as being composed of two parts: first, the current state distribution is projected forward from \( t \) to \( t + 1 \); then it is updated using the new evidence \( e_{t+1} \). This two-part process emerges quite simply when the formula is rearranged:

\[
P(\mathbf{X}_{t+1} | e_{1:t}, e_{t+1}) = P(\mathbf{X}_{t-1}, e_{t+1}) \quad \text{(dividing up the evidence)}
\]

\[
= P(e_{t+1} | \mathbf{X}_{t+1}, e_{1:t}) P(\mathbf{X}_{t+1}) 0.1 \quad \text{(using Bayes’ rule)}
\]

\[
= P(e_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | e_{1:t}) \quad \text{(by the sensor Markov assumption). (15.4)}
\]

Here and throughout this chapter, \( a \) is a normalizing constant used to make probabilities sum up to 1. The second term, \( P(\mathbf{X}_{t+1} | e_{1:t}) \), represents a one-step prediction of the next state, and the first term updates this with the new evidence; notice that \( P(e_{t+1} | \mathbf{X}_{t+1}) \) is obtainable directly from the sensor model. Now we obtain the one-step prediction for the next state by conditioning on the current state \( \mathbf{X}_t \):

\[
P(\mathbf{X}_{t+1} | e_{1:t+1}) = a P(e_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | e_{1:t})
\]

\[
= a P(e_{t+1} | \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t) \quad \text{(Markov assumption). (15.5)}
\]

Within the summation, the first factor comes from the transition model and the second comes from the current state distribution. Hence, we have the desired recursive formulation. We can think of the filtered estimate \( P(\mathbf{X}_{t+1} | e_{1:t}) \) as a “message” \( \mathbf{f}_{t+1} \) that is propagated forward along the sequence, modified by each transition and updated by each new observation. The process is given by

\[
\mathbf{f}_{t+1} = \alpha \text{FORWARD}(\mathbf{f}_t, e_{t+1}),
\]

where \( \text{FORWARD} \) implements the update described in Equation (15.5) and the process begins with \( \mathbf{f}_0 = P(\mathbf{X}_0) \). When all the state variables are discrete, the time for each update is constant (i.e., independent of \( t \)), and the space required is also constant. (The constants depend, of course, on the size of the state space and the specific type of the temporal model in question.) The time and space requirements for updating must be constant if an agent with limited memory is to keep track of the current state distribution over an unbounded sequence of observations.

Let us illustrate the filtering process for two steps in the basic umbrella example (Figure 15.2). That is, we will compute \( P(R_2 | u_{t=1}) \) as follows:

- On day 0, we have no observations, only the security guard’s prior beliefs; let’s assume that consists of \( P(\mathbf{r}_0) = (0.5, 0.5) \).
- On day 1, the umbrella appears, so \( U_1 = \text{true} \). The prediction from \( t = 0 \) to \( t = 1 \) is

\[
P(R_1) = \sum_{r_0} P(\mathbf{r}_0) P(\mathbf{r}_1 | \mathbf{r}_0)
\]

\[
= (0.7, 0.5) \times 0.5 + (0.5, 0.5) \times 0.5 = (0.5, 0.5).
\]

Then the update step simply multiplies by the probability of the evidence for \( t = 1 \) and normalizes, as shown in Equation (15.4):

\[
P(R_1) \rightarrow P(R_1 | u_{t=1}) = \alpha (0.9, 0.2) (0.5, 0.5) = (0.818, 0.182).
\]
• On day 2, the umbrella appears, so \( u_2 = \text{true} \). The prediction from \( t=1 \) to \( t=2 \) is
\[
P(R_2 \mid u_1) = \sum_{r_1} P(R_1 \mid r_1) P(r_1 \mid t, u_1)
\]
\[
= (0.7, 0.3) \times 0.818 + (0.3, 0.7) \times 0.182 = (0.627, 0.373),
\]
and updating it with the evidence for \( t=2 \) gives
\[
P(R_2 \mid u_1, u_2) = \alpha P(u_2 R_2) P(R_2 \mid u_1) = (0.9, 0.2)(0.627, 0.373)
\]
\[
= (0.565, 0.435) (0.883, 0.117).
\]
Intuitively, the probability of rain increases from day 1 to day 2 because rain persists. Exercise 15.2(a) asks you to investigate this tendency further.

The task of prediction can be seen simply as filtering without the addition of new evidence. In fact, the filtering process already incorporates a one-step prediction, and it is easy to derive the following recursive computation for predicting the state at \( t+k+1 \) from a prediction for \( t+k \)
\[
P(X_{t+k+1} \mid e_1 \ldots e_t) = P(X_{t+k+1} \mid X_t, e_1 \ldots e_t) \cdot (15.6)
\]
Naturally, this computation involves only the transition model and not the sensor model.

It is interesting to consider what happens as we try to predict further and further into the future. As Exercise 15.2(b) shows, the predicted distribution for rain converges to a fixed point \((0.5, 0.5)\), after which it remains constant for all time. This is the stationary distribution of the Markov process defined by the transition model. (See also page 537.) A great deal is known about the properties of such distributions and about the mixing time—roughly, the time taken to reach the fixed point. In practical terms, this dooms to failure any attempt to predict the actual state for a number of steps that is more than a small fraction of the mixing time, unless the stationary distribution itself is strongly peaked in a small area of the state space. The more uncertainty there is in the transition model, the shorter will be the mixing time and the more the future is obscured.

In addition to filtering and prediction, we can use a forward recursion to compute the likelihood of the evidence sequence, \( P(e_1 \ldots e_t) \). This is a useful quantity if we want to compare different temporal models that might have produced the same evidence sequence (e.g., two different models for the persistence of rain). For this recursion, we use a likelihood message \( \ell_{t+1}(X_t) = P(X_t, e_{t+1}). \) It is a simple exercise to show that the message calculation is identical to that for filtering:
\[
\ell_{t+1} \text{FORWARD}(\ell_{t}, e_{t+1}).
\]
Having computed \( \ell_{t} \), we obtain the actual likelihood by summing out \( X_t \):
\[
\text{Liar} = P(e_1 \ldots e_t) = \sum_{X_t} \ell_{t+1}(X_t) \cdot (15.7)
\]
Notice that the likelihood message represents the probabilities of longer and longer evidence sequences as time goes by and \( \alpha \) becomes numerically smaller and smaller, leading to underflow problems with floating-point arithmetic. This is an important problem in practice, but we shall not go into solutions here.