Anul XLV 2000

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

2

Redacţia: 3400 Cluj-Napoca, str. M. Kogălniceanu nr. 1 Telefon 405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

3D TERRAIN RECONSTRUCTION USING SCATTERED DATA SETS

DUMITRU RADOIU AND ADRIAN ROMAN

Abstract. The paper presents a scientific visualization technique using scattered data. The visualization system is based both on modules written by the authors, implementing controlled Shepard interpolation algorithm and ready-to-use VTK classes.

1. Introduction

The proposed visualization system is based both on modules written by the authors (implementing sub-sampling, re-sampling and controlled Shepard interpolation algorithm), sustaining the modeling level, and VTK classes [4], for the logical visualization and physical visualization level.

The application goal is to transform the 2D data, representing the altitude measured in some known points, non-uniform distributed, into interactive 3D maps. The structure of the output data/files should allow the Internet distribution. The main issues are the speed and accuracy of the used algorithms [7].

A file that contains the non-uniform distributed data, namely, represents the data set: the position of the nodes on a horizontal plane and the corresponding altitudes. The controlled re-sampling of the input data solves the modeling problem; in order to obtain a uniform distributed data set. The re- sampling propagates the local properties (the altitude value) towards the unknown-valued points. The propagation is implemented using the modified local Shepard interpolation. A number of constraints are necessary [5, 2, 3]. The whole process is described below.

2000 *Mathematics Subject Classification.* 65D18.

1998 *CR Categories and Descriptors.* I.3.6 [**Computing Methodologies**]: Computer Graphics – *Methodology and Techniques.*

The uniform distributed data set is visualized using the warping of a plan, taking into account the corresponding altitude of each point [1, 6]. Artificially associated colors allow a better perception of the visualized object.

2. Re-sampling of the data set

The data set analysis shows that the input data set is non-uniform, meaning that the data is represented by unconnected nodes and their associated altitudes. In order to obtain a uniform distribution, as requested by the logical visualization level, a re-sampling of the data set is performed.

We denote by N the number of non-uniform distributed points on a plane, in a considered domain B.

$$(x_i, y_i), i = 1, \ldots, N \tag{1}$$

$$(x_i, y_i) \in B \subseteq \mathbb{R}^2 \tag{2}$$

The corresponding altitudes are given by function $f : B \to \mathbb{R}$, which associates to each point (x_i, y_i) a real value $f(x_i, y_i)$ denoted by f_i.

The visualization platform used, personal computers, and the dimension of the data set, 25500 nodes, imposed the use of a modified version of the local Shepard interpolation. The introduced constraints had as result an important speed improvement.

We denote by Φ the interpolation function $\Phi : B \to \mathbb{R}$. The Shepard interpolation function is expressed as sum of weights:

$$\Phi(x, y) = \sum_{j=1}^{N} w_j(x, y) f_j \tag{3}$$

The interpolation function has to fulfill the condition:

$$\Phi(x_i, y_i) = f_i, i = 1, \ldots, N. \tag{4}$$

Then weight function is supposed to:

$$\sum_{j=1}^{N} w_j(x,y) = 1. \tag{5}$$

$$w_j(x,y) = \begin{cases} \geq 0 & \text{for all } (x,y) \in B \\ = 1 & \text{for } (x,y) = (x_j, y_j) \\ = 0 & \text{for } (x,y) = (x_k, y_k), k \neq i, (x_k, y_k) \notin B \end{cases} \tag{6}$$

The weights of the local Shepard interpolation are defined below:

$$\begin{cases} w_j(x,y) = \frac{\frac{1}{\Psi_j^{\mu}}}{\sum_{i=1}^{N} \frac{1}{\Psi_i^{\mu}}}, 0 < \mu < \infty, \text{ with} \\ \Psi_j(x,y) = \begin{cases} \frac{R}{r_j} - 1 & \text{for } 0 < r_j < R, \\ 0 & \text{for } r_j \geq R \end{cases} \end{cases} \tag{7}$$

where r_j is the distance between the points (x,y) and (x_j, y_j):

$$r_j(x,y) = \sqrt{(x - x_j)^2 + *y - y_j)^2}, j = 1, \ldots, N \tag{8}$$

R represents the radius of circle, centered in the point (x,y), that determines those interpolation points (x_i, y_i) that are going to influence the interpolation function. We consider $\mu = 2$.

The N nodes can be seen as belonging to a rectangle defined by the coordinates (x_{min}, y_{min}) and (x_{max}, y_{max}), where:

$$\begin{aligned} x_{min} &= \min\{x_j | j = 1, \ldots, N\} \\ x_{max} &= \max\{x_j | j = 1, \ldots, N\} \\ y_{min} &= \min\{y_j | j = 1, \ldots, N\} \\ y_{max} &= \max\{y_j | j = 1, \ldots, N\} \end{aligned} \tag{9}$$

The rectangle area that contains all the interpolation points is:

$$A = (x_{max} - x_{min})(y_{max} - y_{min}). \tag{10}$$

The minimum value of R can be easily approximated. We associate to each node (x_j, y_j) a region that has the area equal to A/N. If the N initial nodes would be uniform distributed then each region should contain only one node (Figure 4).

R minimum is, in fact, the diagonal of the rectangle of area A/N.

The idea is to propagate the local properties, introduced by the initial N nodes, to all points. We start from the point:

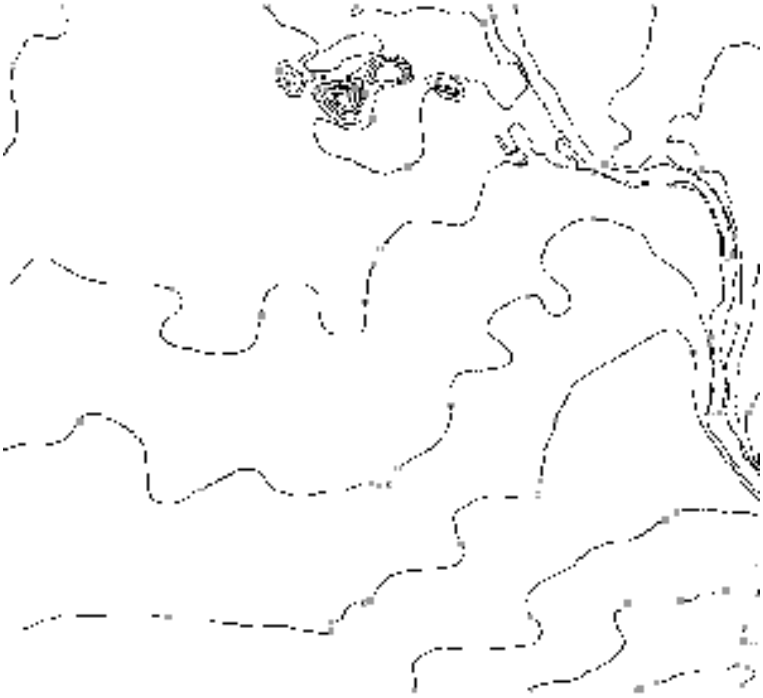


FIGURE 1. The data set representing Dealul Melcilor (Braşov) (isoline representation)

$$x_{CM} = \frac{\sum_{i=1}^{N} x_i \cdot f(x_i, y_i}{\sum_{i=1}^{N} x_i} \tag{11}$$

$$y_{CM} = \frac{\sum_{i=1}^{N} y_i \cdot f(x_i, y_i}{\sum_{i=1}^{N} y_i} \tag{12}$$

The starting point can be chosen, as well, by other techniques: clustering, visual inspection, etc.

Algorithm CONTROLLED LOCAL SHEPARD INTERPOLATION

InputData: A set of N nodes on a plane, the number of columns and rows used for re- sampling, the coordinates of the starting point (optional)

Outputata: A uniform data set

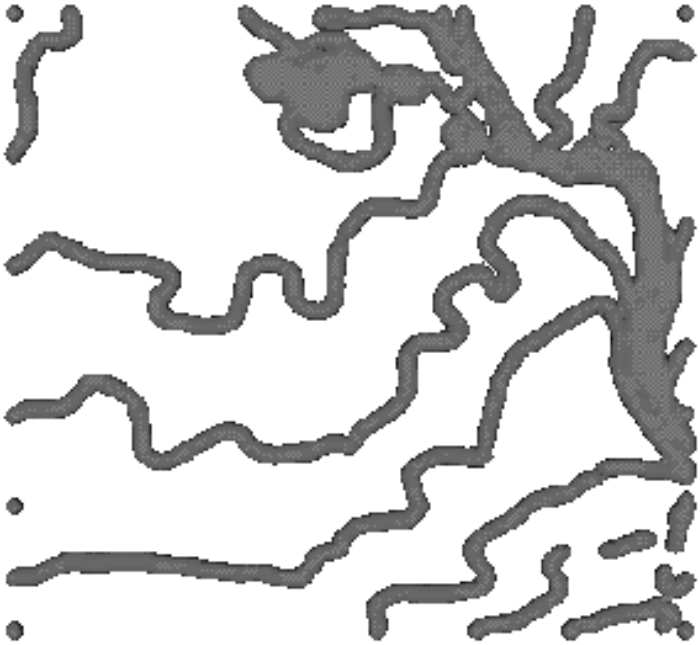


FIGURE 2. The data set representing Dealul Melcilor (Brasov) (visualized by the splatting technique and the iso- surfaces generation)

(1) For the N node we find x_{min}, y_{min}, x_{max} and y_{max}.
(2) We compute R minimum.
(3) We generate the uniform data set.
(4) We search for the starting point for interpolation.
(5) We use the above described interpolation function to compute the attribute value.
(6) We insert the computed value in the set of the interpolation points.
(7) We repeat the steps 5 and 6 covering the nodes in a spiral motion for the whole grid.

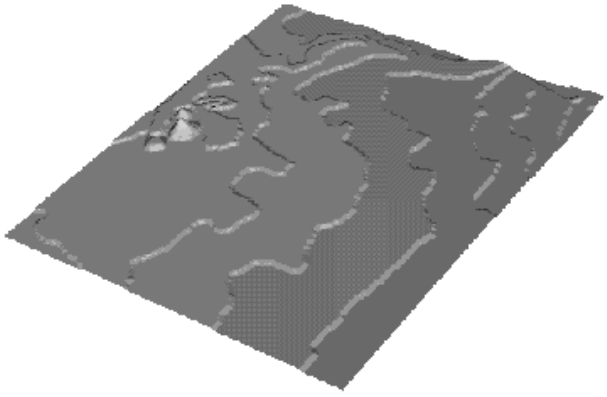


FIGURE 3. Visualization of the uniform distributed data set, representing Dealul Melcilor (Brasov) (warping technique); the warping factor is exaggerated for better perception

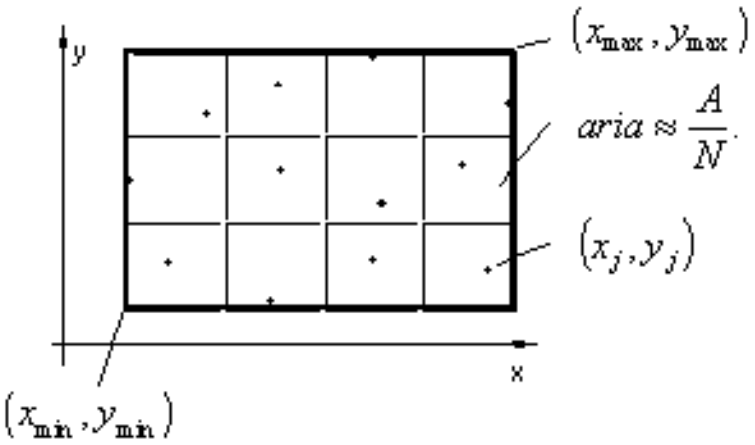


FIGURE 4. The rectangle with all the interpolation points

3. WARPING

The 3D reconstruction of the terrain is performed by warping taking into account the corresponding altitude of each point. The warping is done by the movement of the points of a 2D surface following the normal direction to the surface. The warping is controlled by a scaling factor (Figure 3).

4. Artificially associated colors

The perception of the terrain characteristics could be improved by colors artificially associated to each point, the color mapping taking into account the altitude.

Generally, if there is no attribute to be directly mapped into a color table, the necessary attributes have to be generated. A filter producing scalar values corresponding to a certain altitude does the scalar generation.

