Chapter 4
Rule Based Reasoning

Abstract The popularity of rules-based systems (RBSs) is due to their naturalness. This chapter presents the potential applications of RBSs, the working of RBS, forward and backward chaining RBSs, their Algorithms, and inferencing using these systems. The analysis of complexity of preconditions, and efficiency of rule selection are introduced to sufficient depth, as well the cofmparison between the two types of RBSs are presented. A typical RBS, and other methods—model-based and case-based approaches are also discussed. In addition, number of solved, as well exhaustive list of exercises are provided at the end of the chapter for practice. The chapter concludes with its summary.

Keywords Rule-based systems (RBSs) · Forward chaining · Backward chaining · Forward chaining Algorithm · Backward chaining Algorithm · Model-based reasoning · Case-based reasoning · Conflict resolution

4.1 Introduction

Symbolic rules are popular for knowledge representation and reasoning. Their popularity stems mainly from their naturalness, which facilitates comprehension of the represented knowledge. The basic form of a rule is,

$$if <conditions> then <conclusion>$$

where $<conditions>$ represent the *conditions* or *premises* of a rule, and the $<conclusion>$ represent its conclusion or consequence. The conditions of a rule are connected between each other with logical connectives such as *AND/OR* thus forming a logical function. When sufficient conditions of a rule are satisfied, the conclusion is derived and the rule is said to *fire* (or *trigger*). Rules represent general knowledge regarding a domain.

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_4

Table 4.1 Application areas of rule based systems

Problem	System functions
Troubleshooting	Analyzing situations, suggesting measures, and prescribing preventative actions
Process control	Identifying problematic data and remedies of inequalities
Quality assurance	Assessment of tasks, proposing the practices, and enforcing the requirements
Equipment maintenance	Diagnosing various faults and recommending the repairs
Component selection	Specifying requirements and matching parts from an electronics catalog
Computer operation	Analyzing requirements to be fulfilled, selecting and operating a software
Product configuration	Specifying preferences, and identifying parts that satisfy constraints

In this chapter we will discuss the use of easily stated *if-then* rule to solve problems. In particular you will learn the *forward-chaining* from the assertions and *backward* chaining from hypotheses. The illustrative examples will explain the rule firing sequences and graph search. The complexities of Algorithms for both forward and backward rule chaining are also analyzed and presented. In addition, the complexities of preconditions of rules, and other popular types of *expert systems*, like—*Case based reasoning*, and *model based reasoning* are presented.

Rule-based systems (RBSs) constitute the one of the most common and simple to implement means for codifying the problem-solving know-how of human experts. Experts tend to express most of their problem-solving techniques in terms of a set of *situation-action* rules, and this suggests that RBSs should be the method of choice for building knowledge-intensive expert systems. The RBSs share following key properties [3]:

1. They incorporate practical human knowledge in conditional *if-then* rules,
2. Their skill increases at a rate proportional to the enlargement of their knowledge bases,
3. By selecting the relevant rules and combining them appropriately, it is possible to solve a large range of problems, of varying complexities.
4. They can be designed to adaptively decide the best sequence of rules to execute, and
5. The RBS can retrace the reasoning steps and explain the justification for the conclusions drawn. In this retracing, it can show each rule executed, using natural language.

Table 4.1 lists the major application areas addressed by RBS.

Learning Outcomes of this Chapter:

1. Explain the difference between rule-based, case-based and model-based reasoning techniques. [Familiarity]
2. Inferences in rule-based systems. [Usage]
3. Complexities of rule-based systems. [Assessment]

4.2 An Overview of RBS

Roughly speaking, a Rule Based System comprises a *knowledge base* and an *inference engine* (see Fig. 4.1). The knowledge base is collection of facts and rules. Facts are simple statements with constant or variable object lists, while the Rules always express a conditional sentence, with one or more premises and a consequent component. The rules needs to be interpreted, which means, if a premise can be satisfied the consequent also be satisfied. The consequent can be a conclusion or an action. When it is an action, the effect of satisfying the premises is to schedule that action for execution, and when the consequent defines a conclusion, the effect is to infer the conclusion [2].

Several key techniques for organizing RBSs have emerged. Rules can be used to express *deductive knowledge*, such as logical relationships, and thereby to support inference, verification, or evaluation tasks. Conversely, rules can be used to express goal-oriented knowledge that an RBS can apply in seeking problem solutions and cite in justifying its own goal-seeking behavior. The rules can also be used to express *causal* relationships, which an RBS can use to answer "what-if" questions, or to determine possible causes for specified events.

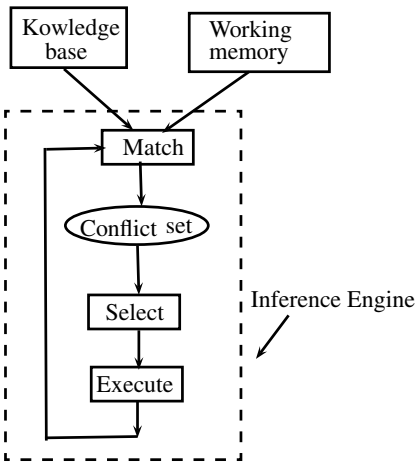

Fig. 4.1 Inference cycle of a forward-chaining RBS

In a rule based system, each *if* pattern may match to one or more of assertions in a collections of assertions. The collections of assertions is called *working-memory* (Fig. 4.1). The assertions collectively may match to premises of one or more rules in the knowledge base. This is done by "match" block of the inference-engine. All the rules matching with the assertions in the working memory are put in the *conflict set*, from where one of the rule is "selected" based on some conflict resolving criteria set for, and then the selected rule is executed. The resulting consequences/conclusions (the then patterns) are put in the working memory to form new assertions, and the process continues till desired result (goal) is not reached.

