Constraint Satisfaction

- ✓ Constraint Satisfaction
  - ✓ The goal is to discover some problem state that satisfies a given set of constraints. E.g. Puzzles, real-world perceptual labeling problems, design tasks etc.
  - ✓ Helps in reducing the amount of search required as compared with a method that attempts to form partial solutions directly by choosing specific values for components of the eventual solution.
  - ✓ The fundamental issue is to include restrictions first that can be inferred from the rules of arithmetic. This reduces the number of guesses.
  - ✓ The initial state contains the constraints that are originally given in the problem description.
  - ✓ A goal state is any state that has been constrained enough, enough has to be defined for each problem.

- ✓ Constraint Satisfaction
  - ✓ It is a 2 step process.
    - ✓ First, constraints are discovered and propagated as far as possible throughout the system.
    - ✓ Then, if there is still no solution, search begins. A guess about something is made and added as a new constraint. Propagation can then occur with this new constraint and so forth.

Algorithm : Constraint Satisfaction

1. Propagate available constraints. To do this, first set *OPEN* to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until *OPEN* is empty:

(a) Select an object *OB* from *OPEN.* Strengthen as much as possible the set of constraints that apply to *OB.*

(b) If this set is different from the set that was assigned the last time *OB* was examined or if this is the first time *OB* has been examined, then add to *OPEN* all objects that share any constraints with *OB.*

(c) Remove *OB* from *OPEN.*

2. If the union of the constraints discovered above defines a solution, then quit and report the solution.

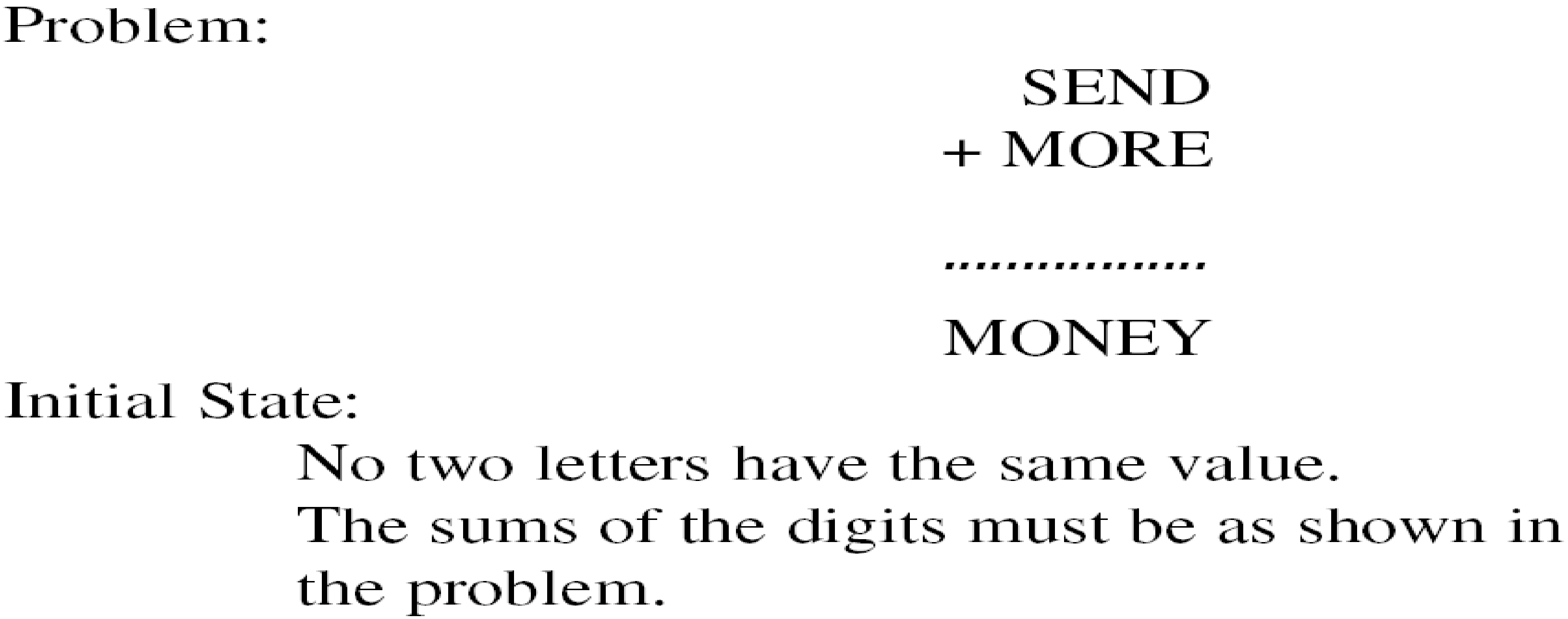
3. If the union of the constraints discovered above defines a contradiction, then return failure.

4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:

(a) Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.

(b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

A Cryptarithmetic Problem



Problem:

SEND

+ MORE

....