5. Conclusions

The 3D visualization of the terrain remains a hot issue. The controlled local Shepard interpolation algorithm has to be further tested against standard data sets.

Our tests included:

- known uniform functions were sampled and the values were visualized;
- data sets were then sub-sampled uniformly or non-uniformly;
- resulted data set was used as input data for the modeling module implementing our modified Shepard algorithm;
- the starting point was automatically chosen or it was chosen as result of a visual inspection of the non-uniform distributed data set (Figure 2);
- the two visual objects, obtained from the initial data set and the interpolated data set, respectively, were visually inspected.

The 3D visualization of the terrain can be performed automatically if the proper interface is implemented.

References

[1] Brodie K., Butt S., Mashwama P., Visualization of Surface Data to Preserve Positivity and Other Simple Constraints, Computer and Graphics, Vol. 17, No 2, 1985, p. 55–64

[2] Brodlie K., Mashwama P., "Controlled Interpolation for Scientific Visualization" in "Scientific Visualization", Overview, Methodologies, Techniques, IEEE Computer Society, 1997, p. 253–276.

[3] Nielsen G. M., "Scattered Data Modelling", IEEE Computer Graphics and Applications, Vol. 13, No. 1, 1993, p. 60–70

[4] Radoiu D., "VTK in Desktop Scientific Visualization", in Advanced Educational Technologies, Editura Universitatii Petru Maior, 1999, p. 21–30

[5] Radoiu D., "Interpolare Shepard locala cu restrictii", Comunicare, Sesiunea de comunicari a cadrelor didactice, Universitatea Petru Maior, 1999

[6] Ruprecht D., Muller H., Image Warping with Scattered Data Interpolation, IEEE Computer Graphics and Applications, March 1995, p. 37–43

[7] Stewart A. James, "Fast Horizon Computation al All points of a Terrain With Visibility and Shading Applications", IEEE Transactions on Visualization and Computer Graphics, Jan-March 1998, Vol4, nr 1, pp. 82–93

Universitatea Petru Maior, Târgu Mures

Universitatea Politehnica Bucureşti

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

PARALLEL VERIFICATION AND ENUMERATION OF TOURNAMENTS

GÁBOR PÉCSY AND LÁSZLÓ SZŰCS

Abstract. The area of tournaments is extensively discussed in literature. In this article the authors introduce asymptotically optimal sequential algorithms for the verification of score vectors and score sequences and a sequential polynomial algorithm for enumeration of complete tournaments. The extensions of these algorithms to different parallel architectures including CREW PRAM, linear array, mesh and hypercube are also presented. It is shown that most of the parallel algorithms discussed here are work-optimal extensions of the sequential ones.

1. Introduction

Round-robin tournaments are popular in the world of sport, games or elections and they are very much discussed in computer science as well. A tournament is an $n \times n$ real matrix. The elements of the main diagonal t_{ii} equal to zero and the pairs of symmetric elements $t_{ij} : t_{ji}$ give the result of the match between P_i (the i-th player) and P_j. The sum of the elements of the i-th row (s_i) is called the score of the i-th player. A non-decreasingly ordered sequence of the scores is the score sequence of the tournament.

The most usually discussed problems regarding tournaments include:

- Verification of a score sequence/score vector means the decision if there exists a tournament for a given score sequence/score vector.
- Enumeration of score sequences means the counting of the possible different score sequences for a given number of players (n).

The outline of the paper is as follows. The following section describes the problems and the used computational models more formally. Section 3 deals with verification problems and their sequential and parallel solutions.Then Section 4

2000 *Mathematics Subject Classification.* 05C20, 68Q25, 65Y05.

1998 *CR Categories and Descriptors.* G.2.1 [**Discrete mathematics**]: Combinatorics – *Counting problems*; F.2.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical algorithms and problems – *Computations on discrete structures* C.1.4 [**Processor architectures**]: Parallel architectures – *Distributed architectures* .

presents our results about enumeration. Finally, a table summarizes the results with possible future works.

2. Basic notions

2.1. Tournaments.

Tournament: A round-robin tournament is an $n \times n$ real matrix $T_n = [t_{ij}]$ ($n \geq 2$). The elements of the main diagonal t_{ii} equal to zero and the pairs of symmetric elements $t_{ij} : t_{ji}$ give the result of the match between P_i (the i-th player) and P_j. $t_{ij} = t_{ji}$ means a draw, while $t_{ij} > t_{ji}$ means the win of P_i against P_j.

Score vector: The sum of the elements of the i-th row (s_i) is called the score of the i-th player and the vector $(s_1, \ldots, s_n)$ is called the score vector of the tournament.

Score sequence: A non-decreasingly ordered sequence of the scores is denoted by $q =< q_1, \ldots, q_n >$ and is called the score sequence of the tournament.

Complete tournament: We call a tournament complete if in it the permitted elements are 0 and 1 and the sum of the symmetric elements ($t_{ij}+t_{ji}$, where $i \neq j$) is always 1. A set of tournaments is called complete for a given n if it contains all possible n player complete tournaments.

2.2. Computational models.

Sequential model: A RAM running pseudo-code similar to structured programming languages.

PRAM: Parallel RAM, consists of a shared memory and possibly infinite number of RAMs which operate with the same pseudo code as in the sequential case. Depending on the methods of accessing the shared memory there are different types of PRAM.

EREW: Exclusive Read Exclusive Write
ERCW: Exclusive Read Concurrent Write
CREW: Concurrent Read Exclusive Write
CRCW: Concurrent Read Concurrent Write

As concurrent read of shared memory is usually allowed while the result of concurrent write is ambigous we decided to use CREW PRAM in our study.

Linear array: A linear array consists of p processors (named $1, 2, \ldots, p$). Processor i has two direct bidirectional interconnection links to its neighbouring processors ($i-1$ and $i+1$) except processor 1 and p which has only one neighbour.

Mesh: A mesh is an $a \times b$ grid in which there is a processor at each grid point. The edges correspond to communication links and are bidirectional. In this paper we consider only square meshes, where $a = b$.

Hypercube: A hypercube of dimension d has $p = 2^d$ processors. Each processor can be labeled with a d-bit binary number. A processor is connected only to those processors which label differs in only one bit.

Work-optimal: We call a parallel algorithm work-optimal compared to a given sequential algorithm if $\frac{P_n * p}{S_n} = O(1)$, where S_n is the run time of the sequential algorithm, P_n is the run time of the parallel algorithm and p is the number of processors.

Notice that if a parallel algorithm is work-optimal compared to a given asymptotically optimal sequential algorithm then the parallel algorithm itself is asymptotically optimal as well.

3. Verification

Verification of a score sequence/score vector means the decision if there exists a tournament for a given score sequence/score vector. Landau [5] proved the following theorem which gives necessary and sufficient condition for the existence of a complete tournament for a particular score sequence.

Theorem 1. *A non-decreasing sequence of n integers $< q_1, \ldots, q_n >$ is a score sequence if and only if*

$$\sum_{i=1}^{k} q_i \geq \binom{k}{2}$$

for each $k = 1, 2, \ldots, n$ with equality for $k = n$.

3.1. **Sequential algorithm.** Theorem 1 can be directly applied to verify score sequences as they are ordered non-decreasingly. The following algorithm solves this problem in $\Theta(n)$ time and with $O(1)$ auxiliary memory.

```
1  s:=0; i:=1; ok:=(q_n<n);
2  while i<n and ok loop
3      s:=s+q_i;
4      ok:=s≥(i*(i-1)/2);
5      i:=i+1;
6  end loop
7  ok:=ok and (s+q_n)=(n*(n-1)/2);
8  return ok;
```

Algorithm 1: Sequential algorithm for score sequence verification

As the trivial lower bound for the verification problem is $\Omega(n)$ — the algorithm has to read the input — Algorithm 1 is asymptotically optimal for score sequence verification.

In case of score vectors the input is not ordered properly so Theorem 1 (and Algorithm 1) can not be applied directly. One possible solution is to sort the input and then apply Algorithm 1 to the result. It is known that sorting of general keys takes $\Omega(n \log n)$ time but if keys are integer numbers from the range [0..k] then they can be sorted in $O(max(n,k))$ time. Such algorithm can be found in chapter 9 of [1]. In case of a score vector all elements must belong to range [0..n-1] so the vector can be sorted in $O(n)$ time. This condition can also be verified in $O(n)$ time, so we get the following algorithm.

Step 1: Verify whether all elements in the vector fall in the range [0..n-1]. If not then the input can not be a score vector.
Step 2: Sort the input.
Step 3: Use Algorithm 1.

Algorithm 2: Sequential verification of score vectors

Note that Algorithm 2 is asymptotically optimal for the same reason as Algorithm 1.

3.2. **Parallel algorithms.** In this section we provide an efficient way to implement Algorithm 1 and Algorithm 2 on different parallel architectures. [1]

3.2.1. *PRAM.* On a CREW PRAM Algorithm 1 can be implemented in a very straightforward way.

Step 1: For all processors compute the prefix-sums (r_i) of the input sequence.
Step 2: Processor p_i $(i := 1, 2, \ldots, n-1)$ calculates $l_i := (r_i \geq (i * (i-1)/2))$ while p_n calculates $l_n := (r_n = (n * (n-1)/2))$.
Step 3: Calculate $OK := l_1 \wedge \ldots \wedge l_n$ using the prefix computation algorithm with all processors.

Algorithm 3: Simple parallel algorithm for score sequence verification

Step 1 can be done in $O(\log n)$ time, Step 2 takes $O(1)$ time and Step 3 is $O(\log n)$ again. Note that in case of CRCW PRAM Step 3 takes only $O(1)$ time.

This algorithm uses $O(n)$ processors and operates in $O(\log n)$ time therefore it is not work-optimal, but it can be improved to run on $O(\frac{n}{\log n})$ processors in $O(\log n)$ time which is work-optimal. To achieve this we divide the input

[1] It is assumed that the reader is familiar with the prefix-sum computation and other well-known parallel algorithms summarized in [2] as they are building blocks of the following algorithms.

into $\log n$ long pieces. Processor p_i will sequentially calculate prefix-sums of $s_{(i-1)\log n+1}, \ldots, s_{i \log n}$. Then the processors will apply the original prefix computation algorithm on the sums of the pieces. In the third step each processor will update the prefix-sums of the corresponding piece by adding the sum of all the previous pieces. After this the processors will calculate l_i values sequentially for all elements belonging to them and finally they perform a prefix computation using the same algorithm as for the prefix-sum calculation to determine $OK := l_1 \wedge \ldots \wedge l_n$.

Step 1: For all processors compute sequentially the prefix-sums $(t_{i,j},$ where $i := 1, 2, \ldots, \frac{n}{\log n}; j := 1, 2, \ldots, \log n)$ of the corresponding piece of input sequence.

Step 2: For all processors compute the prefix-sums $(r_{i \log n})$ of $t_{i,\log n}$.

Step 3: For all processors compute sequentially $r_{(i-1)\log n+j} := r_{(i-1)\log n} + t_{i,j}$ $(i := 1, 2, \ldots, \frac{n}{\log n}; j := 1, 2, \ldots, \log n)$.

Step 4: Processor p_i $(i := 1, 2, \ldots, \log n)$ calculates $l_{(i-1)\log n+j} := (r_{(i-1)\log n+j} \geq (((i-1)\log n + j) * ((i-1) * \log n + j - 1)/2))$ using equality at the last position.

Step 5: Calculate $OK := l_1 \wedge \ldots \wedge l_n$ using the prefix computation algorithm described in Step 1–3, with all processors.

Algorithm 4: Work-optimal verification of score sequences on CREW PRAM

In this algorithm all steps take $O(\log n)$ time so the whole algorithm works in $O(\log n)$ time as well. It uses $O(\frac{n}{\log n})$ processors so this is a work-optimal parallelization of Algorithm 1. As Algorithm 1 is asymptotically optimal algorithm for the score sequence verification problem the same holds for Algorithm 4 as well.

Notice that in this algorithm only the parallel steps (Step 2 and 5) require interprocessor communication and these steps are all parts of prefix computations.

3.2.2. *Linear array.* A lower bound on every interconnection networks for a problem is the diameter of the network if all processors of the network contributes to the computation of the result. As the diameter of a linear array of n processors is $n-1$, $\Omega(n)$ is a lower bound for the score sequence and score vector verification problems. These problems can be solved in $O(n)$ time on a single processor as well, the trivial (and optimal) solution is to send all data to the first processor of the array – this can be done in $O(n)$ time – and do the verification there, using the sequential algorithms. These solution are work-optimal if and only if the number of processors in the array is $O(1)$.