A system like this is called *deduction system*, as it deduces new inferences from the rules and assertions. A realistic example of a rule is:

```
R1: if 1. stiff neck,
       2. high temperature, and
       3. impairment of conciousness occur togetehr,
    then
          meningitis is suspected.
```

In the above, meningitis is a disease related to"Inflation of membrane of spinal chord and brain" will be suspected by this rule if the "if" patterns 1, 2, 3 are found true.

RBS can be applied in judiciary systems also. Following is an example of a more complex a rule.

```
If the plaintiff received an eye injury
 and it was one eye injured
 and treatment for eye required surgery
 and recovery from the injury was almost complete
 and visual acuity was slightly reduced by the injury
 and there is fixed condition,
then increase the injury trauma factor by $5,000.
```

Facts is the kind of data in a knowledge base that express assertions about properties, relations, propositions, etc. In contrast to rules, which the RBS interprets as imperatives, facts are usually static and inactive implicitly. Also, a fact is silent regarding the pragmatic value and dynamic utilization of its knowledge. Although in many contexts facts and rules are logically interchangeable, in the RBSs they are quite distinct.

In addition to its static memory for facts and rules, an RBS uses a *working-memory* to store temporary assertions. These assertions record earlier rule-based inferences. We can describe the contents of working memory as problem-solving state information. Ordinarily, the data in working memory adhere to the syntactic conventions of facts. Temporary assertions thus correspond to dynamic facts.

The basic function of an RBS is to produce results. The primary output may be—a problem solution, an answer to a question, or result of an analysis of some data. Whatever the case, an RBS employs several key processing determining its overall

activity. A *world* manager maintains information in working memory, and a built-in control procedure defines the basic high-level loop; if the built-in control provides for programmable specialized control, an additional process manages branches to and returns from special control blocks.

Some times, the patterns specify the *actions* rather than *assertions*, e.g., "to put them on the table". In such case the rule based system is called *reaction system*.

In both the deduction systems and reaction systems, forward chaining is the process of moving from the *if* patterns to *then* patterns, where *if* patterns identifies the appropriate situation for deductions of new assertion, and performance of an action in the case of *reaction system*.

4.3 Forward Chaining

In a forward chaining system, we start with the initial facts, and use the rules to draw new conclusions (or take certain actions), given those facts. Forward chaining systems are primarily *data-driven*. Whenever an if pattern is observed to match an assertion, the antecedent is *satisfied*. When all the *if* patterns of a rule are satisfied, the rule is *triggered*. When a triggered rule establishes a new assertion or performs an action, it is *fired.* This procedure is repeated until no more rules are applicable (Fig. 4.1).

The selection process is carried out in two steps:

1. *Pre-selection*: Determining the set of all the matching rules, also called the *conflict-set*.
2. *Selection*: Selection of a rule from the conflict set by means of a conflict resolving strategy.

4.3.1 Forward Chaining Algorithm

A simple forward chaining Algorithm 4.1 starts from the known facts in the knowledge base, and triggers all the rules whose premises are the known facts, then adds the consequent of each into the knowledge base. This process is repeated until the query is answered or until there is no conclusion generated to be added into the knowledge base. We will use symbols θ, λ, γ to represent substitutions. The *unify* is a function unifies the newly generated assertion q' and the query α, and returns a unifying substitution λ if they are unified, else returns null.

Let α be the goal. The forward-chaining Algorithm 4.1 picks up any sentence $s \in \Gamma$, where Γ is knowledge base, and checks all possible substitutions θ for s. Let, the predicate form of s is $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow q$. On substituting θ, say, it results to $(p_1 \wedge p_2 \wedge \ldots p_n)\theta = (p'_1 \wedge p'_2 \wedge \ldots p'_n)$, such that p'_i's $\in \Gamma$. Let $(p'_1 \wedge p'_2 \wedge \ldots p'_n) \rightarrow q'$, such that $q' = q\theta$. This q' is added into *new* inference. If the

inference q' and goal α unify, then their unifier λ is returned as the solution, else the Algorithm continues.

Algorithm 4.1 Forward-chaining(Input: Γ, α) // α is a query, Γ is knowledge base

```
1: while True do
2:   new = { }
3:   for each sentence s ∈ Γ do
4:     Convert s into the format p1 ∧ p2 ∧ ··· ∧ pn → q
5:     for each substitution θ such that (p'1 ∧ p'2 ∧ ..., ∧ p'n) ← (p1 ∧ p2 ∧ ..., ∧ pn)θ for
       some p'i s ∈ Γ do
6:       q' ← qθ
7:       new ← new ∪ {q'}
8:       λ ← unify(q', α)
9:       if λ is not null then
10:          return λ
11:      end if
12:    end for
13:    Γ ← Γ ∪ new // add new inferences in knowledge base
14:  end for
15: end while
16: Return Fail
```

The Algorithm may fail to terminate in the case when the raised query has no answer. For example, every natural number can be generated by recursively applying successor operation on a natural number, and assuming that 0 is natural number. This will lead to indefinite loop for very large numbers.

$$naturalnum(0).$$
$$\forall x[naturalnum(x) \rightarrow naturalnum(succ(x)).]$$

The following example demonstrates the manual run of forward chaining Algorithm.

