1988

Artificial Intelligence through Prolog by Neil C. Rowe

Rowe, Neil C.

Prentice-Hall

Representing uncertainty in rule-based systems

Numbers are important in most areas of engineering and science. In artificial intelligence, one use of numbers is to quantify the *degree* to which we are certain about something, so we can rank our deductions. This lets us model the world more realistically. We'll show how these kinds of numbers can be easily added to rule-based systems in Prolog, and suggest ways--conservative, middle-of-the-road, and liberal--to manipulate them.

Probabilities in rules

Rules in rule-based systems have so far been absolute: when things are absolutely true on the right side of a rule, then the thing on the left side is absolutely true. But in many real-world situations, inferences or even facts are to some degree uncertain or probabilistic. This is particularly true when facts in a rule-based system represent evidence, and rules represent hunches (hypotheses) explaining the evidence. Examples are the diagnosis rules for a car-repair expert system; many diagnoses can't be certain unless the car is taken apart, but suggestive evidence can be found from the way the car behaves.

Mathematicians have for a long time used probabilities to model degrees of uncertainty in the world. A probability is the fraction of the time we expect something will be true. Other numbers are used in artificial intelligence to represent uncertainty, but probabilities came first, so we'll prefer them here.

As we mentioned in Section 2.10, we can add probabilities as an last argument to Prolog facts. So to say a battery in a randomly picked car is dead 3% of the time, we can put the fact

```
battery(dead,0.03).
```

in the Prolog database | REFERENCE 1|. .FS | REFERENCE 1| Some Prolog implementations don't allow real (decimal) numbers. If yours doesn't, you can represent probabilities by the integer closest to a million times the probability. So 200,000 would represent a probability of 0.2. You'll also have to modify the formulas given later in this chapter so the math will work right. Use 1,000,000 wherever 1 occurs in formulas, and divide all products by 1,000,000, and multiply all two-number quotients by 1,000,000. Addition and subtraction don't have to be modified.

Also, many Prolog implementations that do handle decimal numbers require digits both before and after the decimal point, so you have to say "0.03" instead of just ".03". We'll do that in this book. .FE We can modify predicate expressions in rules similarly. For instance, if 20% of the time when the car won't start it is true the battery is dead, we could write:

```
battery(dead,0.2) :- ignition(wont_start,1.0).
```

We can write different rules for inference of the same fact from different sources of evidence, each with its own probability. So if 50% of the time when the radio won't play the battery is dead:

```
battery(dead,0.5) :- radio(wont_play,1.0).
```

So if we want to reason about whether the battery is dead, we should gather all relevant rules and facts. Then somehow we must combine the probabilities from facts and successful rules to get a cumulative probability that the battery is dead. This we call the *or-combination* issue with probabilities, since you can think of rules with the same left-side predicate name as an implicit "or".

A second issue is that rules can be for a different reason than facts. Consider the preceding rule for the likelihood the battery is dead when the ignition won't start. Suppose we are not sure the ignition won't start--we've tried it a few times, and the car didn't seem to respond. (It might respond if we waited an hour to try it, which would be true if the engine is flooded.) What now is our probability that the battery is dead? It must be less than 0.2, because the new conditions worsen the implication, but how much less? It seems the 0.2 must be combined with the ignition-dead probability. We will call this the *rule versus evidence probabilities* issue.

A third issue is that rules can have "and"s of several predicate expressions on their right sides, and if each has a probability, we must somehow combine those numbers--what we call the *and-combination* issue. Note this is quite different from the "or-combination", because weak evidence that any "and"ed expression is satisfied means weak evidence that the whole "and" is satisfied (a chain is as strong as its weakest link).

Probabilities often arise in artificial intelligence applications when reasoning backward. For instance, if the battery of a car is dead, the car will not start; and if there is a short in the electrical system of a car, the car will not start. Those things are absolutely certain. But if a car will not start, then we must reason backward to figure the likelihood that the battery is dead; we can't be completely certain because another cause like a short in the electrical system could also explain the failure to start. But reasoning backward from effects to causes has many important applications, so we must do it if we want computers to behave intelligently.

Some rules with probabilities

To show the flexibility of probabilities in rules, here are some examples. Suppose 6 times out of 10 when the car won't start, the battery is dead. Then:

```
battery(dead,0.6) :- ignition(wont_start,1.0).
```

The 0.6 here is a *conditional probability*, a probability something happens supposing something else happens. But we could also treat the right-side expression as something that couldn't possibly be uncertain, something that is either true or false. Then we could write the rule with a probability only on the left side:

```
battery(dead,0.6) :- ignition(wont_start).
```

Now suppose the car won't start, and we measure the battery voltage with a voltmeter. Suppose we're not skilled at using a voltmeter. The following rule would apply:

```
battery(dead,P) :- voltmeter(battery_terminals,abnormal,P).
```

This says that the battery is dead with the same probability that the voltmeter measurement is outside the normal range. Not being skilled, we might not be measuring the voltage properly (the terminals might be reversed, or the range setting on the voltmeter might be too low or too high, causing us to incorrectly read no voltage). So the uncertainty of the voltmeter measurement is reflected in the conclusion. Note if **P** is 1 (if we are completely sure of our voltmeter measurement) then **P** is 1 for the battery being dead too.

Suppose we want to rewrite the preceding rule to ignore very weak evidence for the conclusion; this will help avoid unnecessary computation on insignificant things. We can put in an arithmetic comparison:

```
battery(dead,P) :- voltmeter(battery_terminals,abnormal,P),
  P > 0.1.
```

This says that if the voltmeter reading is outside the normal range with probability **P**, and **P** is more than 0.1, then the battery is dead with that same probability **P**.

Now consider what to do when the evidence on the right side of a rule can be certain or uncertain, but when it is certain it does not imply certainty of the left side. This can happen when the right side is a conclusion itself. For instance:

```
battery(dead,P) :- electrical_problem(P2), P is P2 * 0.5.
```

This says that half the time when the car has an electrical problem, the battery is dead. Mathematically, it takes the probability of an electrical problem and multiplies it by 0.5 to get the probability of the battery being dead.