3.2.3. *Mesh.* The diameter of a p processor mesh is $\sqrt{p}$, so $\Omega(\sqrt{p})$ is a lower bound to an algorithm. The mesh adaptation of Algorithm 3 solves the problem in $O(\sqrt{n})$ if $p = n$. But we can apply the same technique as in Algorithm 4. If we assign $n^{\frac{1}{3}}$ element for each processor of a $n^{\frac{1}{3}} \times n^{\frac{1}{3}}$ mesh then both the sequential and the

parallel steps work in $O(n^{\frac{1}{3}})$ time. The number of processors in this case is $n^{\frac{2}{3}}$ so the algorithm is work-optimal.

3.2.4. *Hypercube.* In a p processor hypercube, prefix computation can be performed in $O(\log p)$ time, which means that adoptations of Algorithm 3 and Algorithm 4 can work in the same time bounds as in case of CREW PRAM.

3.2.5. *Parallel score vector verification.* Unfortunately there is no known work-optimal parallel sorting algorithm for integer key from a given domain. This means that the most difficult step in the parallel adoptation of Algorithm 2 is the sorting. The complexity of sorting usually exceeds the complexity of the other steps so the overall complexity of the algorithm equals the complexity of sorting the input. For PRAM and hypercube there are algorithms which can sort general keys in $O(\log^2 n)$ time.

4. Enumeration

Enumeration of score sequences means the counting of the possible different score sequences for a given number of players (n).

For $n > 1$ let $f_n(T, E)$ be the number of non-decreasing sequences of integers satisfying

$$\sum_{i=1}^{n} q_i = T, q_n = E \text{ and } \sum_{i=1}^{k} q_i \geq \binom{k}{2}, k = 1, 2, \ldots, n-1.$$

Narayana and Bent in [7] presented a recursive formula for determining $f_n(T, E)$:

$$f_1(T, E) = \begin{cases} 1, & \text{if } T = E \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

for $n \geq 2$

$$f_n(T, E) = \begin{cases} \sum_{k=0}^{E} f_{n-1}(T - E, k), & \text{if } T - E \geq \binom{k}{2} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Let t_n be the number of possible score sequences in case of n players. For $n > 1$ we have the following formula for t_n:

$$t_n = \sum_{E=r}^{n-1} f_n(\tbinom{n}{2}, E), \text{where } r = \left\lfloor \frac{n}{2} \right\rfloor.$$

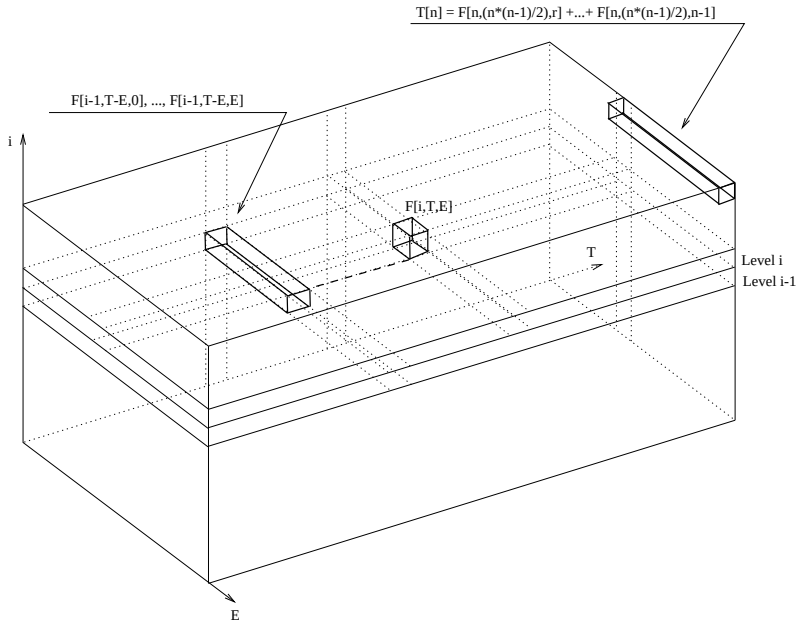


FIGURE 1. Array element dependencies in the non-optimized formula

4.1. **Units of measure.** The experimental results indicate that the value of t_n is increasing exponentially with n ($t_n = \Omega(2^n)$) which implies that we need $\log t_n = \Theta(n)$ memory to store a single number. This also implies that addition of such numbers takes $\Theta(n)$. In the next parts of the article we will count the number of operations (addition, send, receive and assignment) on the elements of the array during the analysis of the algorithms. In a real implementation all of these operations can be done in $O(n)$ time.

4.2. **Sequential algorithms.** The most straightforward recursive calculation of t_n using the recursive formula (1) has exponential run time so it is not applicable for bigger n values. Using dynamic programming the run time can be reduced significantly into polynomial domain.

4.2.1. *Algorithm using dynamic programming.* The following algorithm uses array of size $n \times n \times (n(n-1)/2+1)$ elements and works in $\Theta(n^5)$ time.

The operation of the algorithms can be divided into two phases. First phase is filling in the array F which contains the values of $f_i(T,E)$ for $i = 1..n, T = 0..\frac{n(n-1)}{2}$ and $E = 0..n-1$, thus the dimensions of the array are $n \times \frac{n(n-1)}{2}+1 \times n = \Theta(n^4)$. Calculating a particular $F[i,T,E]$ element takes $\Theta(1)$ time for $i = 1$ —

```
1  for i:=1 to n loop
2      for T:=0 to (n*(n-1)/2) loop
3          for E:=0 to n-1 loop
4              if i=1 then
5                  if T=E then
6                      F[i,T,E]:=1;
7                  else
8                      F[i,T,E]:=0;
9                  end if;
10             else
11                 F[i,T,E]:=0;
12                 if (T-E)≥ ((i-1)*(i-2)/2) then
13                     for k:=0 to E loop
14                         F[i,T,E]:=F[i,T,E]+F[i-1,T-E,k];
15                     end loop;
16                 end if;
17             end if;
18         end loop;
19     end loop;
20 end loop;
21 TN:=0;
22 for E:=(n div 2) to n-1 loop
23     TN:=TN+F[n,(n*(n-1)/2,E];
24 end loop;
25 return TN;
```

Algorithm 5: Calculating the number of score sequences using dynamic programming

lines 5–9 — and $O(n)$ otherwise — lines 11–16 (see Figure 1). This means that the whole algorithm runs in $O(n^5)$ time. The second phase is to calculate the number of score sequences (TN) using the filled array F (see Figure 1).

4.2.2. *Improved algorithm.* In Algorithm 5 the number of the used array elements is $\Theta(n^4)$ so $O(n^4)$ is a lower bound to the run time of any solution using this approach, but the run time is $O(n^5)$. We show that using a proper reformulation of equation (1) the run time of the algorithms can be reduced to $\Theta(n^4)$.

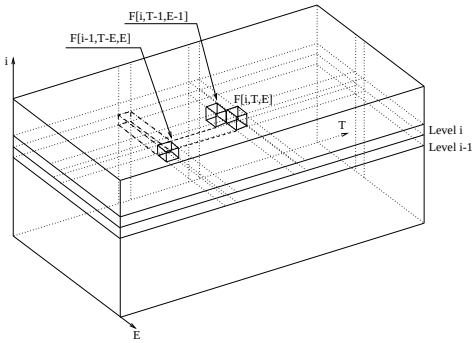


FIGURE 2. Dependencies of elements in the array in case of the optimized formula

$$
\begin{aligned}
f_i(T,E) &= \sum_{k=0}^{E} f_{i-1}(T-E,k) \\
&= \sum_{k=0}^{E-1} f_{i-1}((T-1)-(E-1),k) + f_{i-1}(T-E,E) \\
&= f_i(T-1,E-1) + f_{i-1}(T-E,E)
\end{aligned} \tag{2}
$$

Notice that when we compute $f_i(T,E)$ we already know $f_i(T-1,E-1)$ so we can replace the loop in lines 13–15 of Algorithm 5 with a simple assignment (see Figure 2).

4.3. **Parallel algorithms.**

4.3.1. *PRAM.* A straightforward parallel implementation of the non-optimized formula is the following. We fill in one level of the array in one round. We have $\frac{n}{\log n}$ processors for each element in the level (figure 3). These processors perform a prefix computation to calculate the value using the original formula (1). This takes $O(\log n)$ time. The array has n levels so the whole algorithm runs in $O(n \log n)$. On a single level of the array there are $n\binom{n}{2}$ elements, which means that we need $n\binom{n}{2}\frac{n}{\log n} = \frac{n^3(n-1)}{2\log n} = O(\frac{n^4}{\log n})$ processors to achieve this. Unfortunately this solution is not work-optimal as the amount of work done is $O(n^4 \log n) * O(n \log n) = O(n^5)$.

This algorithm used the property of (1) that the value of a particular element in a certain level depends on other elements from a lower level only. This way we could avoid the synchronization overhead between the processors working on different elements. Using the optimized formula we have to use results from the same level

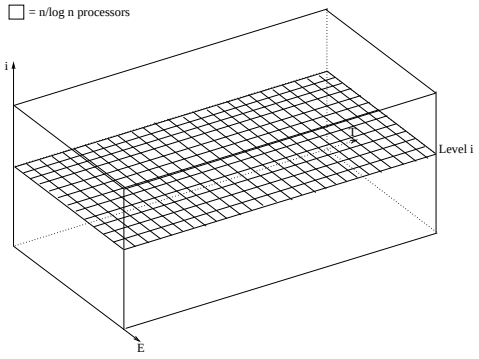


FIGURE 3. Using brute-force approach on PRAM architecture

as well. More accurately the value of $f_i(T, E)$ depends on $f_i(T-1, E-1)$ which in turn depends on $f_i(T-2, E-2)$ etc. This dependency limits the maximum number of processors that a work-optimal algorithm can utilize.

Here we present three possible work-optimal algorithms, using n, n^2 and $\frac{n^3-n}{2\lfloor \log n \rfloor}$ processors.

Each algorithm calculates the values level by level. The first version assigns a processor to each possible values of T and these processors calculate $f_i(T, E)$ for the different E values one by one. As the value of T belongs to the domain $[0..\binom{n}{2}]$ so we need $\binom{n}{2} + 1$ processors and each calculates $f_i(T, E)$ for $E = 0, \ldots, n-1$ which requires $\Theta(n)$ time. There are n levels in the array so the whole run time of the algorithm is $\Theta(n^2)$.

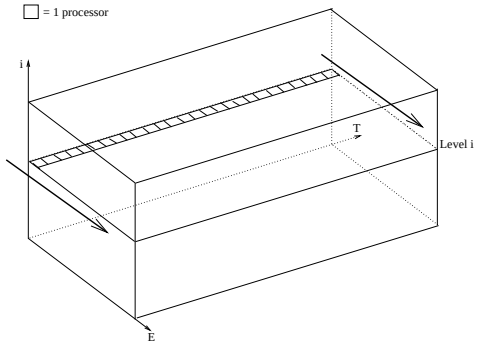


FIGURE 4. PRAM algorithm using n^2 processors

The second algorithm assigns processors to each possible values of E and these processors calculate $f_i(T, E)$ for the different T values one by one. This means

that we need n processors and due to symmetry the run time of this algorithm is $\Theta(n^3)$.

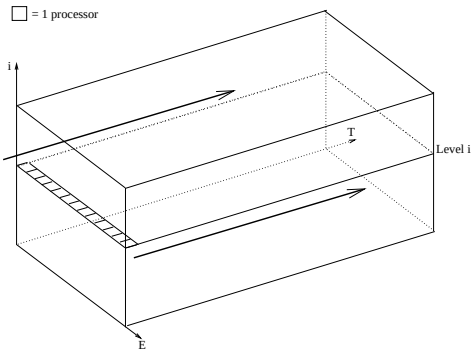


FIGURE 5. PRAM algorithm using n processors

The third algorithm uses a bit different approach. For this algorithm, computation of one level takes two steps. During the first step the processors set the elements of the level to 0. There are $\frac{n^3-n}{2}$ elements in a level, we have $\frac{n^3-n}{2\lceil \log n \rceil}$ processors so it takes $O(\log n)$ time to accomplish. In the second step the algorithm calculates $f_i(T+j, E+j), (j = 1..n)$ using prefix computation algorithm with $\frac{n}{\log n}$ processors on $f_{i-1}(T-E, j)$. This also takes $\log n$ time, so a single level can be calculated in $\log n$ time, the array has n levels so the whole algorithm works in $O(n \log n)$.