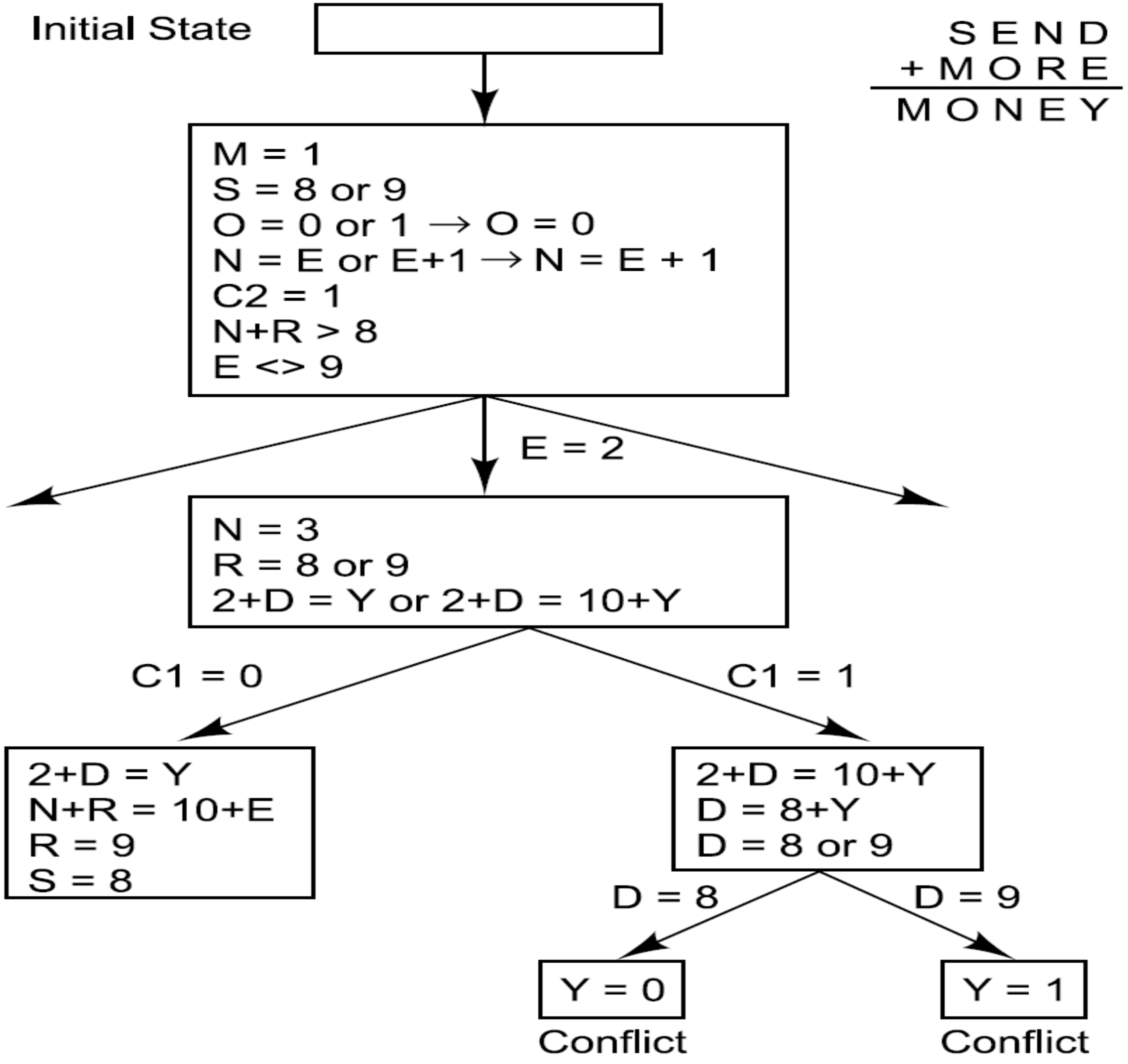
MONEY

Initial State:

No two letters have the same value.

The sums of the digits must be as shown in the problem.

Solving a Cryptarithmetic Problem



From column 5, **M = 1** since it is the only carry-over possible from the sum of two single digit numbers in column 4.

Since there is a carry in column 5, and M = 1, then **O = 0**

Since O = 0 there cannot be a carry in column 4 (otherwise N would also be 0 in column 3) so **S = 9**.

If there were no carry in column 3 then E = N, which is impossible. Therefore there is a carry and N = E + 1.

If there were no carry in column 2, then (N + R) mod 10 = E, and N = E + 1, so (E + 1 + R) mod 10 = E which means (1 + R) mod 10 = 0, so R = 9. But S = 9, so there must be a carry in column 2 so **R = 8**.

To produce a carry in column 2, we must have D + E = 10 + Y.

Y is at least 2 so D + E is at least 12.

The only two pairs of available numbers that sum to at least 12 are (5,7) and (6,7) so either E = 7 or D = 7.

Since N = E + 1, E can't be 7 because then N = 8 = R so **D = 7**.

E can't be 6 because then N = 7 = D so **E = 5** and **N = 6**.

D + E = 12 so **Y = 2**.