Finally, here's an example of evidence combination. If there is an electrical problem and battery is old, then we suspect (with maybe a probability of 0.9, because evidence is stronger than for the preceding rule) that the battery is dead. But suppose we're not sure of either contributing factor, the problem being electrical or that the battery being old. We must somehow decrease the 0.9 by the uncertainty of the factors. One simple way (the probabilistic-independence-assumption method) is to take the product of 0.9 and the probabilities of the two factors, like this:

```
battery(dead,P) :- electrical_problem(P2), age(battery,old,P3),
   P is P2 * P3 * 0.9.
```

So if we believe there's an electrical problem with probability 0.7, and we're 80% sure the battery is old, an estimate of the probability that the battery is dead is |0.7 * 0.8 * 0.9 = 0.504|.

All probabilities are estimates. We'll treat our probabilities as rough indications of certainty, mainly important relative to one another. We won't insist that all our probabilities of a certain kind sum to 1, because rarely can we feel we've covered all possibilities. For instance, a infinity of things can go wrong with your car, and it wouldn't be reasonable to compute the probability that your car won't start because mutant rats have eaten the engine. (We won't even insist that probabilities of a certain kind must sum to no more than 1, because it's hard to analyze the amount of overlap among probabilities.)

Combining evidence assuming statistical independence

The last section shows that combining probabilities in rule-based systems is very important. Surprisingly, there is no fully general mathematical approach to combining; the problem can be proved mathematically intractable. But we can give some formulas that hold under particular assumptions, and most artificial-intelligence work follows this route. Alas, people make frequent mistakes with probabilities, as shown by experiments, so it's a bad idea to look for guidance from the way people reason.

The easiest assumption we could make is that the different forms of evidence are probabilistically independent. That is, occurrence of one kind of evidence does not make another kind any more or less likely. This situation often happens when the evidence comes by very different reasoning methods, and can't "interact". For instance, suppose we are writing an expert system to give advice about stock-market investments. We might have two rules:

1. If a quarterly report on economic indicators today says that interest rates will go up this year, then the stock market index will go down tomorrow with probability 0.7.

2. If the stock market index has gone up for three straight days, it will go down tomorrow with probability 0.4.

These two rules reflect quite different phenomena. So the success of the first rule won't make us believe the second is any more likely to succeed on the same day, and vice versa. (Maybe there is a little bit of influence--a stock market going down a lot might indirectly cause interest rates to go up--but the connection is pretty weak.) That's what we mean by statistical independence.

When statistical independence applies, probability theory says the probability of both of two uncertain events occurring is the product of the probabilities of each event occurring individually. In general, if events |A, B, C, ...| are statistically independent, then in mathematical notation

```
p   left [ A   "and"   B "and"   C   "and"   ... right ]
= p ( A ) p ( B ) p ( C ) ...
```

where |p (A)| means "probability of event A", etc. This formula defines *and-combination* of the probabilities of the events with the assumption of probabilistic independence.

As for "or"s (unions) of events instead of "and"s (intersections), consider the *Venn diagram* in Figure 8-1. Let regions represent events, so areas of regions represent probabilities, and the area of region A is |p(A)|. Then the area of the region representing the "or" of A and B is the area of A plus that area of B, minus the area in common between both A and B (we counted the area in common twice, hence we must subtract it out once). So since areas correspond to probabilities:

```
p ( A   "or"   B ) = p ( A ) + p ( B ) - p ( A   "and"   B)
```

That last formula applies to *any* events. But when the independence assumption holds, we can think of the Venn diagrams as being drawn in a special way, so that events A and B correspond not to circles but to rectangular regions that cross at right angles. See the top diagram in Figure 8-2. Here the area of the whole square (representing the *universe* or all possible events) is 1, and the probabilities of A and B are each proportional to a distance along the side of the square. So the area of the upper left subrectangle is the probability |p (A "and" B)|, and the area of a rectangle is its length times its width, or is |p (A) p (B)|. Hence substituting in the preceding equation, we get the formula for "or-combination" of two probabilities with the independence assumption:

```
p ( A   "or"   B ) = p ( A ) + p ( B ) - p ( A ) p ( B )
```

We can generalize this to the "or" of three events:

```
p ( A   "or"   B   "or"   C ) = p ( A ) + p ( B) + p ( C ) - p ( A   "and"   B)
- p ( A   "and"   C ) - p ( B   "and"   C ) + p ( A   "and"   B   "and"   C )



= p ( A ) + p ( B ) + p ( C ) - p ( A ) p ( B ) - p ( A ) p ( C )
- p ( B ) p ( C ) + p ( A ) p ( B ) p ( C )
```

Using mathematics we can prove the general formula for the "or-combination" of a set of probabilities assuming probabilistic independence:

```
p   left [ A   "or"   B   "or"   C   "or"   ... right ] =
1 - [ ( 1 - p ( A ) ) ( 1 - p ( B ) ) ( 1 - p ( C ) ) ... ]
```

Prolog implementation of independence-assumption "and-combination"

We can define a predicate that implements the preceding independence-assumption "and-combination" formula. It will take two arguments: an input list of probabilities, and an output number for the combined probability.

```
indep_andcombine([P],P).
indep_andcombine([P|PL],Ptotal) :- indep_andcombine(PL,P2), Ptotal is P2 * P.
```

We just call this predicate as the last thing on the right side of rules, to combine the "and"ed probabilities in a rule. If we had a rule without probabilities like this:

```
f :- a, b, c.
```

we would turn it into a rule with probabilities like this:

```
f(P) :- a(P1), b(P2), c(P3), indep_andcombine([P1,P2,P3],P).
```

That addresses the third issue discussed in Section 8.1. Interestingly, **indep_andcombine** can also address the second issue discussed in Section 8.1, that of modeling rule strengths. Suppose we have a rule:

```
g(P) :- d(P1), e(P2), indep_andcombine([P1,P2],P).
```

The **indep_andcombine** handles uncertainty of **d** and **e**, but the rule itself may be uncertain, meaning that the conclusion **g** has a probability less than 1 even when P1 and P2 are both 1. We could characterize this rule uncertainty itself with a probability, the probability that the rule succeeds given complete certainty of all terms "and"ed on its right side. If this probability were 0.7 for instance, we could rewrite it:

```
g(P) :- d(P1), e(P2), indep_andcombine([P1,P2,0.7],P).
```

In other words, rule uncertainty can be thought of as a "hidden" "and"ed predicate expression) with an associated probability.