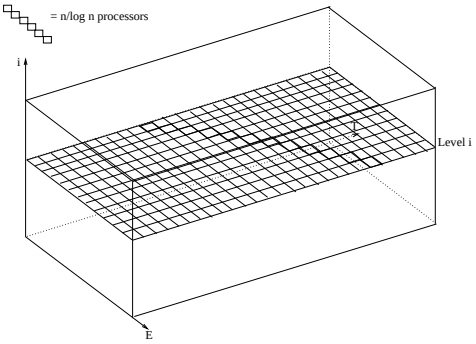


FIGURE 6. PRAM algorithm using $\frac{n^3}{\log n}$ processors

4.3.2. *Linear array.* The second work-optimal algorithm given for PRAM can be adapted to n processor linear array as well. Each processor is assigned a possible value of E. The processor stores the two-dimensional subarray belonging to that particular value . The processors use Algorithm 6.

Step 1: For i:=1 each processor calculates $F[i,T,E] := (T = E)?1 : 0$.
Step 2: For i:=2..n each processor performs Algorithm 7.
Step 3: The processors perform a prefix computation to determine t_n.

Algorithm 6: Enumeration of score sequences on n processor linear array

```
Each processor (E:=0..n-1) on level i (i:=2..n) does the following:
1  for T:=1 to n*(n-1)/2 loop
2      if E > 0 and T > 0) then
3          receive Z:=F[i,T-1,E-1] from processor E-1;
4      else
5          Z:=(i=2 and T=0 and E=0)?1:0;
6      end if;
7      if T-E ≥ ((i-1)*(i-2)/2) then
8          F[i,T,E]:=Z+F[i-1,T-E,E];
9      else
10         F[i,T,E]:=0;
11     end if;
12     if E < n-1 and T < n*(n-1)/2 then
13         send F[i,T,E] to processor E+1;
14     end if;
15 end loop;
```

Algorithm 7: Calculating $f_i(T,E)$ values on an n processor linear array

4.3.3. *Mesh.* As linear array can be embedded to a mesh the algorithm given in the previous section can be applied for meshes as well.

However there exists another work-optimal algorithm using n^2 processors. Let the processors be indexed from 1 to n (i) and from 0 to $n-1$ (E). Processor (i,E) has a one-dimensional subarray containing $f_i(T,E)$ for the possible different T values. This way to calculate $f_i(T,E)$ for a particular value of T it has to communicate with two of its neighbours.

The enumeration problem can be solved using the following algorithms:

4.3.4. *Hypercube.* As mesh can be embedded to a hypercube the same algorithms given for meshes can be applied.

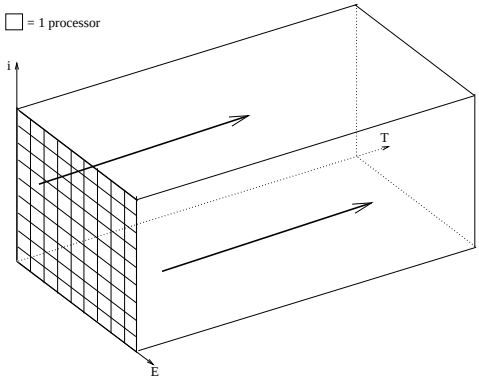


FIGURE 7. Mesh algorithm using n^2 processors

Step 1: For i:=1 and E:=0..n-1 each processor performs Algorithm 9. For i:=2..n and E:=0..n-1 each processor performs Algorithm 10.

Step 2: The processors $(n, 0), \ldots, (n, n-1)$ perform a prefix computation to determine t_n.

Algorithm 8: Enumeration of score sequences on n^2 processor mesh

5. Further optimizations

The algorithms given in the previous section use $\Theta(n^4)$ array elements. It's easy to see that each algorithm (except the last one) at a given time uses only two levels of the array. Calculating the ith level we need the (i-1)th one for that. This means that we don't have to store all levels only the current and the previous one. This optimization will reduce the number of necessary elements to $2 * n * n * (n-1)/2 = n^3 - n^2 = \Theta(n^3)$.

6. Conclusions

6.1. **Summary.** The table below summarizes our results for p processors and n-player tournaments:

Problem	**Sequential**	**Linear array**	**Mesh**	**Hypercube**	**PRAM**
Score sequence	$\Theta(n)$	$\forall p \in \mathbb{N}$ $\Theta(n)$	$p = n^{\frac{2}{3}}$ $\Theta(n^{\frac{1}{3}})$ work-opt.	$p = \frac{n}{\log n}$ $\Theta(\log n)$ work-opt.	$p = \frac{n}{\log n}$ $\Theta(\log n)$ work-opt.

<table>
<tr><td rowspan="3">Score vector</td><td rowspan="3">$\Theta(n)$</td><td rowspan="3">$\forall p \in \mathbb{N}$
$\Theta(n)$</td><td rowspan="3">$p = n$
$O(n^{\frac{1}{2}})$</td><td>$p = n$
$O(\log^2 n)$</td><td>$p = n$
$O(\log^2 n)$</td></tr>
<tr><td>$p = n^2$
$O(\log n)$</td><td>$p = n^2$
$O(\log n)$</td></tr>
<tr><td rowspan="3">Enumeration of score sequences</td><td rowspan="3">Recursive formula with dynamic programming:
$\Theta(n^4)$</td></tr>
<tr><td>$p = n$
$\Theta(n^3)$
work-opt.</td><td>$p = O(n)$
$\Theta(n^3)$
work-opt.</td><td>$p = O(n)$
$\Theta(n^3)$
work-opt.</td><td>$p = n$
$\Theta(n^3)$
work-opt.</td></tr>
<tr><td></td><td>$p = n^2$
$\Theta(n^2)$
work-opt.</td><td>$p = O(n^2)$
$\Theta(n^2)$
work-opt.</td><td>$p = n^2$
$\Theta(n^2)$
work-opt.</td></tr>
<tr><td></td><td></td><td></td><td></td><td>$p = \frac{n^3 - n}{2\lfloor \log n \rfloor}$
$\Theta(n \log n)$
work-opt.</td></tr>
</table>

The notion of completeness of tournaments can be extended to k-completeness.

k-**complete:** We call a tournament k-complete if its elements are non-negative integers and the sum of the symmetric elements is always k ($t_{ij} + t_{ji} = k$, where $i \neq j$) . A set of tournaments is called k-complete for a given n if it contains all possible n player k-complete tournaments.

From the definition it follows that a complete tournament is 1-complete. The theorems and algorithms presented above can be easily extended to k-complete tournaments.

6.2. **Future Works.** In this section we try to identify some possible directions to do further research.

- Fine-tuning the presented non work-optimal algorithms if possible or design new ones.
- As we saw the value of t_n increases exponentially this also implies that $f_i(T, E)$ values are increasing in similar order. Storing such values requires $O(n)$ bits. However it is possible that the average size of the elements in the array is smaller.
- The task of reconstruction means that for a given score sequence we construct a tournament. The asimptotically optimal sequential algorithms solve this problem in $\Theta(n^2)$ time. Parallel reconstructing algorithms for the problem are to be considered.
- Parallel algorithm for calculating the lexicographical successor of a given score sequence.
- Parallel listing of score sequences for a given n.

```
Each processor (i,E) (i:=2..n; E:=0..n-1) does the following:
1  for T:=0 to n*(n-1)/2 loop
2      if E>0 and T>0 then
3          receive Z:=F[i,T-1,E-1] from processor (i,E-1);
4      else
5          Z:=(i=2 and T=0 and E=0)?1:0;
6      end if;
7      if T-E≥((i-1)*(i-2)/2) then
8          if T=0 then
9              Y:=(i=2 and E=0)?1:0;
10         else
11             receive Y:=F[i-1,T-E,E] from processor (i-1,E);
12         end if;
13         F[i,T,E]:=Z+Y;
14     else
15         F[i,T,E]:=0;
16     end if;
17     if T<n*(n-1)/2 then
18         if E<n-1 then
19             send F[i,T,E] to processor (i,E+1);
20         end if;
21         send F[i,T-E,E] to processor (i+1,E);
22     end if;
23 end loop;
```

Algorithm 9: Calculating $f_i(T, E)$ values for $i > 2$

```
Each processor (1,E) (E:=0..n-1) does the following:
1  for T:=0 to n*(n-1)/2 loop
2      F[1,T,E]:=(E=T)?1:0;
3      if T<n*(n-1)/2 and T≥E then
4          send F[1,T-E,E] to processor (2,E);
5      end if;
6  end loop;
```

Algorithm 10: Calculating $f_1(T, E)$ values

The techniques that were used in the presented algorithms aimed the parallel adoption of a sequential dynamic programming solution. These techniques should be extended to other algorithms using dynamic programming.

Acknowledgement. The authors would like to thank Antal Iványi for sharing his knowledge about tournaments and being open to discuss our ideas.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest (1990), *Introduction to Algorithms*, McGraw-Hill, MIT Press, New York.

[2] E. Horowitz, S. Sahni, S. Rajasekaran (1998), *Computer Algorithms*, Computer Science Press, New York.

[3] A. Iványi, *Good tournaments*, submitted to Annales Univ. Sci. Budapest., Sectio Math.

[4] A. Iványi, *Maximal tournaments*, In: Fourth Join Conf. on Math. and Comp. Sci. Felix, June 5–10, 2001, submitted to Pure Math. and Appl.

[5] H. G. Landau (1953), *The condition for a score structure III*, Bull. Math. Biophysics, pp. 153–158.

[6] J. W. Moon (1968), *Topics on Tournaments*, Holt, Rinehart & Winston, New York.

[7] T. V. Narayana, D. H. Bent (1964), *Computation of the number of score sequences in round-robin tournaments*, Canad. Math. Bull. 7 (1), pp. 133–136.

[8] K. B. Reid (1996), *Tournaments: scores, kings, generalizations and special topics*, Congressus Numerantium 115, pp. 171–211.

Department of General Computer Science, Eötvös Loránd University, 1117 Budapest, Pázmány Péter sétány 1/B., Hungary

E-mail address: pici@elte.hu and slice@elte.hu

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

AN APPROXIMATE ALGORITHM TO ESTIMATE PLAUSIBLE LOCATIONS OF UNDISCOVERED HYDROCARBON ACCUMULATIONS IN SPARSELY DRILLED AREAS

GHEORGHE CIMOCA

Abstract. This paper uses concepts and principles pertaining to a natural geometrical data structure (the Voronoi diagram) in a theoretical attempt to estimate the sites and perimeters to really succeed in an exploration for undiscovered new hydrocarbon accumulations in an oil basin/system. The proposed algorithm can be applied to oil, methane gas or oil and gas combined reserves in a natural area of hydrocarbon accumulation, characterized by the same hydrocarbon source.

1. Introduction

The starting point of this mathematical experiment was a report [7], published by the Petroconsultants Group in 1993, on a new method for estimating undiscovered petroleum potential with applications to the giant oil fields of the world, such as: Arabo-Iranian basin, Campos basin, Gippsland basin, Kutei, South Sumatra, Niger delta, Timan-Pechora, North Sea grabens, Transylvania basin, as well as other petroleum systems. Their estimation is based on the best fit with fractal parabolas of oil field size distributions. Meanwhile, new methodologies meant to estimate the amount of undiscovered hydrocarbon reserves were announced in several reports ([11], [5]), whose results haven't been published so far.

However, a more "delicate" and obviously more difficult problem can be posed:

> "Is it possible to make forecasts/estimates, with a certain degree of plausibility, on the locations (sites, perimeters, extents, zones) of the "presumably existing" hydrocarbon accumulatios, not yet discovered in a sparsely drilled oil system?"

Using our knowledge of Voronoi diagrams ([3], [10]) which we had previously applied to some natural geological data structures (e.g., mineral deposits) we arrived at a first approach of the above problem. The most important question was:

2000 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I3.5. [**Computational Geometry** and **Object Modeling**] .

how to formulate, in mathematical terms, a location principle to find the "most plausible" sites and corresponding extents of new oil fields in a drilled area?