Example 4.1 Produce the inference, given the knowledge base $\Gamma = \{man(x) \rightarrow mortal(x), man(socrates)\}$ and query $\alpha = mortal(w)$, i.e., "Who is mortal?"

We follow the Algorithm 4.1 manually and note that $p_1 \wedge \cdots \wedge p_n = p_1 = man(x)$; $p'_1 = man(socrates)$, substitution $\theta = \{socrates/x\}$, and $q = mortal(x)$. Also,

$$\begin{aligned} q' &= q\theta \\ &= mortal(x)\{socrates/x\} \\ &= mortal(socrates). \end{aligned}$$

Also, $new = \{\} \cup \{q'\} = mortal(socrates)$.
On unification of α (i.e, $mortal(w)$), and q', the unifier λ obtained is,

$$\begin{aligned}\lambda &= unify(q', \alpha) \\ &= unify(mortal(socrates), mortal(w)) \\ &= \{socrates/w\}.\end{aligned}$$

Hence, the answer for query is $w = socrates$. □

Complexity Issues

The complexity of the forward chaining Algorithm is determined by the inner loop in the Algorithm 4.1. It finds all the possible premises such that the premises unify with certain set of facts in the knowledge base Γ. The process is called *pattern matching*, and tried for every rule, for every substitution θ. Thus, if set of rules are P, there are n^k substitutions for each rule, assuming that in worst case k arguments and n number of literals' assignments for each argument in the P. This makes the complexity of above Algorithm as,

$$|P|n^k \tag{4.1}$$

which is exponential. However, since the size and arities are constant in the real-world, the complexity of expression (4.1) is a polynomial in nature. To search the predicates for matching, they are indexed and a hash function is generated for quick search.

4.3.2 Conflict Resolution

When we are doing data-directed reasoning, we may not like to fire all of the rules in case more than one rule is applicable. In cases where we want to eliminate some applicable rules, some kind of conflict resolution is necessary for arriving at the most appropriate rule(s) to fire. In a deduction system all rules generally fire, but in a reaction system, when more than one rule are triggered at the same time, only one of the possible actions is desired [4].

The conflict resolving strategy is used to avoid superfluous rules. The most obvious strategy is to choose a rule at random, out of those that are applicable. Following are some common approaches for deciding the preferences for the rule to fire.

Selection by order

Selection by order may either on the order in which the rule appears in the knowledge base, or the order may be based on how recent the rule was used.

Order

Pick the first applicable rule in the order they have been presented. This is the type of strategy used by *Prolog*, and is one of the most common ones. Production system programmers would take this strategy into account when formulating rule sets.

Recency

Select an applicable rule based on how recently it has been used. There are different versions of this strategy, ranging from firing the rule that matches on the most recently created (or modified) wff, to firing the rule that has been least recently used. The former could be used to make sure that a problem solver stays focused on what it was just doing (typical of depth-first search); the latter would ensure that every rule gets a fair chance to influence the outcome (typical of breadth-first search).
Selection according to syntactic structure of the rule
There are many ways one can consider the syntax, e.g., it may be conditions and their specificity, or the size of the rule: largest, smallest, or in increasing order of size.

Specificity Criteria

Select the applicable rule whose conditions are most specific. A set of conditions is taken as more specific than other if the set of wffs that satisfy it is a subset of those that satisfy the other. For example, consider the following three rules:

i) *if* $bird(x)$ *then add*$(canfly(x))$
ii) *if* $bird(x) \wedge weight(y, x) \wedge \text{gt}(y, 5kg)$ *then add*$(cannotfly(x))$
iii) *if* $bird(x) \wedge penguin(x)$ *then add*$(cannotfly(x))$

Here, the second and third rules are both more specific than the first, because the condition in (ii) and (iii) are more specific than in (i) (condition of (i) is very general). If we have a bird that has weight greater than 5 kg, or it is a penguin, then the first rule applies, because the condition of "bird" is satisfied. However, in this case the other two rules should take precedence, as they are more specific. Note that if the bird is a penguin and heavy, neither second nor third is applicable, and another conflict resolution criteria must be applied to decide between the second and third rules.

Largest Rule First

Apply the syntactically largest rule, i.e., the rule which contains the most propositions.

Incremental in Size

Solutions to subproblems are constructed incrementally, and are cached to avoid the recompilation.

Selection by Means of Supplementary Knowledge

The supplementary knowledge about the rules may be in the form of priority of the rules, or as meta-knowledge.

Highest Priority First

For this purpose each rule must be given a priority, which may be represented, e.g., by a numeric value.

Apart from the priority criteria, there are additional rules, called *meta-rules*, control the selection.

The simple way of implementing the forward-chained interpreter is to check the preconditions of all the rules, sort the applicable rules in an agenda (i.e., an ordered list) according to the criteria given by the conflict-resolving strategy, and then perform the action of the leading rule in the agenda, followed with this previous agenda is deleted and the process begin as a new.

