The query
  which (x: acetylsalicylic-acid to-be-dissolved-in x)
yields then:
  dilute-NaOH.
This knowledge base consisting of sentences and rules can now be extended at will. For instance if one wants to know in which solvent one should dissolve sulfanilamide, a substance containing the functional group sulfonamide, one adds:
  sulfonamide weak-acidic-function
  sulfanilamide contains-function sulfonamide
and then asks the to-be-dissolved-in query.
Alternatively one can add additional rules such as:
  x to-be-dissolved-in dilute-HCl if (either x acid-base-status weak-base or x acid-base-status strong-base).

In this second Corner we have tried to give an idea about how to construct rules in PROLOG. The way in which knowledge was manipulated in these two Corners is simple but not very efficient. Indeed, PROLOG (and similar languages such as LISP) have really been constructed to manipulate lists of knowledge items instead of single items. Lists will be discussed in the next Corner.

Reference

1 D. L. Massart and J. Smeyers-Verbeke, *Trends Anal. Chem.*, 4 (2) (1985) 50.

FORTH — A good programming environment for laboratory automation? II. An example from the laboratory

Daniel Ph. Zollinger and M. Bos
Enschede, The Netherlands

After a short introduction to FORTH computer language in a preceding Computer Corner[1], in this contribution an application of FORTH in the laboratory is presented. The software package used is Fys-FORTH version 3, which was developed at the State University of Utrecht, The Netherlands. This FORTH, based on the FORTH-79 standard, is provided with useful (additional) utilities such as assembler, editor, floating point arithmetics and graphics. It is written for the Apple II and related microcomputers.

Data acquisition

The main task of a laboratory computer is data acquisition, which is performed using an analog-to-digital converter (ADC) that converts an (electronical) analog signal into a digital one. Timing is an essential part of this procedure as the conversion usually is supposed to take place at a specified point of time.

For this task, two FORTH words, both using assembly language, were defined: ADIN and TIMSTART. Using these words, writing routines for data acquisition can be done with very little programming effort. The word ADIN expects the number of a channel on the stack and puts the result of the conversion onto the stack. For example, after typing $\boxed{\text{2 ADIN}}$ the result of the conversion at channel number two will be on the stack and can be printed by the 'dot' command ($\boxed{.}$). TIMSTART will start a clock routine with a given interval in milliseconds. This value is expected to be on the stack: $\boxed{\text{1 TIMSTART}}$ will start the clock in milliseconds, $\boxed{\text{1000 TIMSTART}}$ does the same in seconds. The actual time is stored in a variable called TIME. (The word TIME will put the adress of the variable on the stack; its contents can be obtained by using the word @ ('fetch') and then printed by using the dot, so $\boxed{\text{TIME @ .}}$.)

Simple testing-routines

Now routines for data acquisition can be developed. In FORTH, defining new words is usually done as follows: the definition is opened with the word $\boxed{:}$, immediately followed by the name of the new word and its definition, using FORTH words that are already known to the dictionary. The definition is closed by the word $\boxed{;}$.

How does a routine look like that scans all the channels of the ADC (in this example there are

Glossary

Explanation of some FORTH words and statements used in the text

.	prints the integer on top of the stack
@	replaces the address on top of the stack by its integer contents
!	stores the second number (integer) on the stack at the address on top of the stack
=	tests whether the second integer on the stack equals the integer on top of the stack and leaves a boolean result
DROP	removes the integer on top of the stack
DUP	duplicates the integer on top of the stack
?KEY	checks whether a key was pressed by the user and leaves a boolean result on the stack
DO . . . LOOP	repetitive statement: the words between DO and LOOP are executed as often as the difference between the two integers on the stack in front of DO
BEGIN . . UNTIL	repetitive statement: the words between BEGIN and UNTIL are executed as long as the integer on top of the stack in front of UNTIL equals zero

twelve) and prints the results of the conversions on a new line?

```
: AD-SCAN 12 0 DO I ADIN
                  CR .
               LOOP ;
```

'I' = loop-counter; 'CR' = carriage return.

Or a word that will continue to perform an A-D-conversion at a given channel and print the result until a key is pressed:

```
: AT BEGIN DUP ADIN
                CR .
              ?KEY
     UNTIL DROP ;
```

Notice that the parameter on the stack, expected by this word, is carried through the BEGIN . . . . UNTIL structure by means of the word DUP and removed by DROP!

