The main advance over SOCRATES-style learning is that the learning procedure is called recursively, and the maximum recursion depth determines how much is learned about the circuit. Time complexity is exponential in r_{max}, the maximum recursion depth, but memory grows linearly with r_{max}. Table 7.16 shows the different recursive learning activities for the circuit of Figure 7.35, where column 0 reflects value assignments before and after recursive learning is done. The recursion causes learning to happen at column 1, which in turn recursively causes learning to occur in column 2. The outcome of this is that the necessary assignment $k = 1$ is learned.

Algorithm 7.8 *Demo_recursive_learning.*

```
{
    for each unjustified line
      {
        for each input: justification
          {
            assign controlling value;
            make implications and set up new list of resulting unjustified lines;
            if (consistent) Demo_recursive_learning ();
          }
        if (there are one or several signals f in circuit, so that f assumes
          same logic value V for all consistent justifications) then learn f = V,
          make implications for all learned signal values;
        if (all justifications are inconsistent) learn that the current
        situation of value assignments is inconsistent;
      }
}
```

A reader interested in finding the details of the recursive learning technique and its applications should rexamine the papers by Kunz and Pradhan [376, 377] and a recent book by Kunz and Stoffel [378].

Legal Assignment Test Generators. Rajski and Cox [531] maintained a *set* of legal signals on each circuit line (a power-set) for implications rather than using just a single signal for implications.

Implication Graph ATPG Algorithms. Chakradhar *et al.* developed the NNATPG [121, 127][§] algorithm family. These ATPG algorithms model logic gate behavior using implication graphs (see Section 7.1.6), which also enable redundancy identification [28, 399] and the modeling of transistor-level faults for ATPG [173, 250]. Chakradhar *et al.* also developed TRAN [122, 128], using a graph *transitive closure* algorithm to perform ATPG for huge circuits very rapidly.

[§]Acronym for *Neural Net ATPG.*