4.3.3 Efficiency in Rule Selection

Following are the criteria for selection of rules for implementation of reasoning process.

Similarity Between Rules

The first improvement over the brute-force approach is based on the similarity between preconditions of different rules. Using this, rules with a common propositions may be eliminated all together if these propositions do not hold. For achieving this, the rules are structured in a tangled hierarchy by constructing a network of interconnecting trees in which the internal decision nodes are formed by the propositions and the leaves of the rules. The path from the root of the tree to a rule includes all the propositions of the precondition of this rule.

Check Applicability of New Rules

This improvement is based on the fact that a complete new calculation of an agenda requires far more effort than a modification on the basis of changes. From one cycle to next, the knowledge base is altered only by the action of the selected rule, which may add or delete portions of knowledge. Thus, it needs to be tested that whether the deleted knowledge destroy the applicability of any rule in the old agenda and whether the added knowledge make any new rules applicable. The later test is performed by running through the rule network with new knowledge as starting point.

Indexing

One way of improving over the brute-force Algorithm is to index all the rules so that each parameter of the predicate produces a reference to its rule. When a parameter takes on a new value or changes its value, the reference enables the rules concerned to be found and checked efficiently.

Consider the following rule, for which many conjunct ordering may be possible,

$$\forall x \forall y [p(x) q(x, y) \rightarrow r(x, a)]. \tag{4.2}$$

Here, all the $p(x)$ may be ordered, next find $q(x, y)$ for each x and y. Alternatively, for each x find $p(x)$, then find all $q(x, y)$ for different y. Which approach is better? Finding this is solution of a *conjective-ordering problem*, which is *NP-hard*. A heuristic can use most constrained, the one having fewest values. This shows

a close relationship between *CSP* (constraint satisfaction problem), discussed in next chapters, and a pattern matching problem. Due to these complexity issues the forward-chaining lgorithm 4.1 is *NP-hard* in its inner loop.

4.3.4 Complexity of Preconditions

Different formalisms have differing expressive power of describing the preconditions and the rules. The differences are those between propositional and predicate logic. The count of number of rules is not a sufficient criteria for the indication of the size of an expert system. Ideally each relevant situation of the domain should be covered by just one rule. Description of a situation by different rules will impair the modularity of the knowledge base since these rules should be meaningful only as group. On the other hand, more expressive power of the rule formalism require a more complex and often more inefficient rule interpreter so that one must check how much complexity is necessary for the domain [4].

The simplest method of evaluating preconditions is lookup in the knowledge base. For example, in the case of query: "Is the pain intensifying synchronous with respiration?", it is only necessary to check whether the value—"synchronous with respiration", is stored under the object - "pain intensifying". With the logical connectives, *and*, *or*, *not*, this situation can be described. However, in the case of numerical or temporal relationships, the looking alone is not sufficient, and additional, calculations are required. For example, for the query "Did shortening of breath occur before throat pain". here, the preposition "before" has reference to time. Does it mean 1 second or a minute, or a day, because all these satisfy the criteria of before ! There is need of comparison, using the relations operators, like, less than, greater than, equal, which are needed to be applied along with the time information. Also, depending on the situation, magnitude of precede be available in the knowledge base.

One of the drawback of forward chaining is that it makes all allowable inferencing based on the facts and rules available, irrespective of whether they are relevant to deriving goals or not. This makes the forward chaining, not a very efficient approach for reasoning. To avoid this problem, we work for selected subclass of rules. The other approach, the backward chaining, arrives to only those premises which certainly lead to goal, hence the system is called goal driven.

4.4 Backward Chaining

While forward chaining allows conclusions to be drawn only from a given knowledge base, a backward-chained rule interpreter is suitable for requesting still unknown facts. In a backward chaining system, you start with some hypothesis (goal) you are trying to prove, and keep looking for rules that would allow you to conclude that hypothesis, perhaps setting new sub-goals to prove as you go.

For this, the goal expression is initially placed in the working memory, followed with this, the system performs two operations:

1. matches the rule's "consequent" with the goal. For example, q in $p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow q$,
2. selects the matched rule $p_1 \wedge p_2 \wedge \ldots p_n \rightarrow q$ and places its premises $(p_1 \wedge \cdots \wedge p_n)$ in the working memory.

The second step in above corresponds to the transformation of the problem into subgoals. The process continues in the next iteration, with these premises becoming the new goals to match against the consequent of other rules. Thus, the backward chaining system, in the human sense are *hypothesis testing*.

Backward chaining uses *stack*. First, the goal is put in the stack, then all the rules which results to this goal are searched and put on the stack. These becomes the subgoals, and the process goes on till all the facts are proved or are available as literals (ground clauses). Every time the goal and premises are decided, the unification is performed.

4.4.1 Backward Chaining Algorithm

The Algorithm for backward chaining returns the set of substitutions (*unifier*) which makes the goal true. These are initially empty. The input to the Algorithm is knowledge base Γ, goals α, and current substitution θ (initially empty). The Algorithm returns the substitution set λ for which the goal is inferred. The Algorithm 4.2 is the backward-chaining Algorithm.

