Middle East Technical ,University
Department of Computer Engineering

CENG 111

Fall '97

Worksheet 4

. . .
Ben, öteki, bir diğeri ona doğru ilerler
ilerlerim
zamanla anlarsın bu bir yanılsama
ölü şairlerin imgelerinden kalma
Sen de değilsin. O da değil
Kuzey yıldızı daha uzakta
yeniden yollara düşerler
düşerim
. . .
—*Murathan Mungan*

REGULATIONS

Due date: 11 January 1998, Sunday *(Not subject to postpone)*

Submission: Electronically. You will be submitting your program source code through a file which you will name as `hw4.sh` by issuing the following single line at the Unix prompt.

```
submit111 hw4.sh
```

The last moment for submission is 24:00 of the due date.

Team: There is **no** teaming up. The homework has to be done and turned in individually.

Cheating: All parties involved (source(s) and receiver(s)) get zero.

1 INTRODUCTION

1.1 The Alphametic Problem

In an *alphametic* problem the task is to find a one-to-one mapping from a set of letters (or symbols) to a subset of digits such that a given arithmetic operation in which each digit is represented by the corresponding letter or symbol is mathematically correct. Since it is among the main interest subjects of recreational mathematics, you can find many example alphametics in journals like *Journal of Recreational Mathematics*. Here is a famous example:

$$\begin{array}{cccccc} & & \mathbf{S} & \mathbf{E} & \mathbf{N} & \mathbf{D} \\ + & & \mathbf{M} & \mathbf{O} & \mathbf{R} & \mathbf{E} \\ \hline & \mathbf{M} & \mathbf{O} & \mathbf{N} & \mathbf{E} & \mathbf{Y} \end{array}$$

The set of letters used is $\{\mathbf{D}, \mathbf{E}, \mathbf{M}, \mathbf{N}, \mathbf{O}, \mathbf{R}, \mathbf{S}, \mathbf{Y}\}$. We will call this set the *alphabet* of the problem. Without making use of some mathematical intelligence (like the deduction of $M = 1$ due to is appearance in the left most position of the sum), generally speaking, one can say that each column in the sum is an equation that holds modulo 10 with a possibility of having received a *carry* from the previous column (except the right most column). So, if there exist M such columns, and N different values a carry can take, there are N^{M-1} possibilities for forming a system of M equations.

Furthermore the restriction on the solution domain inhibits the use of standard equation solution techniques. A brute force computer solution which goes over all possibilities roughly requires a time proportional to $10!/(10-\alpha)!$ where α is the *cardinality* (the count of elements) of the problem alphabet. For the above given example this means 1814400 trials for the worst case. In many cases this figure can be reduced drastically by building in some short cuts.

1.2 Backtracking

Backtracking is a technique used in searching for a solution. Assume there are M number of stages at which you have to make a decision (or choice) to arrive at a result. The problem is that the choice(s) you make may not lead to a satisfactory result (which we will call the *solution*). Now how to proceed? In other words how can we perform a systematic search among the possible choices? Let us assume that for stage i you have a set of choices S_i available. The backtracking method proceeds as follows:

1. Start with $i = 1$. Consider the first member (the first choice) of the S_1 (the set of all possible choices you could make for stage 1).
2. Consider the next i value. That is $i = 2$.
3. Consider the first possible choice of S_2.
4. Now check the consistency of the choices you made for stage 1 and for stage 2. Is there any conflict among the choices made so far? That means: Can you conclude that the choice you made for stage 1 contradicts with the choice you made for stage 2?
5. If your answer is **yes** (there is a contradiction) then this setting of choices has no chance to lead to a solution. Therefore we try an alternative for the <u>most recently made choice</u>. The most recent choice we made was for stage 2, so we pick another choice for it: the next possible choice from the set S_2; if such an alternative exists we pick it and continue with step (4). But if no untested choice of S_2 exists anymore then you have to go back to the choice you made for stage 1 and change it to another choice (the next untried one from the set S_1) then continue from step (2).
6. If your answer is **no** (there is no contradiction) then you may well advance to the next stage where you will make a choice for stage $i + 1$ (namely stage 3).
7. You carry out this procedure until you have to hand a choice for the very last stage (stage M) which does not contradict with all the choices made at previous stages, Now, none of the choices are contradictory and that setting of choices is a solution.

