values. The structure of this computation is shown in Figure 14.8. Using the numbers from Figure 14.2, we obtain \( P(B_j, m) = cs \times 0.00059224 \). The corresponding computation for \( d \) yields \( 0.0014919 \); hence,

\[
P(B_j, m) = cs (0.00059224, 0.0014919) \approx (0.284, 0.716).
\]

That is, the chance of a burglary, given calls from both neighbors, is about 28%.

The evaluation process for the expression in Equation (14.4) is shown as an expression tree in Figure 14.8. The ENUMERATION-ASK algorithm in Figure 14.9 evaluates such trees using depth-first recursion. The algorithm is very similar in structure to the backtracking algorithm for solving CSPs (Figure 6.5) and the DPLL algorithm for satisfiability (Figure 7.17).

The space complexity of ENUMERATION-ASK is only linear in the number of variables: the algorithm sums over the full joint distribution without ever constructing it explicitly. Unfortunately, its time complexity for a network with \( n \) Boolean variables is always \( O(2^n) \)—better than the \( O(n^2) \) for the simple approach described earlier, but still rather grim.

Note that the tree in Figure 14.8 makes explicit the repeated subexpressions evaluated by the algorithm. The products \( P(j I a)P(m I a) \) and \( P(j \neg I a)P(m I \neg a) \) are computed twice, once for each value of \( e \). The next section describes a general method that avoids such wasted computations.

### 14.4.2 The variable elimination algorithm

The enumeration algorithm can be improved substantially by eliminating repeated calculations of the kind illustrated in Figure 14.8. The idea is simple: do the calculation once and save the results for later use. This is a form of dynamic programming. There are several versions of this approach; we present the variable elimination algorithm, which is the simplest.

Variable elimination works by evaluating expressions such as Equation (14.4) in right-to-left order (that is, bottom up in Figure 14.8). Intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable.

Let us illustrate this process for the burglary network. We evaluate the expression

\[
P(B_j, m) = cs \sum P(B) \sum P(e) \sum P(\neg B, e) \sum P(j I a) \sum P(m I a)
\]

Notice that we have annotated each part of the expression with the name of the corresponding factor; each factor is a matrix indexed by the values of its argument variables. For example, the factors \( f_4(A) \) and \( f_5(A) \) corresponding to \( P(j I a) \) and \( P(m I a) \) depend just on \( A \) because \( J \) and \( M \) are fixed by the query. They are therefore two-element vectors:

\[
f_4(A) = \begin{pmatrix} P(j I a) \\ P(j \neg I a) \end{pmatrix} \quad f_5(A) = \begin{pmatrix} P(m I a) \\ P(m \neg I a) \end{pmatrix}
\]

\( f_3(A, B, E) \) will be a \( 2 \times 2 \times 2 \) matrix, which is hard to show on the printed page. (The “first” element is given by \( P(a, 15, e) = 0.95 \) and the "last" by \( P(a, 15, E) = 0.01 \).) In terms of factors, the query expression is written as

\[
P(o, m) = f_1(B) f_2(E) \sum_a f_3(A, E) \times f_4(A) \times f_5(A)
\]
Figure 14.8 The structure of the expression shown in Equation (14.4). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for \( j \) and \( m \).

Figure 14.9 The enumeration algorithm for answering queries on Bayesian networks.