Algorithm 4.2 Backward-chaining(Input: Γ, α, θ) // α is a query, Γ is knowledge base, θ current substitution (initially empty), λ represent substitution set for the query to be satisfied (initially empty)

1: $\theta = \{\}, \lambda = \{\}$
2: $q' \leftarrow \alpha\theta$
3: **for** each sentence $s \in \Gamma$, where $s = p_1 \wedge p_2 \wedge \ldots, \wedge p_n \rightarrow q$ and $\gamma \leftarrow unify(q, q') \neq null$ **do**
4: $\quad \alpha_{new} \leftarrow (p_1 \wedge p_2 \wedge \cdots \wedge p_n)$
5: $\quad \theta \leftarrow \theta\gamma$
6: $\quad \lambda \leftarrow$backward-chaining$(\Gamma, \alpha_{new}, \theta) \cup \lambda$
7: **end for**
8: return λ

Example 4.2 Apply the backward-chaining Algorithm for inferencing from a given knowledge base.

Let $\Gamma = \{man(x) \rightarrow mortal(x), man(socrates)\}$. Assume that it is required to infer answer for "Who is mortal?" That is, goal $\alpha = mortal(w)$, find w. The loop iterations in the backward-chaining Algorithm 4.2are as follows:

1st Iteration: Initially θ is empty, hence, $q' = \alpha\theta = mortal(w)$. From Algorithm and knowledge base Γ, the sentence is, $s = (man(x) \rightarrow mortal(x))$. Next, $\gamma = unify(mortal(x), mortal(w)) = \{w/x\}$. Also, the new goal, $\alpha_{new} = man(x)$. The new value of current substitution is, $\theta \leftarrow \theta\gamma = \{\}\{w/x\} = \{w/x\}$. Next, compute $\lambda = backward\text{-}chaining(\Gamma, man(x), \{w/x\}) \cup \lambda$, as a recursive call.

Recursive call: We apply the Algorithm in a recursive mode, and get $q' = man(x)$ $\{w/x\} = man(w)$. Next, the subgoal $man(w)$ (i.e., q') matches with other subgoal (it is a *fact*) $man(socrates) \in \Gamma$. Hence, $\gamma = unify(man(socrates), man(w)) = \{socrates/w\}$. Next, the new goal is, $\alpha_{new} = man(socrates)$ and new substitution is:

$$\begin{aligned}\theta &= \theta\gamma \\ &= \{w/x\}\{socrtaes/w\} \\ &= \{socrates/x\}.\end{aligned}$$

In the next call of recursion at step 5, $q' = \alpha\theta = man(socrates)$ $\{socrates/x\}$, the substitution fails, as there is no x where "socrates" can be substituted. Hence, $q' = null$. Since γ is null, the *for loop* at step 3 terminates, and returned value of λ is $\{socrates/w\}$, i.e., $w = socrates$, or $mortal(socrtaes)$. □

4.4.2 Goal Determination

If the goal is not known in the knowledge base, the rule interpreter first decides whether it can be derived or must be requested from user. For the derivation of goal, all the rules which includes the goal in their consequent part are processed. These rules can be identified efficiently if they are indexed according to their consequent parts.

Backward chaining therefore contains implicitly a dialogue control, where order of the questions decides the order of the rules for deriving a parameter. This dependency on the order reduces the modularity of the rule-based system. The efficiency of the backward-chained rule interpreter is determined by the formulation of the goal: the more precise the goal, the smaller is the search tree of rules to be checked and questions to be asked. For example, in medicine, a query may be: "What is name of disease?" as against an alternate query of "Is X the name of disease?".

The examples of back-ward chained rule interpreter are MYCIN, and PROLOG.

4.5 Forward Versus Backward Chaining

Whenever the rules are such that typical set of facts relate to large number of conclusions, i.e., the *fan-out* is larger than *fan-in*, it is better to use backward chaining. This is to stop in explosive growth in search space. On the other hand, when the rules

Table 4.2 Knowledge base for Animal-Kingdom

S. no.	Rule
1	$sponge(x) \rightarrow animal(x)$
2	$arthopod(x) \rightarrow animal(x)$
3	$vertebrate(x) \rightarrow animal(x)$
4	$fish(x) \rightarrow vertebrate(x)$
5	$mammal(x) \rightarrow vertebrate(x)$
6	$carnivore(x) \rightarrow mammal(x)$
7	$dog(x) \rightarrow carnivore(x)$
8	$cat(x) \rightarrow carnivore(x)$

are such that a typical hypothesis can lead to many facts, i.e., *fan-in* is larger than *fan-out*, it is better to use forward chaining. The reason being that if we go backward in a case where fan-in is high, it would lead to exponentially growing search space, so we prefer forward chaining in such knowledge.

If there is a situation that all the required facts are known, and we want to get all the possible conclusions from these, the forward chaining is better. However, if the rules are such that facts are incompletely known, we cannot proceed from the facts to conclusions, and thus goal driven strategy should be used.

Forward chaining is often preferable in cases when there are many rules with the same conclusions, because we choose the satisfying premises. In backward chaining it may lead to non-satisfying premises. The following examples demonstrate this.

Example 4.3 Given a knowledge base for an animal kingdom, infer *animal(bruno)* after adding of *dog(bruno)*. The taxonomy of the animal kingdom includes such rules as shown in Table 4.2.