Here's an example. Suppose we have the following rule and facts:

```
f(P) :- a(P1), b, c(P2), indep_andcombine([P1,P2,0.8],P).
a(0.7).
b.
c(0.95).
```

Then for the query

```
?- f(X).
```

P1 will be bound to 0.7, and **P2** to 0.95. Predicate **indep_andcombine** computes |0.7 * 0.95 * 0.8 = 0.537|, and **P** is bound to that; so that's **X**, the total probability of predicate **f**.

For rules that refer only to things absolutely true and false, "and-combination" is unnecessary. The rule-strength probability need only be on the left side, as for instance:

```
f(0.7) :- a, b, c.
```

Prolog implementation of independence-assumption "or-combination"

The remaining issue discussed in Section 8.1 was "or-combination" of probabilities. Independence-assumption "or-combination" can be defined analogously to independence-assumption "and-combination" but using the last formula in Section 8.3:

```
indep_orcombine([P],P).
indep_orcombine([P|PL],Ptotal) :- indep_orcombine(PL,P2),
 Ptotal is 1 - ( (1-P) * (1-P2) ).
```

"Or-combination" is needed when we have multiple evidence for the truth of predicate **f**, as for instance:

```
f(0.5) :- a.
f(0.7) :- b, c.
f(0.8) :- d, not(e).
```

"Or-combination" is more awkward than "and-combination" because we must use a new predicate name to represent the combination | REFERENCE 2|. So for the preceding example we must define something like **f_overall** that represents the cumulative likelihood of **f**. This can use a special built-in predicate called **bagof**, used this way:

```
f_overall(P) :- bagof(P2,f(P2),PL), indep_orcombine(PL,P).
```

We'll explain this **bagof** predicate more formally in Section 10.6. For now we'll note that **bagof** has three arguments: a variable (an input), a predicate expression containing that variable (an input), and a list of all values to which the variable in the expression can successfully be bound (an output). So the **bagof** says to make a list PL of all the P2 such that **f(P2)** succeeds. If the argument to **f** represents the probability of event **f**, the list PL will contain all the probabilities found for the predicate **f** by every possible rules and fact. These can be combined with the **indep_orcombine** | REFERENCE 2|. .FS | REFERENCE 2| An alternative simpler implementation of indep_orcombine is possible in many dialects of Prolog with the "univ" feature (see Section 7.14) that converts from lists to predicate expressions, symbolized by "**=..**", where the left side is an expression and the right side is its component list of symbols. It's used like this: **new_indep_orcombine(F,P) :- Pred =.. [F,P2], bagof(P2,Pred,PL), indep_orcombine(PL,P).** .FE

As an example, suppose we have these rules and facts:

```
g(0.7) :- a.
g(P) :- b(P1), c(P2), indep_andcombine([P1,P2,0.9],P).
g(0.3) :- a, d.
a.
b(0.8).
c(0.9).
d.
```

Then to combine evidence for the **g** predicate we need:

```
total_g(P) :- bagof(P2,g(P2),PL), indep_orcombine(PL,P).
```

and to use it we must query:

```
?- total_g(X).
```

All three **g** rules will succeed with these facts, and **P** in the second **g** rule will be bound to |0.8 * 0.9 * 0.9 = 0.648|. So **PL** in the **total_g** rule will be bound to **[0.7,0.648,0.3]**. Now we must call on **indep_orcombine**. |1 - ((1 - 0.648) * (1 - 0.3)) = 1 - 0.352 * 0.7 = 0.7536|, and |1 - ((1 - 0.7) * (1 - 0.7536)) = 1 - 0.3 * 0.2464 = 0.92608|. Hence the argument **P** to **total_g** will be bound to 0.92608, and hence to **X**, the total probability that **g** is true.

The conservative approach

Independence when combining probabilities is a strong assumption. It does not hold when one event causes another, or when two events are both caused by some other event. For instance with a small appliance, a totally nonfunctioning device suggests an electrical problem, and a frayed cord suggests an electrical problem. This could be represented as:

```
electrical_problem(0.5) :- doesnt_work.
electrical_problem(0.6) :- frayed_cord.
```

If both rules succeed, the independence-assumption probability of an electrical problem is 0.8. But that's too high, because the cord problem could explain the not-working observation: the cord being frayed could cause the device not to work. So the true combined probability should be closer to 0.6, the number in the second rule.

One approach when independence does not hold is to be very conservative, very careful not to overestimate the cumulative probability. This *conservative* approach is sometimes called a *fuzzy set* approach, actually it is more general than what is called "fuzzy set theory", representing a mathematically provable lower bound. Consider a "conservative orcombine". Whatever the total probability for some event with multiple positive evidence, it must be no worse than the probability of the strongest evidence: positive evidence shouldn't ever disconfirm other positive evidence. So we could define a "conservative orcombine" to operate on probability lists in place of **indep_orcombine**, to use whenever independence clearly doesn't hold between the probabilities:

```
con_orcombine(PL,P) :- max(PL,P).
max([P],P).
max([P|PL],P) :- max(PL,P2), P > P2.
max([P|PL],P2) :- max(PL,P2), not(P > P2).
```

The **max** definition is from Section 5.5. The middle diagram of Figure 8-2 is the Venn diagram for the "conservative-or" case.

For a corresponding "conservative andcombine", we could just give "0"--that's always plenty conservative. But if the "and"ed probabilities are all large, we can prove a nonzero value. Consider the two probabilities 0.7 and 0.8 for two events. 30% of the time the first event must be false. That 80% for the second event can't all "fit" into 30%; only part of it can, with another 50% left over. So at least 50% of the time both events must occur; i.e., the minimum probability is 0.5.

In general, the conservative value for |p (A "and" B) | is

```
maxfunction ( p ( A ) + p ( B ) - 1 , 0 )
```

where "maxfunction" is a mathematical function having a value (not a predicate like the previous **max**), and its value is the larger of its two arguments. We will define:

> *The value of maxfunction(X,Y) is the larger of the two numbers X and Y;*
> The value of minfunction(X,Y) is the smaller of the two numbers X and Y.