2. Voronoi diagrams: general concepts

Let X be a non-empty arbitrary set. A function $d : X \times X \to \mathbb{R}$ is said to be a *distance* or a *metric* on X if it satisfies the following conditions:

$$d(x, y) = 0 \iff x = y;$$

$$d(x, y) = d(y, x);$$

$$d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in X.$$

The pair (X, d) is called a *metric space*.

A simple example is the real plane $\mathbb{R}^2$ with the metric defined by:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \quad \forall x = (x_1, x_2), y = (y_1, y_2) \in \mathbb{R}^2.$$

This metric is called the *Euclidean* metric on $\mathbb{R}^2$. Let (X, d) be a metric space. A subset Y of X is said to be *bounded* (with respect to the metric) if

$$\sup \{d(x, y) | x, y \in Y\} < \infty.$$

Let Y be a non-empty subset of X and $x \in X$. The real number:

$$d(Y, x) = \inf \{d(y, x) | y \in Y\}$$

is called distance from x to Y. Let $M(X)$ be the set of all non-empty, bounded subsets of (X, d) and $M'(X)$ the set of all non-empty and closed subsets of (X, d). If $Y, Z \in M(X)$, then the real number:

$$e(Y, Z) = \sup \{d(Y, z) | z \in Z\}$$

is called *gauge* or *excess* of Y from Z.

If (X, d) is a metric space, then the function $\rho : M'(X) \times M'(X) \to \mathbb{R}$ defined by:

$$\rho(Y, Z) = \max \{e(Y, Z), e(Z, Y)\} \quad \forall Y, Z \in M'(X)$$

is a metric on $M'(X)$ [8]. This metric is called the *Pompeiu-Hausdorff* metric.

Suppose now that $S = \{C_1, \ldots, C_k\}$ is a finite set of distinct points in $\mathbb{R}^2$ and $f : \mathbb{R}^2 \times S \to [0, \infty[$ is a given function called the *influence* or *authorithy* function.

If $i \in \{1, \ldots, k\}$, then the subset of $\mathbb{R}^2$ defined by:

$$reg(C_i) = \{x \in \mathbb{R}^2 | f(x, C_i) \leq f(x, C_j), \forall j \in \{1, \ldots, k\} \setminus \{i\}\}$$

is said to be the *influence region* of C_i.

We call the set $\{reg(C_1), \ldots, reg(C_k)\}$ the *Voronoi diagram* generated by S with influence function f. In fact, the Voronoi diagram is a covering of the real plane by a set of regions associated with members of the point set S and an influence function f.

The sets $reg(C_i), i = \{1, \ldots, k\}$ are sometimes called *faces* of the Voronoi diagram. The intersection of two faces gives a *Voronoi edge* and the intersection of two edges is called a *Voronoi vertex.*

We'll denote by $Vor(S, f)$ the set consisting of all points of the edges of a Voronoi diagram generated by S and an influence function f.

If $i, j \in \{1, \ldots, k\}, i \neq j$, then the subset of $\mathbb{R}^2$ defined by:

$$sep(C_i, C_j) = \{x \in \mathbb{R}^2 | f(x, C_i) = f(x, C_j)\}$$

is called the *separation curve* of C_i and C_j, and the set:

$$dom(C_i, C_j) = \{x \in \mathbb{R}^2 | f(x, C_i) \leq f(x, C_j)\}$$

defines the *dominance region* of C_i over C_j.

The following relations hold true:

$$dom(C_j, C_i) = [\mathbb{R}^2 \setminus dom(C_i, C_j)] \cup sep(C_i, C_j),$$

$$reg(C_i) = \cap\{dom(C_i, C_j) | j \in \{1, \ldots, k\} \setminus \{i\}\}.$$

If the influence function f is the Euclidean metric d of $\mathbb{R}^2$, then the planar (ordinary) Voronoi diagram is obtained. In this case, $sep(C_i, C_j)$ is the perpendicular bisector m_{ij} between C_i and C_j, and $dom(C_i, C_j)$ is the half plane defined by m_{ij}, containing C_i. Therefore, being the intersection of $k - 1$ half planes, $reg(C_i)$ is a convex set.

When $f = d$, we call the region $reg(Ci)$ the (ordinary) *Voronoi polygon* associated with C_i, or the Voronoi polygon of C_i denoted $V(C_i)$. Since a Voronoi polygon is a closed set, it contains its boundary denoted by $\partial V(C_i)$. The term polygon is used to denote the union of the boundary and of the interior. The boundary of a Voronoi polygon may consist of line segments, half lines or infinite lines, which we call *Voronoi edges.* Alternatively, we may define a Voronoi edge as a line segment, a half line or an infinite line shared by two Voronoi polygons.

If $V(C_i) \cap V(C_j) \neq \emptyset$, then the set $V(C_i) \cap V(C_j)$ gives a Voronoi edge which may degenerate into a point. If $V(C_i) \cap V(C_j)$ is neither empty nor a point, we say that the Voronoi polygons $V(C_i)$ and $V(C_j)$ are *adjacent.*

For the sake of simplicity, if $f = d$, instead of $Vor(S, f)$ we write $Vor(S) = \partial V(C_i) \cup \ldots \cup \partial V(C_k)$.

Now let A be a closed subset of $\mathbb{R}^2$ and $\mathfrak{T} = \{T_1, \ldots, T_k\}$, where each $T_i, i \in \{1, \ldots, k\}$ is a closed subset of A. If the elements of the set $\mathfrak{T}$ satisfy $[T_i \setminus \partial T_i] \cap [T_j \setminus \partial T_j] = \emptyset, \forall i, j \in \{1, \ldots, k\}, i \neq j$, then we call the set $\mathfrak{T}$ a *pretessellation* of A. A pretessellation $\mathfrak{T}$, where all $T_i, i \in \{1, \ldots, k\}$ are convex sets is called a *convex pretessellation.*

A pretesselation $\mathfrak{T} = \{T_1, \ldots, T_k\}$ with $A = \cup\{T_i | i = 1, \ldots, k\}$ becomes a *tesselation.* A planar Voronoi diagram is a tessellation which consists of convex polygons with three or more vertices. A planar tessellation in which any T_i in $\mathfrak{T}$

is a triangle $\forall i \in \{1, \ldots, k\}$ is called a *triangulation* of A. Two vertices sharing an edge in a triangulation are called adjacent.

Given a planar Voronoi diagram where generators are not colinear and their number is three or more, but finite, we join all pairs of generators whose Voronoi polygons share a common Voronoi edge, thus obtaining a new tessellation. If the new tessellation consists only of triangles, we call it a *Delaunay triangulation*; otherwise, we call it a *Delaunay pretriangulation*. In the case of the Delaunay pretriangulation, we partition the non-triangular polygons into triangles by non-intersecting line segments joining the vertices. As a result, the Delaunay pretriangulation becomes a Delaunay triangulation.

3. Locational mathematical model

Let's consider an oil system (or a geological-tectonical region) externally delimited, on a geographical map, by the boundary of a simple polygon A. Suppose that in this oil system k oil fields have been discovered.

Let the points $C_1, \ldots, C_k$ be the centers/sites/domes and let the simple polygons $\wp_1, \ldots, \wp_k, C_i \in \wp_i \subset A, i \in \{1, \ldots, k\}$ be the extents/contours of these fields, being situated on the same map. Moreover, $\wp_1 \cap \ldots \cap \wp_k = \emptyset$.

We consider the set $S = \{C_1, \ldots, C_k\}$. In addition, let's denote $B_i = \partial\wp_i, i \in \{1, \ldots, k\}$ and $P = \cup\{\wp_i | i = 1, \ldots, k\}$.

Now, we formulate the following question: where, in this region A, can the centers of a given number of new, posible oil fields be most plausibly located ?

In order to get an answer it is important to restate and formalize the above verbal problem more precisely, in mathematical terms. With that end in view, let d be the Euclidean metric on $\mathbb{R}^2$, M' the set of all non-empty closed subsets of $\mathbb{R}^2$ and ρ the Pompeiu-Hausdorff metric on M'.

In order to mathematically formalize the locational problem, we must adopt an essential assumption:

Assumption.

$$B_1 \cap \ldots \cap B_k = Vor(S) \cap A$$

If the above assumption is correct, then we believe the most plausible location of the centers of m undiscovered oil fields in the oil system A leads to the following optimization problem:

Location problem. Find m points $C_{k+1}, \ldots, C_{k+m}$ in $A \setminus P$ such that:

$$\rho(Vor(S \cup \{C_{k+1}, \ldots, C_{k+m}\}), B_1 \cup \ldots \cup B_k) =$$

$$\min\{\rho(Vor(S \cup S'), B_1 \cup \ldots \cup B_k) | S' \in \Sigma\}$$

where:

$$\Sigma = \{S' \subseteq A \setminus P | card(S' \setminus S) = m\}.$$

Remarks.

(1) In fact, the above location problem is to determine within a simple polygon A the locations of a given number (m) of points, outside of a pretessellation of A (in $A \setminus P$), so that the Pompeiu-Hausdorff distance between two Voronoi diagrams having some common generators is minimized. The distance can be defined as the sum of Pompeiu-Hausdorff distances between the pairs of Voronoi polygons [2] with common generators.

(2) A couple of location problems are similar, although much easier than ours: the recognition of a Dirichlet (Voronoi) tessellation [1], [12] and the geographical optimization problem from [6]. These problems start from a convex tessellation. By contrast, we start from a more general, non convex pretesselation, denoted in the following by (A, S, P).

4. The approximate algorithm

Being fully aware of the difficulty of the above location problem, we have tried to find only an approximate solution. Our approximate algorithm is of an incremental type [9] and uses some remarks on distortions of a Voronoi diagram when one point moves [4].

Let $S = \{C_1, \ldots, C_k\}$ be the set of sites/centers of discovered oil fields and an arbitrary point $C_0 \in A \setminus P$. In the following, we denote by $V(i), i = 1, \ldots, k$, the Voronoi polygon of C_i in the Voronoi diagram generated by S and by $V_0(i)$ the Voronoi polygon in the Voronoi diagram generated by $S_0 = S \cup \{C_0\}$. Let $\mathfrak{T}_0$ be the Delaunay triangulation of the set S_0.

For $C_i \in S$, we have $V(i) = V_0(i)$, if and only if C_i and C_0 are not adjacent vertices in $\mathfrak{T}_0$ [4]. Moreover, if C_i and C_0 are adjacent and $\wp_i \subset V(i)$, it doesn't follow that $\wp_i \subset V_0(i)$.

We say that the "center" C_0 is admissible in pretessellation (A, S, P) in respect to $\mathfrak{T}_0$, if for every $C_i \in S$, such that C_i and C_0 are adjacent vertices in $\mathfrak{T}_0$, then $\wp_i \subset V_0(i)$.

If $C_p, C_q \in A \setminus S, C_p \neq C_q$, let us denote by $V_p(i)$, respectively by $V_q(i)$, the Voronoi polygon of $C_i \in S$ in $Vor(S_p)$, respectively $Vor(S_q)$, where $S_p = S \cup C_p$ and $S_q = S \cup C_q$. Let $\mathfrak{T}_p$, respectively $\mathfrak{T}_q$ be the Delaunay triangulations of S_p, respectively S_q. Moreover, let $\mathring{A}(p)$ be the set of points in S which are adjacent with C_p in $\mathfrak{T}_p$, and $\mathring{A}(q)$ the points in S adjacent with C_q in $\mathfrak{T}_q$.

Let C_p and C_q be two admissible centers in (A, S, P) corresponding to $\mathfrak{T}_p$ and $\mathfrak{T}_q$, respectively. We say that C_p is preferred to C_q if

$$\pi_p = \sum_{C_i \in \mathring{A}(p)} \rho(V_p(i), V_p(p)) \leq \sum_{C_i \in \mathring{A}(q)} \rho(V_q(i), V_q(q)) = \pi_q,$$

where ρ is the Pompeiu-Hausdorff distance.

The number π_p evaluates the distortion effect of the point C_p on the Voronoi diagram generated by S. At the same time, π_p represents a measure of plausibility. The smaller π_p, the more plausible C_p.

Let $G(p,p)$ be a uniform rectangular grid with sides parallel to the coordinate axes, which contains A, and the number p of horizontal and vertical grid lines an even integer.

The algorithm. Step 1. Let p be the smallest positive even integer such that $m < p^2$ and let $G(p,p)$ be the minimal uniform rectangular grid covering A.

Let $n := 0$ and $W^0 := S$.