It is required to show that, $KB + dog(bruno) \rightarrow animal(bruno)$.

1. *Forward Chaining.* We start with rule 7, and unify $dog(bruno)$ with $dog(x)$. Next, we will successively infer and add *carnivore(bruno)*, *mammal(bruno)*, *vertebrate(bruno)*, and *animal(bruno)*. The query will then succeed immediately. The total work is proportional to the height of the hierarchy of this taxonomy, which is 4.
2. *Backward-chaining.* Alternatively, if we use backward chaining, the query $animal(bruno)$ will unify with the first rule above and generate the sub-query $sponge\ (bruno)$, which will initiate a search for *bruno* through all the subdivisions of sponges. Not finding, it tries with 2nd rule, but orthopod is not in consequent. Next, with rule 3, the goal driven chain is: "animal $\rightarrow$ vertebrate $\rightarrow$ fish", which fails. The successful invocation rule sequence is "3 $\rightarrow$ 5 $\rightarrow$ 6 $\rightarrow$ 7." Thus, it searches the entire taxonomy of animals looking for $dog(bruno)$. We note that work done in the background chaining much more than forward chaining. However, this is not necessarily true always. □

In some cases, it is desirable to combine both forward and backward reasoning, and due to the merits of individual rules they are identified as forward / backward. However, this process results to a far more complex mechanism.

4.6 Typical RB System

The R_1 (later called as *XCON*—for e**x**pert **con**figurator) was production rule-based expert system, which was written in *OPS5* language by *John P. McDermtt* in 1978 to help in ordering DEC's VAX computer system by automatically selecting the computer systems components, based on the requirements of customers. When fully developed it had 2500 rules, which provided over 95% accuracy in configuration of systems as per customers needs [5].

The XCON differed from other systems mainly in its use of "match" rather than "generate-and-test" as its central problem solving strategy. It exploits knowledge of its task domain to generate a single acceptable solution. The input to XCON is customer's order and output is a set of diagrams showing spatial relationships among the components of the order. These diagrams act as guidance to the technician who assembles the system. For example two inter-dependent activities needs to be performed for configuring a VAX mini-computer system:

1. It is determined that the customer's order is complete; if it is not, whatever are the components in shortfall as per the standard order, must be added.
2. Next, the spatial relationships among all of the components is to be determined (including those that are added).

The criterion of success whether a configuration is complete or not, cannot be determined by any simple test, but involves instead particular knowledge about all the individual components and their relationships. The criterion of successful spatial arrangement is that it should be geometrically compact, and in addition, it involves particular knowledge on a component by component basis. Thus, the task accomplishment is defined by a large set of constraints comprising a large amount of knowledge.

4.7 Other Systems of Reasoning

In addition to the methods discussed above, there exists reasoning methods, like, *model based reasoning*, *case based reasoning*, and some *hybrid* combinations of these.

4.7.1 Model-Based Systems

Consider a situation of applying the goal driven strategy of trouble-shooting electronic circuits, like, TV, or electron-microscope. On confirming that the system does not function (i.e., given the goal), we try to find out systems modules (i.e., sub-goals) and check which out of these are working. For the unit which is not working, we try to find out its sub units, and carry on the check like this, till we reach to component levels, like resisters, capacitors, etc. In addition, we also need to look at the history of failure rate of the components. This, in fact becomes a very exhaustive work.

Instead, it would be better if we take the physical model of the system, with mathematical equations describing the interactions and relationships between them. It would then base the diagnosis on the signal reading at various points in the circuit.

A *Model based* reasoning comprises following details:

1. A description of each component in the device. These description can simulate the behavior of the component.
2. A description of the device's internal structure: A representation of components and their interconnections, with ability of simulation of interactions.
3. Information about the input output relationships and magnitudes of values at input and outputs.

To have a glimpse of a model based reasoning, consider a model shown in Fig. 4.2. The input and output relations are functionally defined, expected and actual values are shown in the output. Given this, we can determine the faulty component.

This model comprises *adders* and *multiplications* units which deliver the output as per the functions they perform. In the multiplier-2 the actual output is 56, but expected is 63. Hence the possibility is, either the Mult-2 is faulty or its inputs are faulty. Since the actual output of Mult-1 and its expected are matching, that confirms that, Mult-1 and its two inputs are correct. Thus possibility left is, either first input to Mult-2 is wrongly delivered, or add-2 is malfunctioning, or input D is incorrect. Assuming that connections are not faulty, then either Mult-2 is faulty, or Add-3 is faulty, or the input D is faulty, which can be systematically checked to locate the fault.

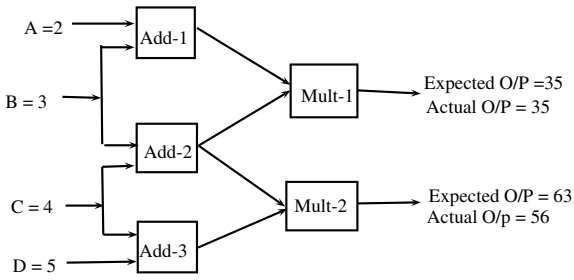

Fig. 4.2 Model to be troubleshooted

4.7.2 Case-Based Reasoning

The case-based reasoning (CBR) uses specific database for problem solving. Consider the case of medical practitioner, who uses medical history of failed as well as successful "cases" for diagnosis of future patients. If a new patient's case appears similar to one of the earlier patient's successful case treatment case, the knowledge history of old case is useful. At the same time if some old case has failed, that will not be tried again due to available history.