To generalize the "and" formula to any number of "and"ed expressions, we can define:

```
con_andcombine([P],P).
con_andcombine([P|PL],0) :- con_andcombine(PL,P2), P + P2 < 1.0.
con_andcombine([P|PL],Ptotal) :- con_andcombine(PL,P2),
  P3 is P+P2, not(P3<1.0), Ptotal is P + P2 - 1.0.
```

The bottom diagram in Figure 8-2 is the Venn diagram for the **conservative_and** case. So the diagram for conservative "or" is different from the diagram for conservative "and". In general, you should use the conservative "or" when there are strong positive correlations between evidence, and the conservative "and" when there are strong negative correlations.

To illustrate, let's use the same example of the last section but substitute in **con_orcombine** and **con_andcombine**:

```
total_g(P) :- bagof(P2,g(P2),PL), con_orcombine(PL,P).
g(0.7) :- a.
g(P) :- b(P1), c(P2), con_andcombine([P1,P2,0.9],P).
g(0.3) :- a, d.
a.
b(0.8).
c(0.9).
d.
```

The second **g** rule binds its **P** to |0.8 + (0.9 + 0.9 - 1) - 1 = 0.6|. Then PL is bound to **[0.7,0.6,0.3]**, and **P** in **total_g** is bound to 0.7.

Note we don't have to pick the independence-assumption or conservative approach exclusively in a rule-based system. We can choose one in each situation based on our analysis of appropriateness.

The conservative "or-combination" is particularly useful for handling prior (or *a priori*) probabilities. These are "starting" probabilities for the likelihood of an event on general grounds. They are usually numbers close to 0, and they can be expressed in the Prolog database as *facts* (instead of rules) with probabilities. For instance, checking under the hood of a car for frayed wires is a lot of work, so an expert system for auto diagnosis might first try to diagnose an electrical problem without asking for such an inspection, using an "a priori" probability of 0.01 of any wire being frayed. The independence assumption is a poor idea for combining a priori probabilities because they represent a summary of many sources of evidence.

The liberal approach and others

There's also a *liberal* approach: compute the maximum probability consistent with the evidence. This isn't as useful as the "conservative" approach, but applies for "and"s whenever one piece of evidence implies all the others, and applies for "or"s whenever pieces of evidence are disjoint (i.e., prove the same conclusion is ways that cannot hold simultaneously). Liberal formulas can be derived from the conservative formulas by relating "and" and "or", as with the formula for two variables:

```
p ( A   "or"   B ) = p ( A ) + p ( B ) - p ( A   "and"   B)
```

which can also be written

```
p ( A   "and"   B ) = p ( A ) + p ( B ) - p ( A   "or"   B)
```

Since |p (A)| and |p (B)| are known, the maximum value of the first formula occurs when |p (A "and" B)| has a minimum, and the maximum of the second formula occurs when |p (A "or" B)| has a minimum (note the important minus signs). These needed minima are given the conservative formulas. So the liberal bound on two-argument "or"s is

```
p ( A ) + p ( B ) - maxfunction ( p ( A ) + p ( B ) - 1 , 0 )

= - maxfunction ( -1 , - p ( A ) - p ( B ) ) = minfunction ( 1 , p ( A ) + p ( B ) )
```

and the liberal bound on "and"s is

```
p ( A ) + p ( B ) - maxfunction ( p ( A ) , p ( B ) )

= - maxfunction ( - p ( B ) , - p ( A ) ) = minfunction ( p ( A ) , p ( B ) )
```

To generalize, the liberal approach for "and-combination" is the minimum of the probabilities:

```
lib_andcombine(PL,P) :- min(PL,P).
min([X],X).
min([X|L],X) :- min(L,X2), X < X2.
min([X|L],X2) :- min(L,X2), not(X < X2).
```

(The **min** is just like the **max** of Sections 5.5 and 8.6.) Similarly, the general "or-combination" is the sum of the probabilities, provided this number is not greater than 1:

```
lib_orcombine(PL,1.0) :- sumup(PL,P), P > 1.0.
lib_orcombine(PL,P) :- sumup(PL,P), not(P > 1.0).
sumup([P],P).
sumup([P|PL],Ptotal) :- sumup(PL,P2), Ptotal is P + P2.
```

The middle diagram in Figure 8-2 shows the liberal "and" case graphically, and the bottom diagram in Figure 8-2 illustrates the liberal "or". That is, they're the situations for the conservative "and" and "or" reversed. In general, you should use the liberal "and" when there are strong positive correlations between evidence, and the liberal "or" when there are strong negative correlations--just the opposite of the advice for the conservative formulas.

Consider the same example we have used before, but with **lib_orcombine** and **lib_andcombine**:

```
total_g(P) :- bagof(P2,g(P2),PL), lib_orcombine(PL,P).
g(0.7) :- a.
g(P) :- b(P1), c(P2), lib_andcombine([P1,P2,0.9],P).
g(0.3) :- a, d.
a.
b(0.8).
c(0.9).
```

```
d.
```

Now **P** in the second **g** rule will be bound to 0.8, the minimum of **[0.8,0.9,0.9]**. Then **PL** is bound to **[0.7,0.8,0.3]**, and **P** in **total_g** is bound to 1.0.

Again, we can use the liberal approach wherever we like in a rule-based system, and the conservative and independence-assumption approaches elsewhere in the same system too. The formulas are summarized in Figure 8-3. (All the formulas are associative, so we can get the n-item formula from the two-item formula.) If we aren't sure any is best, we can take a weighted average. Or we could invent our own formula. But we must be careful, because not all formulas make sense. Reasonable criteria are (1) smoothness, that the formula never makes abrupt jumps in value as input probabilities smoothly vary; (2) consistency, that it never gives values outside the range between the conservative and liberal values; (3) commutativity, that the order of binary combination doesn't matter; and (4) associativity, that the formula gives the same result no matter how the expressions are grouped (for example, combining |p sub 1| with the combination of |p sub 2| and |p sub 3| must be the same as combining the combination of |p sub 1| and |p sub 2| with |p sub 3|).