A word to sample a signal at high rate from channel number 3 during one period of the line frequency (in this case 50 Hz) could look like this:

```
: BURST 1 TIMSTART
        BEGIN 3 ADIN TIME @ 20 =
        UNTIL;
```

After execution of this word there will be a whole lot of numbers on the stack – the user must take care of this and avoid 'stack overflow' himself.

Often it is advantageous to have an array of data, for example in the use of a 'digital filter'. Such a structure can be created by typing `VARIABLE DATA 198 ALLOT`, which makes it possible to store 100 data points. The word VARIABLE creates a memory location of 2 bytes (here called DATA) and the word ALLOT adds a number of bytes to this memory location. To store 100 data points from channel 4, sampled with an interval of 25 milliseconds, one can define the following word:

```
: GETDATA 1 TIMSTART
          100 0 DO BEGIN TIME @
                      I 25 * 25 + =
                   UNTIL
                   4 ADIN
                   I 2 * DATA + !
                LOOP ;
```

where 'I 25 * 25 +' is the sampling time; 'I 2 * DATA +' is the adress of the Ith element of the array at which the result is stored by the word $\boxed{!}$.

A complete program

Starting with simple definitions as the ones shown here we developed a program for DC-TAST polarography including

- keyboard input of scanning parameters (initial and final potential, scan rate);
- automatic recording of the polarogram and stor-

age of the data on disk;
- determination of number and location of the reduction waves in the polarograms;
- least-squares curve fitting (using matrix-inversion routines) of the individual polarographic waves yielding half-wave potential, limiting current value and slope of the so-called log-plot;
- addition of reagent solutions to the polarographic cell from a motor burette.

All the words for this program were written in FORTH, such as TAST (recording of a polarogram), SAVEDATA (storage of data on disk), SMOOTH (Savitzky–Golay smoothing), ADD (addition of solution to the polarographic cell) and FITWAVES (for curve-fitting).

Conclusions

Like any other computer language, FORTH has some strong and (more or less serious) weak points – in this respect it is a common language. Its specific properties, however, bring about an unusual amount of flexibility which makes it possible to use FORTH as the adequate personal tool for solving (laboratory) automation problems.

Reference

1 D. P. Zollinger and M. Bos, *Trends Anal. Chem.*, 4 (3) (1985) 60.

Daniel Ph. Zollinger and M. Bos are at the Laboratory of Chemical Analysis, Technical University of Twente, 7500 AE Enschede, The Netherlands.

interface

Expert systems and analytical chemistry

J. W. A. Klaessens, G. Kateman and B. G. M. Vandeginste
Nijmegen, The Netherlands

Introduction

The increasing applicability of computers has given rise to the rapid development of a new research area in analytical chemistry, generally referred to as chemometrics. This can be seen as the integration of computer science into analytical chemistry, and via the computer the use of complex statistical concepts. These statistical techniques, such as Kalman filtering[1] and curve resolution[2], depend critically on the arithmetical aid of the computer. For instance, in HPLC using multiwavelength diode array detection the use of the curve resolution technique would be difficult without computer assistance. An extremely important point, however, is that these techniques enable the chemist to extract extra information from experimental data.

To extract as much reliable information as possible from an experiment is an important objective in analytical chemistry. Equally important is how to obtain the experimental data. How can a complex analytical apparatus be optimally tuned? Which procedure should be used when a component must be determined in a given sample? In a broader sense, how should an analytical laboratory be organised in an optimal way to meet the demands of its clients? Answering such questions is in fact decision making. A purely statistical approach does not yield satisfactory results. Indeed, to make such decisions one has to rely heavily on experience which makes problem solving highly personal. This process can be substantially improved by the use of expert systems[3].

What are expert systems?

Expert systems can be defined as problem solving programs that solve substantial problems generally acknowledged as being difficult and requiring expertise. In order to avoid a devaluation of its meaning, it must be stressed that an expert system must be able to solve substantial problems. Essentially it is just a computer program, although usually not written in conventional computer languages like FORTRAN and ALGOL68. There are differences between expert systems and conventional programs. The latter have a fixed structure. On running the program, one knows beforehand the sequence of instructions carried out by the computer: the line of reasoning must be written out beforehand. An expert system, however, does the reasoning itself: in the space of possible solutions, it tries to determine the right solution. In other words, the system reasons from one node to another in a space that can be represented as a graph or a tree, until the final solution is reached.