Figure 4.3 shows the functional block diagram of case based reasoning. Often, the researchers use the combination of above cases for reasoning due to their advantages.

The similar is the situation in the case of legal practitioners. Lawyers, are allowed by law, to quote the old case histories in their legal proceedings, if that case matches in the present situations. They can also give a different interpretation for the old case to deal with new situations, even if they do not fully match [1].

The above are the examples of *case based reasoning*.

Many other cases can be counted in this category: programmers often reuse the old code to fit in the new situation with partial modification, architects reuse the design of older successful architectures, the wars winners learn from the history of wars won, all of us often follow the success stories of others in life, and so on. Many a times, the case where we need to apply the reasoning is not identical to the previous, hence some transformations are required to fit into the current situation. Thus, this reasoning is a a kind of learning, from the analogy, called *analogical leaning*.

For each new problem, the case based reasoning researchers share the following common operations:

1. Retrieve appropriate cases from the storage (case database).
2. Modify the retrieved case to fit into the current situations.
3. Apply the transferred rule.
4. Save the solution along with the case details, for success or failure for future use.

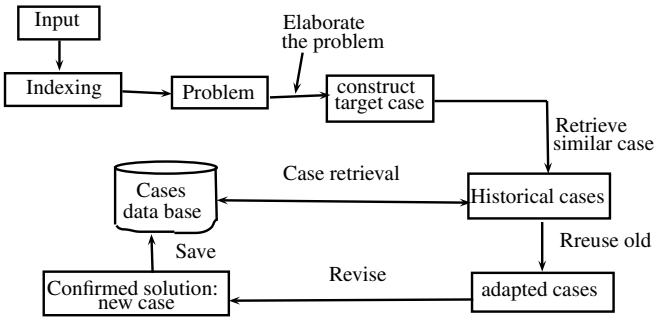

Fig. 4.3 Case-based Reasoning

4.8 Summary

Some of the applications of rule based expert systems are: component selection, equipment maintenance, product configuration, computer operation & troubleshooting, process control, and quality assurance. The basic form of a rule is,

$$if <conditions> then <conclusion>$$

where <conditions> represent the conditions or premises of a rule, and <conclusion> represent its conclusion or consequence.

A Rule Based System (RBS) consists of a knowledge base and an inference engine. In addition to its static memory for facts and rules, an RBS uses a *working-memory* to store temporary assertions. These assertions record earlier rule-based inferences. Some times, the patterns specify the *actions* rather than *assertions*, e.g., "to put them on the table". In such case the rule based system is called *reaction system*.

The rule based systems are two types: *forward chaining* (data driven) and *backward chaining* (goal) driven. In gaol driven system, one starts from the hypothesis (goal) and tries to prove it by finding the subgoals, then these subgoal becomes the goals, and so on. In data driven system, we search from the data and search the goal. In both the cases one needs to perform the rule chaining. While forward chaining allows conclusions to be drawn only from a given knowledge base, a backward-chained rule interpreter is suitable for requesting still unknown facts. The other types of reasoning systems are: Case based system (CBS) and Model based systems (MBS).

When we are doing data-directed reasoning, we may not like to fire all of the rules in case more than one rule is applicable. For this, some kind of conflict resolution is necessary for arriving at the most appropriate rule(s) to fire. Some conflict resolving strategies may be: *Order of listing*, *recency*, *specificity*, *syntactic structures*, etc.

The criteria for reasoning process can be: similarity between rules, applicability of new rules, and Indexing.

Whenever the rules are such that typical set of facts relate to large number of conclusions, i.e., the *fan-out* is larger than *fan-in*, it is better to use backward chaining. On the other hand, when the rules are such that when a typical hypothesis can lead to many facts, i.e., *fan-in* is larger than *fan-out*, it is better to use forward chaining.

The *case based reasoning* uses specific database for problem solving. A medical practitioner, who uses medical history of failed as well as successful cases for diagnosis of future patients. If a new patient's case appears similar to one of the earlier patient's successful case treatment case, the knowledge history of old case is useful. At the same time if some old case has failed, that will not be tried again due to available history.

Exercises

1. List the various components of a Production System, and explain each.
2. In the above context, explain the importance of "binding", "matching" and "conflict resolution".
3. Suppose there is a production system with four initial facts: A, B, C, D and following three rules:

$$R_1 : if\ A\ then\ E$$
$$R_2 : if\ B \wedge F\ then\ G$$
$$R_3 : if\ C \wedge E\ then\ F$$

   a. Using these rules and facts, explain what is meant by "backward chaining" and show explicitly how it can be used to determine the truth of G?
   b. Explain what is meant by "forward chaining", and show explicitly how it can be used in this case to determine new facts.

4. Consider the following possibilities, suggest the solution strategy to be adopted if the system is implemented as a rule based system:

   a. A subgoal literal is generated such that the higher goal is a subset of the subgoal.
   b. A subgoal literal is generated whose negation unifies with the higher-goal.
   c. A subgoal S literal is generated that is equal to another goal G, and G is neither higher nor lower to S.