Negation and probabilities

Conditions in rules that something *not* be true cause problems for probabilities. For instance:

```
a :- b, not(c).
```

If a, b, and c are all uncertain and the rule itself has a certainty of 0.7, it won't do to just add extra arguments like this:

```
a(P) :- b(P1), not(c(P2)), indep_andcombine([P1,P2,0.7],P).
```

because if there's any evidence for c, no matter how weak, the rule will fail. We want instead for weak evidence for c to *decrease* the probability of a in a small way. The way to handle this is to note the probability of something being false is one minus the probability of it being true. So instead:

```
a(P) :- b(P1), c(P2), inverse(P2,NegP2), indep_andcombine([P1,NegP2,0.7],P).
```

where **inverse** is defined as:

```
inverse(X,IX) :- IX is 1 - X.
```

So we shouldn't use **not**s when probabilities are involved, but use this predicate **inverse** on the resulting probabilities. (But you must still be careful to remember that **p(0.0)** won't match **not(p)**.)

Some artificial-intelligence systems don't follow this approach, however. They try to be more general by reasoning separately about events and their negations. They collect evidence for an event, and combine it with probability combination rules, but they also collect evidence against an event and combine it separately. Then they combine these two cumulative probabilities somehow to get an overall likelihood measure. A simple way used in many expert systems is to take the difference of the probability for something and the probability against something. This number ranges from 1.0 (complete certainty of truth) through 0.0 (complete indecision) to -1.0 (complete certainty of falsity).

An example: fixing televisions

Now we'll give an example of a simple rule-based expert system using probabilities, for diagnosis of malfunctioning equipment. Unfortunately, most expert systems (like most artificial intelligence programs) must be big to do anything worthwhile; otherwise, human beings could do the job fine without them. So to avoid burdening you with an example ten pages long, we must pick something simple and not very useful. So here's an example of the few things wrong with a television set that you can fix yourself (television sets use high voltages so most malfunctions should be treated by trained service personnel.)

When a television set is working improperly, one of two things may be improperly adjusted: the controls (knobs and switches) or the receiver (antenna or cable). So let's write an expert system to estimate probabilities of those two things. We'll assume these probabilities need not be very accurate, but their relative sizes provide a rough guide to where things are wrong.

Consider why the knobs might be adjusted wrong on a television. If it is old and requires frequent adjustment, that could be a reason. Similarly, if kids use your set, and they play with the knobs, that could be a reason too, as well as anything strange you've done lately that required adjustment of the knobs (like taking a photograph of the television picture, requiring that the brightness be turned up very high). Let's write the rules. If a set requires frequent readjusting, then it's quite reasonable that the set is maladjusted today--let's say 50% sure:

```
maladjusted(0.5) :- askif(frequent_adjustments_needed).
```

(The **askif** predicate was defined in Section 7.3; it types out a question for the user, and checks if the response is positive or negative.) For recent unusual usage of the television, it matters how you define "unusual". But let's say 50% to be reasonable, so the rule is:

```
maladjusted(0.5) :- askif(recent_unusual_usage).
```

The rule for children must insist both that children hang around your house and that they would be inclined to mess around with the knobs on your television. That's the "and" of two conditions, so we need an "andcombine". When both conditions hold, it's quite likely the set is maladjusted, so we can give this rule a rule strength of 0.9. So the rule is:

```
maladjusted(P) :- askif(children_present(P1)),
  askif(children_twiddle_knobs(P2)), andcombine([P1,P2,0.9],P).
```

Then to get the cumulative probability the set was recently adjusted, we need an orcombine:

```
recently_adjusted(P) :- bagof(P2,maladjusted(P2),PL), orcombine(PL,P).
```

This last predicate just summarizes predisposing evidence for a set maladjustment, but it doesn't incorporate the best evidence at all, observations of the television set. In other words, two major factors, past and present, must be combined. This could be either an "andcombine" or an "orcombine", but "andcombine" seems preferable because neither factor here implies strongly the diagnosis; it's only when both occur together that evidence is strong. That suggests:

```
diagnosis('knobs on set require adjustment',P) :-
  recently_adjusted(P2), askif(operation(abnormal)),
  andcombine([P2,0.8],P).
```

(Remember, single quotation marks (') indicate character strings in Prolog; everything between two single quotation marks, including spaces, is treated as a unit.)

If there is some uncertainty about whether the television's behavior is normal, we could include a probability as a second argument to the **operation** predicate, combining it in the "andcombine" too. Or we could characterize the operation by many different words. For instance:

```
diagnosis('knobs on set require adjustment',P) :-
  recently_adjusted(P2), askif(operation(mediocre)),
  andcombine([P2,0.5],P).
```

So we have a three-level expert system: an "and" of two expressions, one of which is an "or" of three expressions, one of which in turn is an "and" of two expressions. It's true we could simplify this into two levels by the laws of logic (see Appendix A), rewriting everything in disjunctive normal form or conjunctive normal form (see Appendix A), but this isn't a good idea with rule-based systems. For one thing, extra levels let you group related concepts together to make the rule-based system easier to understand; in a normal form, widely different predicates can be thrown together. Grouping related terms together also enables easier determination of probability values and easier choice of probability combination methods; it's difficult to pick a good combination method for twenty terms, since some of the "and"ed expressions must be considerably more related than others. Many-level rule-based systems also allow easier design and debugging, because they give lots of places to put checkpoints and tracing facilities.

Now let's turn to the other kind of television diagnosis we can make, that the antenna or cable connection to the television set is faulty. For this conclusion we will use the same **diagnosis**, and **operation** predicates as before. But we'll need a new predicate to summarize contributing factors to the faultiness of the antenna or cable connection, **overall_source_problems**:

```
source_problems(0.8) :- askif(television(new)).
source_problems(0.95) :- askif(antenna(new)).
source_problems(0.95) :- askif(cable(new)).
source_problems(0.3) :- askif(recent_furniture_rearrangement).
overall_source_problems(P) :- bagof(P2,source_problems(P2),PL),
  orcombine(PL,P).
```

Let's assume that no one has both an antenna and a cable connection (or if they do, only one of them is operating). We'll use a **source_type** predicate to indicate whether the set up is an antenna or a cable. Then we have two rules for the two diagnoses:

```
diagnosis('antenna connection is faulty',P) :-
  askif(source_type(antenna)), askif(operation(abnormal)),
  overall_source_problems(P).
diagnosis('cable connection is faulty',P) :-
  askif(source_type(cable)), askif(operation(abnormal)),
  overall_source_problems(P).
```

So that's our simple expert system. To use it, we query

```
?- diagnosis(D,P).
```

And each answer that the Prolog interpreter finds to the query will bind **D** to a string representing a diagnosis, and bind **P** to the corresponding probability. To find all diagnoses with nonzero probabilities, we can repeatedly type semicolons.

