are at most $3^{|Conj(n,k)|}$ distinct sets of component tests. Each of these sets of tests can be in any order, so

$$|k\text{-DL}(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)|!$$

The number of conjunctions of k literals from n attributes is given by

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)$$

Hence, after some work, we obtain

$$|k\text{-DL}(n)| = 2^{O(n^k \log_2(n^k))}$$

We can plug this into Equation (18.2) to show that the number of examples needed for PAC-learning a k-DL function is polynomial in n:

$$m \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)$$

Therefore, any algorithm that returns a consistent decision list will PAC-learn a k-DL function in a reasonable number of examples, for small k. The next task is to find an efficient algorithm that returns a consistent decision list. We will use a greedy algorithm called DECISION-LIST-LEARNING that repeatedly finds a test that agrees exactly with some subset of the training set. Once it finds such a test, it adds it to the decision list under construction and removes the corresponding examples. It then constructs the remainder of the decision list using just the remaining examples. This is repeated until there are no examples left. The algorithm is shown in Figure 18.17.

This algorithm does not specify the method for selecting the next test to add to the decision list. Although the formal results given earlier do not depend on the selection method, it would seem reasonable to prefer small tests that match large sets of uniformly classified examples, so that the overall decision list will be as compact as possible. The simplest strategy is to find the smallest test t that matches any uniformly classified subset, regardless of the size of the subset. Even this approach works quite well. The results for the restaurant data are shown in Figure 18.18, and suggest that learning decision lists is an effective way to make predictions.

function DECISION-LIST-LEARNING(*examples*) **returns** a decision list, *No* or failure

if *examples* is empty **then return** the value *No*
t ← a test that matches a nonempty subset $examples_t$ of *examples*
such that the members of $examples_t$ are all positive or all negative
if there is no such *t* **then return** failure
if the examples in $examples_t$ are positive **then** $o \leftarrow Yes$
else $o \leftarrow No$
return a decision list with initial test *t* and outcome *o*
and remaining elements given by DECISION-LIST-LEARNING($examples - examples_t$)

Figure 18.17 An algorithm for learning decision lists.