5. Translate the following knowledge base into predicate form.

   If x is on top of y then y supports x.
   If x is above y and they are touching each other then x is on top of y.
   A phone is above a book.
   A phone is touching a book.

   a. Use forward-chaining to show that the predicate "supports(book, phone)" is true.
   b. Use forward-chaining to show that the predicate "supports(book, phone)" is true.

   Count the number of triggering and rule firing in each of the above cases.

6. Given the propositions A, B, C, D and rules R_1 to R_4,

$$R_1 = if\ A \wedge X \wedge Y\ then\ Z$$
$$R_2 = if\ B \wedge V\ then\ Y$$
$$R_3 = if\ C \wedge V\ then\ X$$
$$R_4 = if\ D\ then\ C$$

a. Use the forward-chaining to determine if Z can be inferred from the above knowledge base.
b. Use the backward-chaining to determine if goal Z succeeds from the above knowledge base.

7. What are the conflict resolving strategies in case more than one rule matches with the assertions? Discuss the merits and demerits of each strategy for selection of a rule to fire.
8. Briefly justify each of the following for conflict resolution. Also, give examples in each case.

a. Specificity criteria.
b. Syntactically longest rule first.
c. Ordering rules.
d. Most recent first.
e. In order of increasing size of the rule.

9. List the following formulas in oder of specificity, and construct a tree such that those having same specificity will stand at the same level in the tree.

a. $b \vee c$
b. $b \wedge c$
c. $a \vee c$
d. $a \wedge c$
e. $a \wedge b$
f. $a \wedge b \wedge c$
g. b
h. c

10. How the assignment of priority to rules can be implemented? Consider the Table 4.2 to demonstrate the assignment of suitable priorities to rules listed in this table. Suggest the structure as how the priorities are to be stored and rules be invoked based on these priorities.
11. Explain what structures can be used and how they will operate, for knowledge based systems, for the following functionalities?

a. If a precondition of one or more rules is false, the system should exclude these rules from participation in the inference process.
b. If firing of some rules infer new knowledge, this should be added into the existing knowledge.
c. To maintain and use index file for rules.

12. For the rule (4.2), if domain size of p is $|p| = m$, domain size of variable x is n, and of y is l,

a. Find out the worst-case time complexity of the rule to be selected,
b. In how many ways ordering can be done?
c. Justify that it is *NP-hard* problem as a general case.

13. Which type of rule chaining you will prefer in the following situations? Also, identify, what is the goal in each case.

    a. To diagnose the case of malaria infection, based on the symptoms of cough, soar throat, regular fever, shivering.
    b. Identify a thief, based on the nature of theft, finger prints, and goods stolen.

14.] Suggest an approach to combine the forward and backward chaining for the knowledge base shown in Table 4.2 (Hint: Try backward chaining to begin with, if it is not doing efficiently, then switch over to forward chaining.)
15. To prove a theorem of geometry using rule-based systems, represent the following statements as production rules:

    a. Corresponding sides of two congruent triangles are also congruent.
    b. Corresponding angles of two congruent triangles are also congruent.
    c. If corresponding sides of two triangles are congruent then the triangles are congruent.
    d. If corresponding sides and the angle covered by them are equal then the triangles are congruent.
    e. Base angles of an isosceles triangle are congruent.

16. In what order the rules should be fired for inference efficiency? Discuss the merits and demerits of choosing a particular order of rule firing.
17. What are the criteria of selection of premises for firing the rules? Discuss this based on specificity and generality of the premises.
18. Consider a network of applicability of rules. Imagine that due to firing of a rule, a new inference is generated. This inference may contradict some fact or goal. Explain, what should be the structure of dependency network so that any inconsistency caused due to new inference is taken care of.
19. Suggest the strategy, as how a tree like rules structure can be used for reasoning in forward direction. Explain or suggest your strategy for following:

    a. Whether the breadth-first or depth-first search is better for rule searching?
    b. Whether the top-down or bottom-up search is better for firing of rule sequences?
    c. How to eliminate one or both the rules, say R_1 and R_2, for consideration, if one of the precondition is false, which is common in both R_1 and R_2?
    d. In the begin of a reasoning process whether you would prefer to choose to fire a rule with large number of preconditions or small?

20. Analyze the Algorithm 4.1 and justify that it is NP-hard as a general case, however, polynomial in real world situation.
21. Analyze the complexity of preconditions in the forward reasoning process.
22. How the following situations are implemented in knowledge base for rule-chaining? Explain.

    a. “Is the pain intensifying with respirations?”

b. "Did shortening of breadth occur before throat pain?"

23. Given the knowledge base of rule-chaining of Table 4.2, show an analysis as, in general, which type, forward or backward search strategy is better? Also, justify your claim.
24. Why, a rule-based system having combination of forward and backward rule-chaining is more complex than either of these both?

References

1. Allen BP (1994) Case-based reasoning: business applications. Commun ACM 37(3):40–42
2. Patterson DW (2001) Introduction to artificial intelligence and expert systems. PHI India
3. Hayes-Roth F (1985) Rule-based systems. Commun ACM 28(9):921–922
4. Puppe F (1993) Systematic introduction to expert systems. Springer-Verlag
5. McDermott JP (1980) R1: A rule-based configurator of computer systems—technical report CMU-CS-80-119, 1–56