Graphical representation of probabilities in rule-based systems

The logic-gate representation of an and-or-not lattice (see Section 6.10) is a useful graphical notation for simple rule-based systems. It can be used for rules with probabilities too. Associate every "andcombine" and "orcombine" with a logic gate in the representation. Indicate rule strengths next to their corresponding logic gates. For rules with only one expression on their right side, use special triangle gates (*attenuators*) with one input and one output. Then each line has an associated probability, computed by proceeding from the inputs through the network, applying the proper formula at each gate. Figure 8-4 shows the and-or-not lattice for the example of the last section.

Getting probabilities from statistics

There's a more fundamental problem with probabilities than combining them, however: getting them in the first place. If probabilities are markedly incorrect, reasoning based on them can't be trusted. But getting good probabilities is often the hardest problem in building a rule-based system. Even when programmers can easily decide what the predicates should be, what things rules should cover, and how rules should be structured, they often have trouble estimating probabilities because it's hard to tell when an estimate is wrong. Two approaches are used: getting probabilities from statistics on data, and getting probabilities from human "experts".

Since people reason poorly about uncertainty, the first approach seems preferable. Often we have a lot of routinely-collected data about the phenomena in a rule-based system, as when our rule-based system further automates human capabilities already partly automated. We can approximate needed probabilities by frequency ratios in the data. For instance, we can approximate rule-strength probabilities by the ratio of the number of times the left side of a rule was satisfied to the number of times the right side was satisfied.

As an example, consider the repair of airplanes. They are expensive, so an organization that owns many of them must keep detailed repair and maintenance records to ensure quality work, to better allocate resources, and to track down trends in malfunctions. Most organizations today computerize these, recording observed malfunctions, their eventually inferred causes, and what was done to fix them--just what we need to assign probabilities to an expert system for diagnosis. For instance, we can count how many times the radar system failed, and how many of those times the widget component was faulty, and take the ratio of the two counts to fill in **<strength>** in the rule:

```
faulty(widget,<strength>) :- failed(radar).
```

However, approximating a real number (a probability) by the ratio of two integers can be hard, and the approximation is often poor when the integers are small. Suppose some event happens with probability 0.001, and we have data for 2000 occurrences. On the average we'll expect two events in those 2000, but the number could be 1, 0, 3, or 4 too, since the event is so rare. According to probability theory, random sets of size N, drawn from an infinite population with fraction F of its members possessing some property, will tend to show the same fraction F with standard error (standard deviation of this fraction) approximately

```
sqrt { F ( 1 - F ) / N }
```

(using the binomial-distribution approximation). This says how good a probability estimate is; the larger this number, the worse the estimate. As a rule of thumb, if the preceding is the same or larger than F, the F fraction should not be trusted.

As an example, suppose something happens 7 times out of 20 possible times. Then |N = 20|, |F = 0.35|, and the standard error by the formula is 0.105. This is significantly less than 0.35, so the probability estimate 0.35 looks OK.

Probabilities derived from others

If a probability is important to our rule-based system, yet the associated standard error of approximation from data is large, we may be able to better estimate the probability from other data. One way is Bayes's Rule. Letting |p (A given B)| represent the probability of event A happening when event B also happens, we can say:

```
p ( A   given   B ) = { p ( A   "and"   B ) }   /   {  p ( B ) }
```

But switching A and B in that equation:

```
p ( B   given   A ) = { p ( A   "and"   B ) }   /   {  p ( A ) }
```

We can solve the second equation for |p (A "and" B)|, and substitute in the first equation, obtaining the usual form of Bayes's Rule:

```
p ( A   given   B ) = { p ( B   given   A ) * p ( A ) }   /   { p ( B ) }
```

This is a useful when we have a rule

```
a(P) :- b(P2), andcombine([P2,<strength>],P).
```

and we want to know what number to put for **<strength>**, the probability that **a** is true given that **b** is true. If we have enough data to compute the reverse--the probability that **b** is true given that **a** is true--then Bayes's Rule can use that number, together with estimates of the overall probabilities of **a** and **b**, to give what we need. This is an especially good idea when **a** *causes* **b**, because then the reverse probability that **b** is true given that **a** is true must be 1.0. For instance, a car absolutely will not start when its battery is dead, so we can approximate the probability that the battery is dead when the car won't start by the ratio of: the overall probability the battery is dead over the overall probability the car won't start.

There are extensions of Bayes's Rule, for instance:

```
p ( A   given   ( B   "and"   C ) ) = { p ( ( B   "and"   C )   given   A ) * p ( A ) }
  /    { p ( B   "and"   C ) }
```

Another trick to get rule probabilities is to use the independence assumption in a special way. Suppose we have:

```
a(P) :- b(P1), c(P2), andcombine([P1,P2,<strength>],P).
```

Now there may be few situations in the data in which both **b** and **c** were true. But if there were many situations in which one of **b** and **c** was true, then we could estimate the probabilities of **a** given **b**, and **a** given **c**, and take the "or-combination" of these two numbers (yes, "or" not "and"; think about it) as an estimate of the rule strength needed. That is, we can precompile an "orcombine".

Subjective probabilities

Even with these tricks we may not have enough data (or perhaps good enough data) to approximate probabilities very well. Then we must guess probabilities ourselves, or preferably, ask a human expert in the task or domain of the rule-based system. This isn't always easy: experts may be hard to find, or their time may be expensive, or they may not understand or feel comfortable with a rule formulation of their knowledge. But there may be no other choice.

As we've said, humans make many mistakes in probability estimation, as demonstrated by psychological experiments. One simple way to make things easier is to let people quantify uncertainty on a different numeric scale than 0.0 to 1.0. For instance, take *degrees of certainty* on a scale 0 to 100, and divide by 100 to get the probability. Better yet, do a nonlinear transformation of the probability scale, for instance with *odds* defined as |p / (1 - p)|. Odds range from 0 to positive infinity, so a probability of 0.9 is odds of 9, a probability of 0.5 is odds of 1, and a probability of 0.1 is odds of 0.111. The logarithm of the odds is also useful; it "flattens out" the curve more, and ranges from minus infinity to plus infinity.