Step 2. Scan the grid $G(p,p)$ rectangle-by-rectangle, in a spiral order, starting from the central rectangle of the grid. For each rectangle D_{q+1} execute the following operations:

a. Let $C_0 :=$ the center of D_{q+1};

b. Construct the Delaunay triangulation $\mathfrak{T}_*$ of the set $W^* = W^q \cup \{C_0\}$. We distinguish the cases:

Case I. If there exists a point $C_i \in S$ which is adjacent to C_0 in $\mathfrak{T}_*$ and $C_0 \in \wp_i$, take the next rectangle.

Case II. If $C_0 \notin P$, choose the most preferred point C^* between C_0 and each of the four rectangle corners which are not in P. Let $W^{q+1} := W^q \cup \{C^*\}$ and $n := n+1$. If $n = m$ **stop**, else go to **Step 1** with $p := 2p$.

Remarks.

(1) In fact, this algorithm locates a m-points planar configuration in a pretessellation, each of the m points having only a local plausibility. This configuration can be a starting point pattern for further, more subtle, improved algorithms.

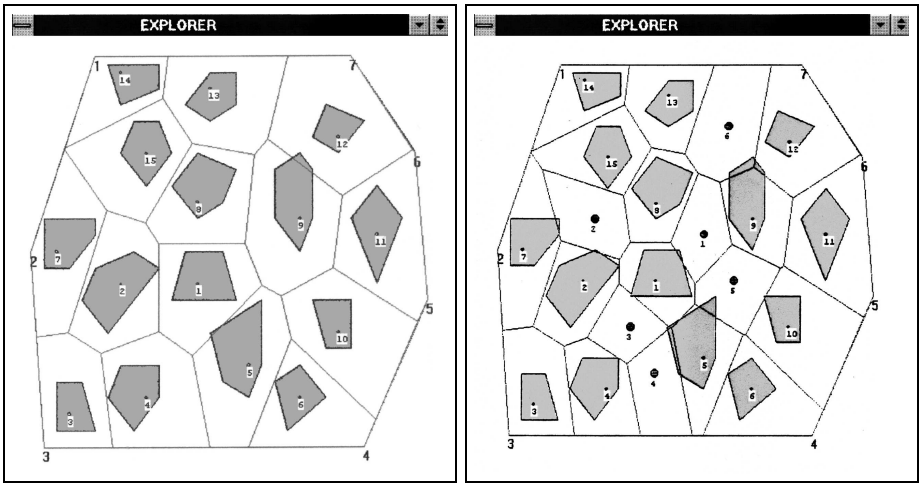
(2) Mathematically, it is easier to insert "new" admissible oil fields closer to the boundary of A, but we have preferred a more "central" configuration for geological reasons.

(3) The algorithm can be relativized to a subzone of A, called zone of geological interest.

A software package named *EXPLORER* has been developed and tested on both non- and real data. *EXPLORER* enables users to:

- visualize a basin in study with all its fields,
- visualize a particular field and its contour,
- visualize a field and its adjacent neighbours,
- visualize the Voronoi diagram of a basin,
- locate a given number of new plausible oil fields and their possible extents, and
- print the founded pattern of "new" and old fields.

In the following figures we present an example of *EXPLORER* outputs for a fictional basin with 15 active oil fields, their contours or extents (gray polygons) and the Voronoi diagram of these fields (left); the forecasted sites of 6 possible new fields (circled dots) and their plausible (in decreasing rank order, #1 having the highest degree of plausibility) extents as their Voronoi polygons (right).



5. An experiment for Transylvania Basin

The *EXPLORER* software application allows the user to "discover" several "new" oil fields as well, in a real oil basin, **Transylvania**, with different plausibilities (depending on the number of scanned rectangles).

We can comunicate, to whom may be interested, two tested results:

(1) Using the information regarding the 23 active fields discovered in the Transylvania basin during 1906-1965, the algorithm proposed 20 "new" locations of methane gas fields. We were surprised to find out that 15 out of these fields were "confirmed" (their extents having a non-empty intersection with at least one contour of an actually discovered field) during 1966-1985 (out of the 29 new fields actually pointed out during this period). Furthermore, 4 more fields were confirmed during 1986-1996 (out of 52 new actually discovered fields). Therefore, 19 out of the 20 sites proposed by algorithm have been confirmed.

(2) Forecasting again 20 possible locations of "new" fields by means of the methane gas field pattern existing in 1985 (i.e., 52 active fields), 17 fields were confirmed during 1986-1996.

6. Conclusions

We are aware that the development of a new method/technology to mathematically forecast the sites and/or extents of new oil fields in an oil system needs a strong collaboration between oil geologists, mathematicians and computer engineers. The above algorithm is just a first step toward a new technology. Algorithm's forecasts zones of possible hydrocarbon accumulations require confirmations by geological parameters. But these forecast perimeters, we believe, are the most plausible locations for the new possible oil fields in an oil basin.

By superposing quantitative geological parameter (e.g., permeability, porosity, pressure, &c.) maps, on this prognosticated locations the exploration expenses and time can be drastically diminished. As the tested results on a real oil basin indicate, we are optimistic and forsee a successful completion (new natural geometrical data structures generated by an influence function $f \neq d$; new location principles; new improved algorithms) of this promising research.

Acknowledgments. I am deeply grateful to Tiberiu Trif, from the Faculty of Mathematics and Computer Science, the Babeş-Bolyai University, Cluj-Napoca, Mihaela Ordean and Ovidiu Pop, from the Computer Science Department, the Technical University of Cluj-Napoca, who contributed their expertise to the development of the *EXPLORER* software application.

References

[1] P.F. Ash, E.D. Bolker, *Recognizing Dirichlet tessellations*, Geometriae Dedicata, 19 (1985), pp. 175-206.

[2] M.J. Attallah, *A linear time algorithm for the Hausdorff distance between convex polygons*, Inf. Proc. Let., 17 (1983), pp. 207-209.

[3] F. Aurenhammer, *Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure*, ACM Comput. Surveys, 23 (1991), pp. 345-405.

[4] L.-M. Cruz Orive, *Distortion of certain Voronoi tessellations when one particle moves*, J. Appl. Prob., 16 (1979), pp. 95-103.

[5] C.C. Barton, G.L. Troussov, *Fractal Methodology for Petroleum Resource Assessment and Fra-A Computer Program That Calculates the Volume and Number of Undiscovered Hydrocarbon*, U.S. Geological Survey; announced in 1997.

[6] M. Iri, K. Murota, T. Ohya, *A Fast Voronoi Diagram Algorithm with Applications to Geographical Optimization*, in Lecture Notes in Control and Information Sciences 59 , Springer Verlag, Berlin, 1984, pp. 273-288.

[7] J. Laherrere, A. Perrodon, G. Demaison, *Undiscovered Petroleum Potential; A new approach based on distribution of ultimate resources*, Petroconsultants S.A., Geneva-London-Houston-St.Leonards-Singapore-Victoria Park, 1994.

[8] I. Muntean, *Analiză funcţională*, Universitatea "Babeş-Bolyai", Cluj-Napoca, 1993.

[9] T. Ohya, M. Iri, K. Murota, *Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparison of Various Algorithms*, J. Operations Res. Soc. Japan, 27 (1984), pp. 306-336.

[10] A. Okabe, B. Boots, K. Sugihara, *Spatial Tessellations; Concepts and Applications of Voronoi Diagrams*, J.Wiley & Sons Ltd, London, 1992.

[11] R.G. Stanley, *The "Checkerboard Method": A New Way to Estimate the Numbers of Undiscovered Hydrocarbon Accumulations in Sparsely Drilled Areas*, U.S. Geological Survey; announced in 1995; not yet published.

[12] A. Suzuki, M. Iri, *Approximation of a tessellation of the plane by a Voronoi diagram*, J. Operations Res. Soc. Japan, 29 (1986), pp. 69-96.

S.C. SIMBOLIC, STR. BACĂU NR. 3, 3400 CLUJ-NAPOCA, ROMANIA, PHONE: +40-64-431.333, FAX: +40-64-199.895

E-mail address: ghcimoca@symbolic.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

A MODIFICATION OF THE TSENG-JAN GROUP SIGNATURE SCHEME

CONSTANTIN POPESCU

Abstract. In this paper we present a modification of the Tseng-Jan group signature scheme [18]. Our scheme appears to be secure in comparation with the Tseng-Jan group signature scheme. The proposed scheme is based on the e-th root problem and the discrete logarithm problem. *Keywords*: Group signature, identity, membership certificate.

1. Introduction

Group signatures allow individual members of a group to sign messages on behalf of the group while remaining anonymous. Furthermore, in case of disputes later a trusted authority, who is given some auxiliary information, can identify the signer. The concept of group signatures was introduced by Chaum and van Heyst [4]. Their schemes have been improved by L. Chen and T. Pedersen [5], who first use a Schoenmaker's protocol [17] to hide a signer's identity. Also, H. Petersen suggested a general method to convert any ordinary digital signature into a group signature scheme [15]. Petersen's method combines the Stadler's verifiable encryption of discrete logarithm [18] and the Schoenmaker's protocol. J. Camenisch and M. Stadler presented the first group signature scheme whose public key and signatures have length independent of the number of group members of one group [2], but this isn't independent of the number of groups. Many group signature schemes have been presented [3], [7], [8], [12], [13], [14], [16]. In [19], Tseng and Jan proposed a group signature scheme, but this was broken in [9] and [10]. In [9], M. Joye, S. Kim and N. Lee showed that the Tseng-Jan scheme is universally forgeable, that is, anyone is able to produce a valid group signature on an arbitrary message. In [10], M. Joye showed that the group signature scheme proposed by Tseng-Jan is not coalition-resistant: two group members can produce untraceable group signatures.

In this paper we present a modification of the Tseng-Jan group signature scheme [19]. Our scheme appears to be secure in comparation with the Tseng-Jan group

2000 *Mathematics Subject Classification.* 94A60.

1998 *CR Categories and Descriptors.* D.4.6. [**Software**]: Operating Systems – *Security and Protection*;

signature scheme. The proposed scheme is based on the e-th root problem and the discrete logarithm problem. The remainder of the paper is organized as follows. In Section 2, we review the scheme proposed by Tseng and Jan. In Section 3 our scheme is described. In Section 4 some security considerations are given and finally, Section 5 concludes with the results of the paper.

2. Tseng-Jan Group Signature Scheme

In this section, we give a short description of the Tseng-Jan group signature scheme and refer to the original paper [19] for more details. The scheme involve four parties: a trusted authority, the group authority, the group members, and verifiers. The trusted authority acts as a third helper to setup the system parameters. The group authority selects the group public/secret keys. He (jointly with the trusted authority) issues membership certificates to new users who wish to join the group. In case of disputes, opens the contentious group signatures to reveal the identity of the actual signer. Finally, group members anonymously sign on group's behalf using their membership certificates and verifiers check the validity of the group signatures using the group public key.

In order to set up the system, a trusted authority selects two large prime numbers $p_1 (\equiv 3 \bmod 8)$ and $p_2 (\equiv 7 \bmod 8)$ such that $(p_1 - 1)/2$ and $(p_2 - 1)/2$ are smooth, odd and co-prime [11]. Let $N = p_1 p_2$. The trusted authority also defines e, d, v, t satisfying $ed \equiv 1 \pmod{\varphi(N)}$ and $vt \equiv 1 \pmod{\varphi(N)}$, selects g of large order in $\mathbb{Z}_N^*$, and computes $F = g^v \pmod N$. Moreover, the group authority chooses a secret key x and computes the corresponding public key $y = F^x \pmod N$. The public parameters are (N, e, g, F, y). The secret parameters are (p_1, p_2, d, v, t, x).

When a user U_i (with identity information D_i) wants to join the group, the trusted authority computes

$$s_i = et \log_g ID_i \pmod{\varphi(N)}$$

where $ID_i = D_i$ or $ID_i = 2D_i$ according to $(D_i \mid N) = 1$ or $(D_i \mid N) = -1$, and the group authority computes

$$x_i = ID_i^x \pmod N .$$

The user membership certificate is the pair (s_i, x_i). To sign a message M, the user U_i (with certificate (s_i, x_i)) chooses two random numbers r_1 and r_2 and computes

$$\begin{aligned} A &= y^{r_1} \pmod N \\ B &= y^{r_2 e} \pmod N \\ C &= s_i + r_1 h(M \parallel A \parallel B) + r_2 e \\ D &= x_i y^{r_2 h(M \| A \| B)} \pmod N \end{aligned}$$

where $h(\cdot)$ is a publicly known hash function. The group signature on message M is given by the tuple (A, B, C, D). The validity of this signature can then be

verified by checking whether

$$D^e A^{h(M\|A\|B)} B \equiv y^C B^{h(M\|A\|B)} \pmod{N}.$$

Finally, in case of disputes, the group authority can open the signature to recover who issued it by checking which identity ID_i satisfies

$$ID_i^{xe} \equiv D^e B^{-h(M\|A\|B)} \pmod{N}.$$

3. Our Group Signature Scheme

This section describes the proposed group signature scheme, which is specified by the key generation, signing messages, verification signatures and opening signatures.

