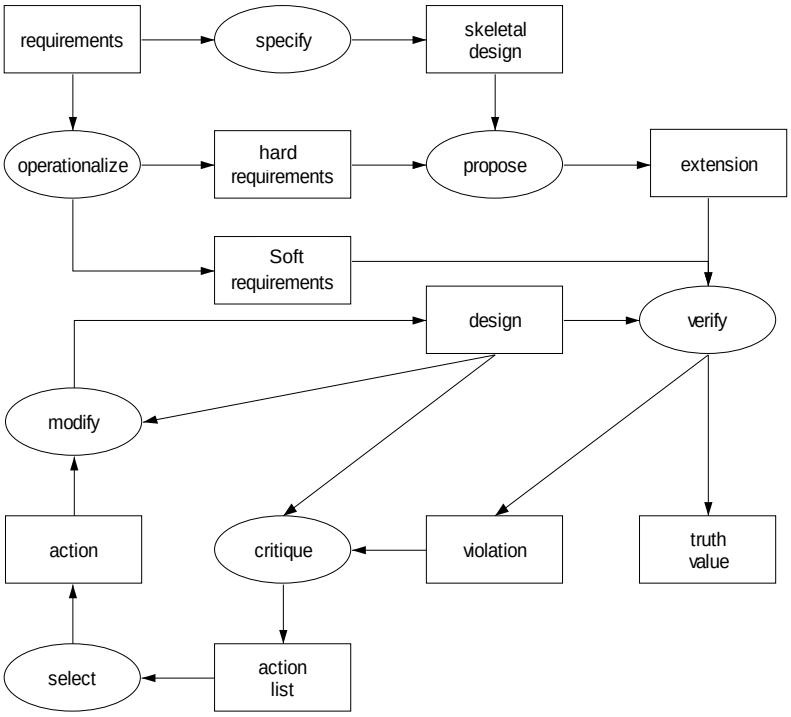
Monitoring



Monitoring: method control

receive(new-finding);
select(new-finding -> parameter)
specify(parameter -> norm);
compare(norm + finding -> difference);
classify(difference + historical-data -> discrepancy);
historical-data := finding **add** historical-data;

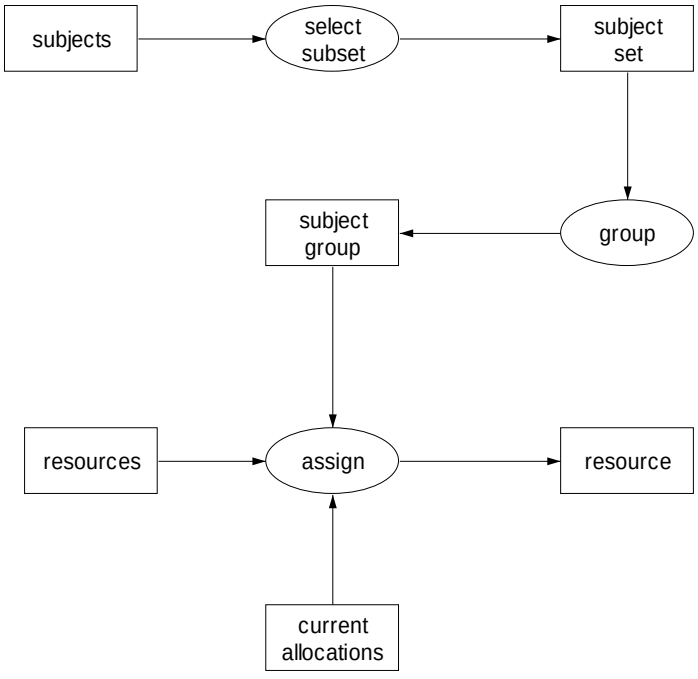
Configuration



Control Method

operationalize(requirements -> hard-reqs + soft-reqs);
specify(requirements -> skeletal-design);
while **new-solution** propose(skeletal-design + design +hard-reqs -> extension) **do**
design := extension **union** design;
verify(design + soft-reqs -> truth-value + violation);
if truth-value = false **then**
critique(violation + design -> action-list);
repeat select(action-list -> action);
modify(design + action -> design);
verify(design + soft-reqs -> truth-value + violation);
until truth-value = true;
end while

Assignment

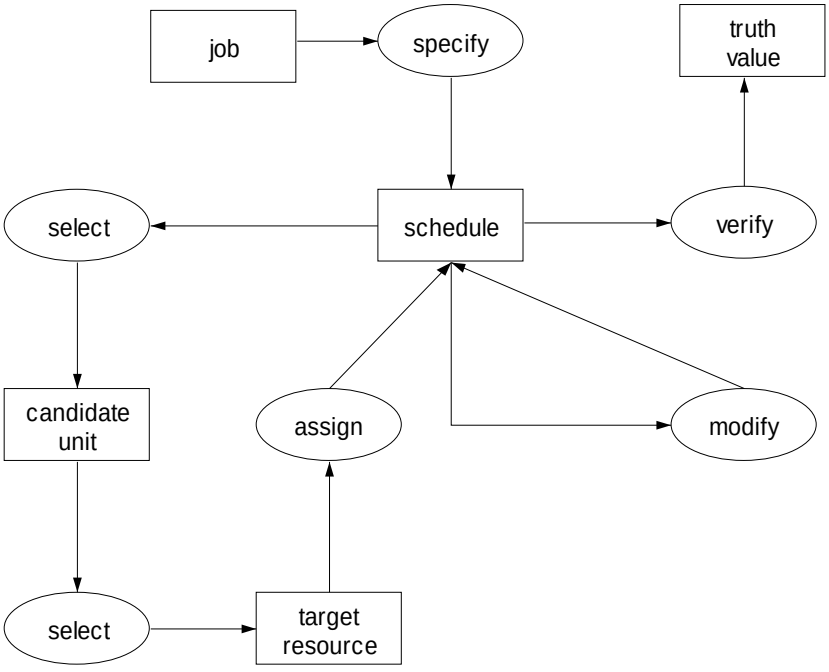


Control Method

```
while not empty subjects do
    select-subset(subjects -> subject-set);
    while not empty subject-set do
        group(subject-set -> subject-group);
        assign(subject-group + resources + current-allocations ->
                                    resource);
        current-allocations := < subject-group, resource > union
                                            current-allocations;
        subject-set := subject-set/subject-group;
        resources := resources/resource;
    end while
    subjects := subjects/subject-set;
end while
```

Note: forward slash '/' means 'delete' or remove from the set

Scheduling



Control Method


```
specify(jobs -> schedule);
while new-solution select(schedule -> candidate-unit) do
   select(candidate-unit + schedule -> target-resource);
   assign(candidate-unit + target-resource -> schedule);
   verify(schedule -> truth-value);
   if truth-value = false then
        modify(schedule -> schedule);
end while
```