variables next, and then all the other variables will fall into place. That's exactly what happens with MRV: NT and SA have two values, so one of them is chosen first, then the other, then Q, NSW, and V in order. Finally T still has three values, and any one of them works. We can view forward checking as an efficient way to incrementally compute the information that the MRV heuristic needs to do its job.

Although forward checking detects many inconsistencies, it does not detect all of them. The problem is that it makes the current variable arc-consistent, but doesn’t look ahead and make all the other variables arc-consistent. For example, consider the third row of Figure 6.7. It shows that when WA is red and Q is green, both NT and SA are forced to be blue. Forward checking does not look far enough ahead to notice that this is an inconsistency: NT and SA are adjacent and so cannot have the same value.

The algorithm called MAC (for Maintaining Arc Consistency (MAC)) detects this inconsistency. After a variable $X_i$ is assigned a value, the INDUCTION procedure calls AC-3, but instead of a queue of all arcs in the CSP, we start with only the arcs $(X_i, X_j)$ for all $X_j$ that are unassigned variables that are neighbors of $X_i$. From there, AC-3 does constraint propagation in the usual way, and if any variable has its domain reduced to the empty set, the call to AC-3 fails and we know to backtrack immediately. We can see that MAC is strictly more powerful than forward checking because forward checking does the same thing as MAC on the initial arcs in MAC’s queue; but unlike MAC, forward checking does not recursively propagate constraints when changes are made to the domains of variables.

### 6.3.3 Intelligent backtracking: Looking backward

The BACKTRACKING-SEARCH algorithm in Figure 6.5 has a very simple policy for what to do when a branch of the search fails: back up to the preceding variable and try a different value for it. This is called chronological backtracking because the most recent decision point is revisited. In this subsection, we consider better possibilities.

Consider what happens when we apply simple backtracking in Figure 6.1 with a fixed variable ordering $Q$, NSW, V, T, SA, WA, NT. Suppose we have generated the partial assignment $\{Q = red, NSW = green, V = blue, T = red\}$. When we try the next variable, SA, we see that every value violates a constraint. We back up to T and try a new color for
Tasmania? Obviously this is silly—recoloring Tasmania cannot possibly resolve the problem with South Australia.

A more intelligent approach to backtracking is to backtrack to a variable that might fix the problem—a variable that was responsible for making one of the possible values of SA impossible. To do this, we will keep track of a set of assignments that are in conflict with some value for SA. The set (in this case \{Q = \text{red}, NSW = \text{green}, V = \text{blue}\}) is called the conflict set for SA. The backjumping method backtracks to the most recent assignment in the conflict set; in this case, backjumping would jump over Tasmania and try a new value for V. This method is easily implemented by a modification to BACKTRACK such that it accumulates the conflict set while checking for a legal value to assign. If no legal value is found, the algorithm should return the most recent element of the conflict set along with the failure indicator.

The sharp-eyed reader will have noticed that forward checking can supply the conflict set with no extra work: whenever forward checking based on an assignment \(X = r\) deletes a value from \(Y\)'s domain, it should add \(X = x\) to \(Y\)'s conflict set. If the last value is deleted from \(Y\)'s domain, then the assignments in the conflict set of \(Y\) are added to the conflict set of \(X\). Then, when we get to \(Y\), we know immediately where to backtrack if needed.

The eagle-eyed reader will have noticed something odd: backjumping occurs when every value in a domain is in conflict with the current assignment; but forward checking detects this event and prevents the search from ever reaching such a node? In fact, it can be shown that every branch pruned by backjumping is also pruned by forward checking. Hence, simple backjumping is redundant in a forward-checking search or, indeed, in a search that uses stronger consistency checking, such as MAC.

Despite the observations of the preceding paragraph, the idea behind backjumping remains a good one: to backtrack based on the reasons for failure. Backjumping notices failure when a variable's domain becomes empty, but in many cases a branch is doomed long before this occurs. Consider again the partial assignment \(WA = \text{red}, NSW = \text{red}\) (which, from our earlier discussion, is inconsistent). Suppose we try \(T = \text{red}\) next and then assign \(14T, Q, V, SA\). We know that no assignment can work for these last four variables, so eventually we run out of values to try at NT. Now, the question is, where to backtrack? Backjumping cannot work, because NT does have values consistent with the preceding assigned variables—NT doesn't have a complete conflict set of preceding variables that caused it to fail. We know, however, that the four variables NT, Q, V, and SA, taken together, failed because of a set of preceding variables. which must be those variables that directly conflict with the four. This leads to a deeper notion of the conflict set for a variable such as NT: it is that set of preceding variables that caused NT, together with any subsequent variables. This is not a complete conflict set of preceding variables, as we have no consistent solution. In this case, the set is WA and NSW, so the algorithm should backtrack to NSW and skip over Tasmania. A backjumping algorithm that uses conflict sets defined in this way is called conflict-directed backjumping.

We must now explain how these new conflict sets are computed. The method is in fact quite simple. The "terminal" failure of a branch of the search always occurs because a variable's domain becomes empty; that variable has a standard conflict set. In our example, SA fails, and its conflict set is (say) \{WA, NT, Q\}. We backjump to Q, and Q absorbs