3.1. **Key Generation.** Our scheme consists of four kinds of participants: a *trusted center* who setup the system parameters, a *group authority* who issues membership certificates to new users who wish to join the group and identifies a signer, a *signer* for issuing group signatures and a *receiver* for verifying them using the group public key.

A trusted center selects two large primes p_1, p_2 as in [19]. Let $n = p_1 p_2$. The trusted center also selects a large integer e (160 bits) with $\gcd(e, \varphi(n)) = 1$ and selects g of large order in $\mathbb{Z}_n^*$, where $\mathbb{Z}_n$ is the integer ring. The group authority chooses a secret key x and computes the corresponding public key $y = g^x \pmod{n}$. The public parameters are (n, e, g, y) and the secret parameters are (p_1, p_2, x). Let $ID_i \in \mathbb{Z}_n$ be an identity information of a user U_i. Finally, let h be a collision-resistant hash function. Suppose now that a user wants to join the group. We assume that communication between the user and the trusted center (between the user and the group authority) is secure, i.e., private and authentic.

When a user U_i wants to join the group, the trusted center computes

$$s_i = ID_i^{\frac{1}{e}} \pmod{n}$$

and the group authority computes

$$x_i = (ID_i + eg)^x \pmod{n}.$$

The user membership certificate is the pair (s_i, x_i).

3.2. **Signing Messages.** To sign a message M, the user U_i, with certificate (s_i, x_i), chooses two random numbers r_1 and r_2 and computes

$$A = y^{r_2 e} \pmod{n}$$

$$B = x_i y^{s_i + r_1} \pmod{n}$$

$$C = x_i y^{r_2} \pmod{n}$$

$$D = s_i h(M \parallel A) + r_1 h(M \parallel A).$$

The symbol $\|$ denotes the concatenation of two binary strings (or of the binary representation of group elements and integers). The group signature on message M is given by the tuple (A, B, C, D).

3.3. **Verification Signatures.** The validity of this signature can then be verified by checking whether

$$C^{eh(M\|A)} y^{eD} \equiv B^{eh(M\|A)} A^{h(M\|A)} \pmod{n}.$$

If this equation holds, he accepts the signature (A, B, C, D), otherwise it is rejected.

3.4. **Opening Signatures.** Finally, in case of disputes, the group authority can open the signature to recover who issued it by checking which identity ID_i satisfies

$$(ID_i + eg)^{xe} \equiv C^e A^{-1} \pmod{n}.$$

4. Security Considerations

A receiver, a group authority and a trusted center, who have no membership certificate (s_i, x_i) of a user U_i, can not generate a group signature. Trusted center knows s_i, but he can not determine x_i, because only the group authority knows the secret key x. The group authority knows x_i, but he can not determine s_i, because only the trusted center knows the e-th root of ID_i.

Given a group signature (A, B, C, D), identifying the actual signer is computationally hard for every one but the group authority.Since no one knows which pair (s_i, x_i) corresponds to which group member, anonymity is guaranteed.

Deciding whether two different signatures are computed by the same group member is computationally hard. The problem of linking two signatures (A, B, C, D) and (A', B', C', D') reduces to looking if either s_i or x_i is common to the two tuples. This is however impossible under Decisional Diffie-Hellman Assumption (see [1], [6]).

Trusted center and a receiver can not determine a signer of the group signature, because only the group authority knows the secret key x. If p_1 and p_2 are sufficiently large, even trusted center can not get x from the public key y. Therefore, an adversary can not forge our group signature scheme on an arbitrary message M.

5. Conclusions

This paper has presented a modification of the Tseng-Jan group signature scheme proposed in [19]. Our scheme appears to be secure in comparation with the Tseng-Jan group signature scheme. The security of the proposed scheme depends on the e-th root problem and the discrete logarithm problem.

References

[1] D. Boneh, *The decision Diffie-Hellman problem*, In Algorithmic Number Theory (ANTS-III), Lecture Notes in Computer Sciences 1423, Springer-Verlag, pp. 48-63, 1998.

[2] J. Camenisch, M. Stadler, *Efficient group signature schemes for large groups*, Advances in Cryptology, CRYPTO'97.

[3] J. Camenisch, M. Michels, *A group signature scheme based on RSA-variant*, BRICS, Denmark, 1998.

[4] D. Chaum, E. Heyst, *Group Signatures*, Advances in Cryptology, EUROCRYPT'91, Lecture Notes in Computer Sciences 950, Springer-Verlag, 1992, pp. 257-265.

[5] L. Chen, T. Pedersen, *New group signature schemes*, Advances in Cryptology, EUROCRYPT'94, Lecture Notes in Computer Sciences 547, Springer-Verlag, 1995, pp. 163-173.

[6] W. Diffie, M. Hellman, *New Directions in Cryptography*, IEEE Transaction Information Theory, IT-22, 6, pp. 644-654, 1976.

[7] S. Kim, S. Park, D. Won, *Group signatures for hierarchical multigroups*, Information Security Workshop, Lecture Notes in Computer Sciences 1396, Springer-Verlag, 1998, pp. 273-281.

[8] S. Kim, S. Park, D. Won, *Convertible Group Signatures*, Advances in Cryptology, ASIACRYPT'96, Lecture Notes in Computer Sciences 1163, Springer-Verlag, 1996, pp. 311-321.

[9] M. Joye, S. Kim, N. Lee, *Cryptanalysis of Two Group Signature Schemes*, 1999 (5 pages).

[10] M. Joye, *On the Difficulty of Coalition-Resistance in Group Signature Schemes*, Technical Report, LCIS-99-6B, 1999.

[11] U. Maurer, Y. Yacobi, *Non-interactive public-key cryptography*, In Advances in Cryptology-EUROCRYPT'91, LNCS 547, Springer-Verlag, 1991, pp. 498-507.

[12] S. Park, I. Lee, D. Won, *A practical group signature*, Proc. of JWISC'95, Japan, 1995, pp. 127-133.

[13] S. Park, D. Won, *A practical identity-based group signature*, Proc. of ICEIC'95, China, 1995, pp. II-64-II-67.

[14] S. Park, S. Kim, D. Won, *ID-based group signature schemes*, Electronics Letters, 1997, pp. 1616-1617.

[15] H. Petersen, *How to convert any digital signature scheme into a group signature scheme*, In Security Protocols Workshop, Paris, 1997.

[16] C. Popescu, *Group signature schemes based on the difficulty of computation of approximate e-th roots*, Proceedings of Protocols for Multimedia Systems (PROMS 2000), Poland, pp. 325-331, 2000.

[17] B. Schoenmakers, *Efficient Proofs of Or*, Manuscript, 1993.

[18] M. Stadler, *Publicly verifiable secret sharing*, Advances in Cryptology, EUROCRYPT'96, Lecture Notes in Computer Sciences 1070, Springer-Verlag, 1996, pp. 190-199.

[19] Y. Tseng, J. Jan, *A novel ID-based group signature*, In T.L. Hwang and A.K. Lenstra, editors, 1998 International Computer Symposium, Workshop on Cryptology and Information Security, Tainan, 1998, pp. 159-164.

University of Oradea, Department of Mathematics, Str. Armatei Romane 5, Oradea, Romania

E-mail address: cpopescu@math.uoradea.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

A METHOD FOR TRAINING INTELLIGENT AGENTS USING HIDDEN MARKOV MODELS

GABRIELA ŞERBAN

Abstract. It is well-known that, in this moment, the field of *intelligent agents* represents an important research direction in Artificial Intelligence, which offers a new method for problem solving and a new way for interaction between the computer and the user. The use of mathematical statistic-methods represents a leading topic in this field. Hidden Markov Models (HMM) are often used as a mathematical tool for modeling the environment of intelligent agents. In this paper we propose a learning method for an agent which recognize characters, based on training an Hidden Markov Model.

Keywords: Artificial Intelligence, Hidden Markov Models, learning.

1. Intelligent agents

The field of *intelligent agents* is in connection with another field of Artificial Intelligence (AI), the field of *machine learning*. *Machine learning* represents the study of system models that, based on a set of data (training data), improve their performance by experiences and by learning some specific experimental knowledge. The attempt of modeling the human reasoning leads to the concept of *intelligent reasoning*. The *reasoning* is the process of conclusion deduction; the *intelligent reasoning* is a kind of reasoning accomplished by humans. Most of the AI systems are deductive ones, able for making inferences (draw conclusions), given their initial or supplied knowledge, without being able for new knowledge acquisition or to generate new knowledge. The learning capability being connected to the intelligent behavior, one of the most important research directions in AI is to implement in the machines the learning capability.

An *agent* [3] is anything that can be viewed as **perceiving** its environment through *sensors* and **acting** upon that environment through *actions*. An *intelligent agent* is an agent with an initial knowledge, having the capability for learning. In the

2000 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I.2.6. [**Computing Methodologies**] : Artificial Intelligence – *Learning* .

followings we present how an agent can be modeled using a Hidden Markov Model, and how the agent can be trained by learning the associated HMM.

2. Hidden Markov Model (HMM)

The Hidden Markov Model (HMM) is a generalization of Markov decision processes, being possible more transitions from a state for the same input. For the same input sequence (of actions) we can have more paths in the HMM, which implies that $P(a_{1,n})$ (the probability to have as input a sequence of n actions, $a_1\, a_2\, \cdots a_n$, shortly written as $a_{1,n}$) is calculated as the sum of the probabilities on all the possible paths. Probability on a given path is calculated by multiplying the probabilities of transitions on the path.

Definition.
An HMM is a 4-tuple $< s^1, S, A, P >$, where S is a finite set of states, $s^1 \in S$ is the initial state, A is a set of input symbols (actions), and $P : SXSXA- > [0,1]$ gives the probability of moving from state s_1 to s_2 on performing action a. Let us consider the following order of the elements of the sets S, A, P: $S = (s^1, \cdots s^\sigma)$; $A = (a^1, \cdots a^\omega)$; $P = (p^1, \cdots p^\epsilon)$., where σ is the number of states, ω is the number of actions and ϵ is the number of transitions.

Let us notice that [4] a_i means the i-th element (action) of an input sequence, while the a^i represents the i-th element of the A set. A transition is defined as a 4-tuple: (s^i, s^j, a^k, p) , which means that the input action a^k in the state s^i transitions to the state s^j with the probability p. For a given input sequence of actions there are more possible paths in the HMM, so, the sequence of states that it has been passed through is not deductible from the input, but *hidden* (this gives the name of the model). The sequence of states $s_1, s_2, \cdots, s_{n+1}$ that has been passed through for an input $a_{1,n}$ is marked shortly with $s_{1,n+1}$.

2.1. **Agents and Hidden Markov Models.** Let us consider a passive learning agent in a known environment represented as a set of *states*. At each moment, the agent executes an *action*, from a set of actions. In such a passive learning model, the environment generates transitions between states, perceived by the agent. The agent has a model of the environment using a *model of actions* (P), where $P(x'|x, a)$ represents the probability for reaching the state x' by taking the action a in the state x.

With the above considerations, the behavior of the agent in a given environment can be seen as a Markov decision process. If the state transitions are nondeterministic (a given action a in a given state x transitions to a set of *successor states*, not to a single successor state), then the Markov model is an HMM where:

- S is the set of the environment states;

- $s^1 \in S$ is the initial state for the agent;
- A is the set of the actions of the agent;
- P is the set of transitions between the states (conditioned by actions).

2.2. **Algorithm for computing the likelihood of an input sequence of actions.** In the followings, we mention a very simple algorithm for computing the likelihood of an input sequence of actions in an HMM. This algorithm, the *"forward"* algorithm [1] calculate the probability of an input sequence of actions $(a_{1,n})$ using the *"forward" probability* (α) and the *"backward" probability* (β).