Something that also helps people is speaking of uncertainty nonnumerically. For instance, let them use the terms "certain", "almost certain", "likely", "suggestive", "possible", "not likely", and "impossible". Each term may map to a probability--say 1.00 for "certain", 0.99 for "almost certain", 0.8 for "likely", 0.5 for "suggestive", 0.2 for "possible", 0.05 for "not likely", and 0.0 for "impossible". If this isn't possible, perhaps different probabilities can be given for different contexts, so a "possible car problem" would be a 0.2 probability, but a "possible nuclear accident" would be 0.001 probability.

Maximum-entropy probabilities (*)

Bayes's Rule extends the utility of both statistics and subjective probability estimates. We can generalize this idea, to accept arbitrary probabilities from the programmer--prior probabilities, conditional probabilities, and joint probabilities--and make "reasonable guess" estimates of others, using some mathematics.

It can be shown mathematically that best guesses (based on certain postulates for guess desirability) are those that maximize entropy, or minimize the *information content*, of probability assignments. This is a mathematical optimization problem, in which we want to maximize

```
sum from {i=1} to m left ( - p ( A sub i ) log ( p ( A sub i ) ) right )
```

for some mutually exclusive set of probabilities |p (A sub i)| that sum to 1, subject to given equality constraints in the form of probabilities already known. Optimization problems like this can be attacked by many methods from operations research, and computer packages are available. But they can take time since they usually iterate.

Sometimes we don't need to iterate to find maximum-entropy probabilities, but we can use algebraic manipulations to get formulas. Here's an example for those of you that know some calculus. Suppose we know the probabilities of two events A and B, |p (A)| and |p (B)|. Suppose we want to find the maximum-entropy probability for |x = p (A "and" B)| (i.e., we want to do "and-combination" in a maximum-entropy way). Then there are four mutually exclusive probabilities involved: |p (A "and" B) , p (A bar "and" B) , p (A and B bar) , p (A bar "and" B bar)|, where we use |A bar| to represent the exact opposite of A, so |p (A bar) = 1 - p (A)|. Then

```
p ( A   "and"   B ) = x   ,    p ( A bar   "and"   B ) = p ( B ) - x   ,
p ( A   "and"   B bar ) = p ( A ) - x   ,

p ( A bar   "and"   B bar ) = 1 - p ( A ) - p ( B ) + x
```

And the preceding summation formula for the entropy is

```
- x log ( x ) - ( p ( B ) - x ) log ( p ( B ) - x ) -
( p ( A ) - x ) log ( p ( A ) - x )
```

```
 - ( 1 - p ( A ) - p ( B ) + x ) log ( 1 - p ( A ) - p ( B ) + x )
```

To find the maximum of this, we take the derivative with respect to x, and set this to zero. Noting that the derivative of |y log (y)| with respect to x is |(1 + log (y)) dy / dx |, we get:

```
- log ( x ) + log ( p ( B ) - x ) + log ( p ( A ) - x )
- log ( 1 - p ( A ) - p ( B ) + x ) = 0
```

```
log [ x ( 1 - p ( A ) - p ( B ) + x ) / ( p ( A ) - x ) ( p ( B ) - x ) ] = 0
```

```
x sup 2 - ( p ( A ) + p ( B ) ) x + p ( A ) p ( B ) =
x sup 2 + ( 1 - p ( A ) - p ( B ) ) x
```

```
x = p ( A ) p ( B)
```

This is just the independence-assumption formula. So the formula we justified intuitively in Section 8.3 has a deeper justification. Formulas for more complicated situations can also be derived with the method.

Consistency (*)

Another problem with subjective probabilities (but also to a lesser extent with data-derived probabilities) is that they can be inconsistent (logically impossible) in a nonobvious way. This often happens when both a priori (unconditional) and conditional probabilities are specified by people. We should therefore run checks on user-given probabilities before entering them into a rule-based system.

As an example, note from the definition of conditional probability that

```
0 <= p ( A   given   B ) p ( B ) = p ( B   given   A ) p ( A ) <= 1
```

Hence

```
p ( A ) >= p ( A   given   B ) p ( B )
```

```
p ( B   given   A ) >= p ( A   given   B ) p ( B )
```

```
p ( B ) >= p ( B   given   A ) p ( A )
```

```
p ( A   given   B ) >= p ( B   given   A ) p ( A )
```

so we can catch some inconsistencies from inequalities.

Keywords:

probability
uncertainty
or-combination
and-combination
rule probability
conclusion probability
independence assumption
conservative assumption
liberal assumption
Bayes's rule
scale transformations
maximum-entropy estimates

.SH Exercises

8-1. Assume:

1. The battery is defective with certainty 0.5 when a car won't start.
2. The battery is defective with certainty 0.8 when the radio is functioning and the radio won't play.
3. You are not sure if your radio is functioning--the probability is 0.9 that it is functioning.
4. This morning your car won't start and the radio won't play.

What is the cumulative probability that your battery is defective this morning? Combine evidence assuming independence of probabilities.

8-2. Consider these rules (the arguments are all probabilities):

```
a(P) :- b(P2), P is P2 * 0.6.
a(P) :- c(P).
```

Suppose **b** is known to be absolutely certain, and **c** is 80 percent certain.

(a) What is the cumulative probability of **a** using the independence assumption?

(b) What is the cumulative probability of **a** using the conservative assumption?

(c) What is the cumulative probability of **a** using the liberal assumption?

8-3. (R,A) Suppose we want to fill in the **<prob1>** and **<prob2>** probability values in the following two rules that infer a flat tire on a car:

```
flat_tire(<prob1>) :- car_steers_strangely.
flat_tire(<prob2>) :- just_ran_over_something.
```

(a) Suppose we have statistics that say:

-- In 200 situations in which the car steered strangely the tire was flat, out of 500 situations in which the car steered strangely;

-- In 800 situations in which you just ran over something the tire was discovered to be flat, out of 1600 situations in which you just ran over something;

-- A flat tire was observed in 1200 situations total.

Estimate **<prob1>** and **<prob2>** for these statistics.

(b) Suppose we also know that 70 times in which both the car steered strangely and you just ran over something the tire was then found to be flat, out of 101 times in which both those two things were observed. Which probability combination method (or-combination) for the preceding two rules is best confirmed here: conservative, independence-assumption, or liberal?