More formally [with the assumption of $next\_of(\mathsf{NONE}, \mathcal{S}) = first\_of(\mathcal{S})$]:

1 $i \leftarrow 1$
2 $choice_i \leftarrow$ NONE
3 **if** $\neg exists\_next\_of(choice_i, S_i)$
4 **then**
5 $decrement(i)$
6 **if** $i < 1$ **then return** (NO SOLUTION)
7 **else** continue from step *3*
8 **else**
9 $choice_i \leftarrow next\_of(choice_i, S_i)$
10 **if** $contradict(next\_of(choice_i, S_i), \{choices_k \mid k = 1 \dots i-1\})$
11 **then**
12 continue from step *3*
13 **else**
14 $increment(i)$
15 **if** $i > M$ **then return** (SOLUTION:$\{choices_i \mid i = 1 \dots M\}$)
16 **else** continue from step *2*

Study this algorithm. You shall understand the idea behind it and not attempt an line by line implemention. A Scheme implementation is more simple and does not require the index i.

The solution provided to the following problem is a good example of the backtracking-technique:

> *How is it possible to place N queens on a $N \times N$ chess board so that they do not threaten each other?*

First we have to write a predicate that will determine whether two queens threaten each other. A queen in chess can move along the column, the rwo, and the two diagonals through her present position. We can encode the positions on the board by numbering th rows and columns. Then the following function wil do:

```
(define (threat i j a b)
  (or (= i a)                                        ; Same row
      (= j b)                                        ; Same column
      (= (- i j) (- a b))                            ; SW-NE diagonal
      (= (+ i j) (+ a b))))                          ; NW-SE diagonal
```

Next, we can represent the configuration of several queens on the board using a list of two-element sublists, each sublist containing th row and cloumn number of one queen on the board. Suppose now that we plan to add a queen to a board. The following function will tell us whether th position (N,M) is safe:

```
(define (conflict n m board)
 (cond ((null? board) #f)
       (#t (or (threat n m (caar board) (cadar board))
               (conflict n m (cdr board))))))
```

With the prelinaries out of the way we can tackle the search problem. We write `queen` such that starting at row 0, an attempt is made to place a queen in column 0. After placing the first queen, it does not make sense to place another queen in the same row, so attention shifts to the next row. A safe square is found there. This process is to continue until all queens are placed on the board. If, at some stage, all squares on a particular row are threatened, the program has to back up. It is to do this by removing the last queen placed on board.

```
(define (queen size) (queen-aux () 0 size))

(define (queen-aux board n size)
  (cond ((= n size) (write (reverse board)))
        (#t (queen-sub board n 0 size))))

(define (queen-sub board n m size)                    ; Try next column
  (cond ((= m size) ())                              ; Hit end of row?
        (#t (cond ((conflict n m board))             ; Conflict
                  (#t (queen-aux (cons (list n m) board) (+ n 1) size)))) ; No
            (queen-sub board n (+ m 1) size))))      ; Now move right one
```

2 PROBLEM

You will write a function `solve-alphametic` which will take a list of 3 words (w_1 w_2 w_3) as argument. This way you introduce the alphametic problem:

$$w_1 + w_2 = w_3$$

It is a rule that none of the words w_k can start with a letter that means the digit 0 (zero). The result of the function call is expected to be an association list of the form:

(($letter_1$ $digit_1$) ($letter_2$ $digit_2$) ... ($letter_n$ $digit_n$))

which is the solution key. Here $\{letter_i \mid i = 1 \ldots n\}$ is the set of all distinct letters which appear in at least one of the 3 words. The association list is expected to be accendingly ordered: $i < j \iff letter_i < letter_j$.
Furthermore, $digit_i \in [0, 9]$ and $digit_i = digit_j \iff i = j$ (verbally: all digits are distinct). If all $letter_i \leftarrow digit_i$ substitutions are made according to the key, all the words will turn into numbers. A correct solution will be recognized as producing:

$$number_1 + number_2 = number_3$$

Here is the function call and the expected result for the introductory example:

```
(solve-alphametic '(SEND MORE MONEY))
((D 7) (E 5) (M 1) (N 6) (O 0) (R 8) (S 9) (Y 2))
```