8-4. (A) Suppose you want to handle or-combination of uncertainties nonnumerically. Suppose the possible degrees of uncertainty are "definitely", "probably", "probably not", and "definitely not".

(a) Suppose the "orcombine" function of any two of those four terms is defined by the table in Figure 8-5. Each row and column represent a pair of values to be combined. Which of the numerical or-combination methods is this equivalent to: conservative, independence-assumption, liberal, or something else?

(b) Suppose the "orcombine" function of any two of those four terms is defined by the table in Figure 8-6. Which of the numerical or-combination methods is this equivalent to: conservative, independence-assumption, liberal, or something else?

(c) For the method of part (b), suppose you want to map "definitely", "probably", "probably not", and "definitely not" into probabilities. It makes sense to have "definitely" = 1.0 and "definitely not" = 0.0. Give probabilities for "probably" and "probably not" consistent with the table in part (b). (There are an infinite number of answers.)

8-5. (A) Combination methods for probabilities use the Prolog **is**, which requires its arithmetic calculation to refer to only bound variables. Consider a rule-based system that uses rules with probabilities and calculates on those probabilities. Does the directionality of **is** mean that one of either backward chaining or forward chaining is impossible? Why?

8-6. (R,A) Consider the following way of doing or-combination of probabilities in an expert system: the cumulative probability is the fraction of the contributing probabilities that are greater than 0.5. So for instance the cumulative probability for contributing probabilities 0.3, 0.9, 0.66, 0.2, and 0.8 would be 0.6.

(a) Define a Prolog predicate **new_orcombine(PL,P)** that computes the cumulative probability **P** of a list **PL** using this method.

(b) Give two major disadvantages of this method, disadvantages not shared by the three methods discussed in the chapter, and explain why they are major.

8-7. Suppose we have R rules concluding D diagnoses such that there are the same number of rules concluding each diagnosis, |R / D|. Assume no intermediate predicates; have each diagnosis rules refer to facts. Suppose the probability of any rule succeeding in a random situation is P, and suppose this probability is independent of the success or failure of other rules.

(a) How many rules will succeed for a situation on the average?

(b) How many diagnoses will succeed for a situation on the average?

8-8. (G) Write 10 or so Prolog rules to predict what the weather will be at 3 P.M. some day, reasoning at noon that day, using probabilities as an additional argument to all predicates that have uncertainty. All rules should have a left side of the form **predict(<weather>,<probability>)**, where **<weather>** is either **sunny**, **partly_cloudy**, or **cloudy**. Choose a good evidence-combination method.

Assume the following predicates are available as the basis for your reasoning at noon. (Try to define some intermediate predicates based on these, which can then be combined to make predictions, to make things more interesting.)

> **current_west_view(<weather>)**: whether the given weather is viewable from a west-facing window right now. The argument can be **sunny**, **partly_cloudy**, or **cloudy**.
>
> **current_east_view(<weather>)**: same for east-facing window.
>
> **raining(<degree>)**: whether it is raining to that degree right now. The **<degree>** can be **light**, **steady_heavy**, and **cloudburst**.
>
> **weatherman_prediction(<weather>)**: whether the TV weatherman predicted that weather for 6 P.M. last night at 11 P.M.
>
> **grandma_prediction(<weather>)**: whether Grandma predicted that weather this morning from how her joints hurt.
>
> **grandma_memory**: whether you remember anything Grandma said that morning.
>
> **radio_prediction(<weather>)**: the weather predicted right now on the local radio news station.
>
> **secretary_has_radio**: you don't have a radio, but the secretary down the hall might.
>
> **secretary_out_to_lunch**: if they are out to lunch, the room is locked, and you can't get in.

8-9. (A,E) Consider the following argument: "Spending money on the lottery must be a good thing, because you constantly hear about people winning big bucks in the battery, in the newspapers and on television."

(a) What is the fallacy (reasoning error) here?

(b) What warning does this illustrate for using probabilities in rule-based systems?

8-10. Most students are adults (18 years old or more). Most adults are employed. We could write:

```
adult(X,P) :- student(X,P2), P is P2 * 0.95.


employed(X,P) :- adult(X,P2), P is P2 * 0.9.
```

Then if we knew that Joe was a student with complete certainty, these rules say that Joe is employed with 0.855 probability, a clearly fallacious conclusion because most students don't work at all, as any professor will tell you. This fallacy is due to a simplification that we have made to make our analysis in the chapter easier.

(a) Show how the problem can be fixed by writing one rule as two.

(b) Suppose we consider this rule rewriting as awkward and we wish to fix things by just changing the probability combination method. The previous method was independence-assumption combination. What happens to the probability of Joe being employed when we use the conservative assumption?

(c) The conservative assumption does not give a reasonable answer either. How can we solve this problem in a reasonably general way?

8-11. (E) (a) In English, double negatives don't always mean what you expect them to mean. Explain why "not unhappy" is different from "happy".

(b) What warning does this suggest to the designer of a rule-based expert system using probabilities? In particular, for what sorts of English words should a designer be careful?

8-12. (P,G) Design a program to diagnose problems with cars. Many artificial intelligence applications involve diagnosis, and automatic diagnostic aids really help. We pick cars because almost everybody knows something about them, and we don't need to hire outside experts as with most expert systems. This diagnosis program should be usable by people who know virtually nothing about cars; it should use nontechnical words and familiar concepts. For instance, it shouldn't expect that a user knows what different sounds mean, and should try to describe sounds by analogies to everyday sounds.

An important part of the project will be the formal definition of concepts that may be concluded (e.g., "the car won't start"). Another part will be enumeration of things a user could be expected to know; in particular, try to use knowledge of the history of the car, what problems it has had in the past. Try not to get too technical; there are plenty of "common-sense" things about cars, particularly for the body and interior of the passenger compartment. For instance, one cause of a rattle in a car could be a baby's rattle under the seat.

Handle uncertain data and uncertain conclusions. Probably the easiest approach is independence-assumption combination. Decide initial values for probabilities, and how to treat evidence against something.

If this is a group project, one person should handle the control structure of the program and provide utilities for everyone else. Other people can specialize in different systems of the car. For instance, someone should probably handle the body and interior of the car, someone the electrical system, someone the engine and fuel system, etc. About thirty rules should be written by each contributor. Emphasize quality of the rules, not quantity.

Go to book index