The *"forward" probability* is defined [1] as the probability of being in state i after seeing the first t observations, given the input sequence.

Let us note by $\alpha_i(t+1)$ the probability of the input sequence $a_{1,t}$ having s^i as final state. In other words:

$$\alpha_i(t+1) = P(a_{1,t}, s_{t+1} = s^i), t > 0 \tag{1}$$

The idea of the algorithm is to calculate the probabilities for all the input subsequences $(a_{1,t}, t = 0, \cdots, n)$ having as final state the state s^i, $i = 1, \cdots, \sigma$, where σ is the total number of states of the Markov model. Having all $\alpha_i(n+1)$ values calculated, the probability $P(a_{1,n})$ is given by:

$$P(a_{1,n}) = \sum_{i=1}^{n} \alpha_i(n+1)$$

Considering that $a_{1,0}$ is the empty sequence, which has the acceptance probability 1, we have that $\alpha_j(1) = 1$ if $j = 1$ and is 0 otherwise , corresponding to the fact that the initial state of every path is s^1.

Using the dynamic programming principle, we can make the following remark: the probability of the input sequence $a_{1,t+1}$ having s^j as final state, is obtained by summing for all state s^i, $i = 1, \cdots, \sigma$ the products between the probability of the input sequence $a_{1,t}$ having s^i as final state and the probability of the transition between the state s^i and the state s^j for the action a_t. Thus, calculation of $\alpha_j(t)$ [4] is made starting with $\alpha_j(1)$, $\alpha_j(2)$ and going until $\alpha_j(n+1)$, using the recursive relation:

$$\alpha_j(t+1) = \sum_{i=1}^{\sigma} \alpha_i(t) P(s^i \xrightarrow{a_t} s^j).$$

Recall that $\alpha_i(t)$ are called "*forward*" probabilities. Using the above considerations, let us notice that the algorithm for finding the highest-probability-paths for a given entry is based on the "*backward*" variant of the dynamic programming principle (using the backward variant of the optimality principle).

As we have mentioned above it is also possible to calculate "*backward*" probabilities, $\beta_i(t)$, with the following definition: $\beta_i(t)$ represents the acceptance-probability of the input $a_{t,n}$, if the state at step t is s^i. In other words, the "*backward*" probability $\beta_i(t)$ computes the probability of seeing the observation from time t+1 to the end, given that we are in state i at time t (given the input sequence).

So [4]:

$$\beta_i(t) = P(a_{t,n} \mid s_t = s^i), t > 1.$$

The probability we are looking for will be

$$\beta_1(1) = P(a_{1,n} \mid s_1 = s^1) = P(a_{1,n})$$

Calculation of β function is made starting with values:

$$\beta_i(n+1) = P(\epsilon \mid s_{n+1} = s^i) = 1, i = 1, \cdots, \sigma.$$

For the recursive case, we have:

$$\beta_i(t-1) = P(a_{t-1,n} \mid s_{t-1} = s^i) = \sum_{j=1}^{\sigma} P(s^i \overset{a_{t-1}}{\to} s^j)\beta_j(t)$$

2.3. **Training Hidden Markov Models.** In the followings, we use the Baum-Welch algorithm [1]("*forward-backward*") for training a Hidden Markov Model. This algorithm, that has given a certain *training* input sequence (an observation sequence $a_{1,n}$), adjusts the probabilities of transitions in the HMM, in order to maximize the probability of the observation sequence. Having an HMM structure already defined, the algorithm will let us train the transition probabilities of the HMM. In fact, we can estimate the probabilities of transitions using a very simple algorithm: for each transition (arc) t which begins in a state s, we calculate how often this arc is used when the entry sequence is $a_{1,n}$. Thus, $P(t)$ is given by

$$P(t) = \frac{how\ often\ the\ arc\ t\ is\ used}{how\ often\ an\ arc\ beginning\ from\ s\ is\ used}$$

More exactly, the probabilities of transitions are calculated with the formula [2]

$$P(s^i \overset{a^k}{\to} s^j) = \frac{C(s^i \overset{a^k}{\to} s^j)}{\sum_{l=1,m=1}^{\sigma,\omega} C(s^i \overset{a^m}{\to} s^l)} \tag{2}$$

Let us notice that the formula (2) is used only if the sum $\sum_{l=1,m=1}^{\sigma,\omega} C(s^i \overset{a^m}{\to} s^l)$ is non-zero, otherwise the probability $P(s^i \overset{a^k}{\to} s^j)$ remains unchanged.

The C function (the "$numberingfunction$") in the above formula is calculated like this [2]:

$$C(s^i \overset{a^k}{\to} s^j) = \frac{1}{P(a_{1,n})} \sum_{t=1}^{n} \alpha_i(t) P(s^i \overset{a^k}{\to} s^j) \beta_j(t+1) \tag{3}$$

Let us notice that for the calculation of $C(s^i \overset{a^k}{\to} s^j)$ we have to know the probabilities of transitions for the HMM model. The main idea of the algorithm is the following: we will start with an estimate for the probabilities, and then use these estimated probabilities to derive better and better probabilities - we calculate the new values of function $C(s^i \overset{a^k}{\to} s^j)$ using the formula (3) and finally we adjust the probabilities of transitions using the formula (2). The measure of the improvement level of probabilities after a training sequence is given by the growth of the probability ($P(a_{1,n})$) of the input sequence compared to it's previous estimation. The process of recalculating the probabilities of transitions is finished when $P(a_{1,n})$ suffers no more significant modifications (in comparison with a given approximation error).

3. Experiment

In this section our aim is to test how a system represented as an HMM (in our example an agent for recognizing characters) works.

3.1. **An agent for recognizing characters.** Let us consider an agent for recognizing two characters "I" and "U". We assume that each character is represented by a binary matrix (for simplification, we consider that the matrix has 4 lines and 3 columns). So, the matrix corresponding to the character "I" is [[100][100][100][100]] and the matrix corresponding to character "U" is [[101][101][101][111]]. For this issue, we propose the model described in Figure 1.

Using the considerations made in subsection 2.1, the model is *hidden*, in other words is an HMM.

Of course, the structure of the Markov model chosen for the modeling of the problem, it is important.

Having as initial state the state "a", the above described HMM accepts the entries **100100100100** (the character "I") and **101101101111** (the character "U") (the entry for a character is obtained by juxtaposing the rows of the corresponding matrix in the following order: the first, the second, the third and the fourth line). The initial probabilities of transitions are calculated in comparison with the two entry sequences which are accepted by the HMM (the characters "I" and "U").

Let us notice that the dimension of the matrix (number of rows and columns)

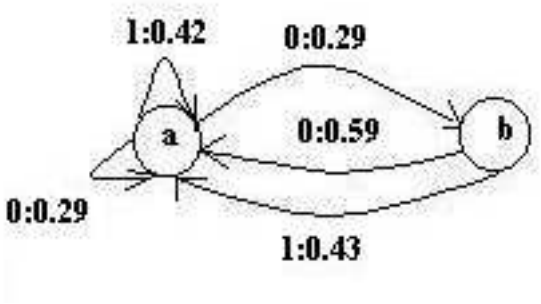


Figure 1. The Hidden Markov Model

used to represent the characters has no influence in the recognition process (only the probabilities of transitions after training the HMM change).

In this HMM, using the algorithm described in subsection 2.3, we observe that:

- the acceptance-probability for the entry **100100100100** (corresponding to the character "I") is $3.9190411 \cdot 10^{-4}$;
- the acceptance-probability for the entry **101101101111** (corresponding to the character "U") is $8.2214139 \cdot 10^{-5}$;
- the acceptance-probability for the entry **100100100111** (corresponding to the character "L") is $2.07292009 \cdot 10^{-4}$.

3.2. **First training.** First, we train the HMM to recognize the character "I" (we use the training algorithm described in subsection 2.4 for the entry sequence **100100100100**).

Considering the approximation error 10^{-7}, the HMM is trained in 13 steps. The probabilities of transitions during the training are described in Table 1 (the columns correspond to the probabilities of transitions).

In the HMM trained to recognize the character "I", we observe that:

- the acceptance-probability for the entry **100100100100** (corresponding to the character "I") is $3.906248 \cdot 10^{-3}$;
- the acceptance-probability for the entry **101101101111** (corresponding to the character "U") is $7.017882 \cdot 10^{-25}$;
- the acceptance-probability for the entry **100100100111** (corresponding to the character "L") is $1.953123 \cdot 10^{-3}$.

3.3. **Second training.** The second training of the HMM is for recognizing the character "U" (we use the training algorithm for entry sequence **101101101111**).

Considering the approximation error 10^{-7}, the HMM is trained in 26 steps, described in Table 2.

TABLE 1. The probabilities of transitions during the first training process

Step	P(a, 1, a)	P(a, 0, a)	P(a, 0, b)	P(b, 1, a)	P(b, 0, a)
1	0.34895903418	0.32739092305	0.32365004277	0.28061623839	0.71938376161
2	0.36807390057	0.30667030898	0.32525579045	0.21734816939	0.78265183061
3	0.39085481661	0.26963514927	0.33951003412	0.15079989110	0.84920010890
4	0.41877605321	0.21221865027	0.36900529651	0.08661076916	0.91338923084
5	0.45055613731	0.13549261529	0.41395124739	0.03595643299	0.96404356701
6	0.47879919631	0.05899019566	0.46221060803	0.00857366994	0.99142633006
7	0.49432319346	0.01463521470	0.49104159184	0.00080952252	0.99919047748
8	0.49898253274	0.00224291095	0.49877455631	0.00001805465	0.99998194535
9	0.49985762067	0.00028976704	0.49985261230	0.00000006090	0.99999993910
10	0.49998185346	0.00003637704	0.49998176950	0.00000000003	0.99999999997
11	0.49999772587	0.00000454959	0.49999772454	0.00000000000	1.00000000000
12	0.49999971564	0.00000056874	0.49999971562	0.00000000000	1.00000000000
13	0.49999996445	0.00000007109	0.49999996445	0.00000000000	1.00000000000

In the HMM trained to recognize the character "U", we observe that:

- the acceptance-probability for the entry **100100100100** (corresponding to the character "I") is $8.111088 \cdot 10^{-22}$;
- the acceptance-probability for the entry **101101101111** (corresponding to the character "U") is $3.251364 \cdot 10^{-3}$;
- the acceptance-probability for the entry **100100100111** (corresponding to the character "L") is $6.103893 \cdot 10^{-17}$.

After the agent was trained for recognizing the characters "I" and "U", the agent receives an entry, for example **100100100111** (the character "L"), which he tries to recognize. The recognition performs the following steps:

- first, the agent computes the probability $p1$ for the given entry in the first environment (trained for "I");
- second, the agent computes the probability $p2$ for the given entry in the second environment (trained for "U");
- third, the agent compares $p1$ and $p2$ and determines the maximum;
- fourth, because $p1$ is greater than $p2$ the agent recognize the character "I" as the most probable for the given entry.

This is a kind of supervised learning, the agent is trained for a few models, and after the training he tries to recognize a given entry. We chose this experiment with two characters because it is simple and illustrates very clearly the idea of training the agent using the training of the HMM.

TABLE 2. The probabilities of transitions during the second training process

Step	P(a, 1, a)	P(a, 0 , a)	P(a, 0, b)	P(b, 1, a)	P(b, 0, a)
1	0.71380471380	0.14141414141	0.14478114478	1.000	0.00
2	0.70681329384	0.12043988151	0.17274682466	1.000	0.00
3	0.69971006779	0.09913020337	0.20115972884	1.000	0.00
4	0.69291260238	0.07873780715	0.22834959047	1.000	0.00
5	0.68680005870	0.06040017610	0.25279976520	1.000	0.00
6	0.68162626416	0.04487879249	0.27349494335	1.000	0.00
7	0.67748162778	0.03244488333	0.29007348889	1.000	0.00
8	0.67431367967	0.02294103901	0.30274528131	1.000	0.00
9	0.67198200820	0.01594602459	0.31207196721	1.000	0.00
10	0.67031480817	0.01094442451	0.31874076732	1.000	0.00
11	0.66914788095	0.00744364286	0.32340847618	1.000	0.00
12	0.66834348425	0.00503045274	0.32662606302	1.000	0.00
13	0.66779488597	0.00338465792	0.32882045610	1.000	0.00
14	0.66742348964	0.00227046892	0.33030604144	1.000	0.00
15	0.66717331840	0.00151995519	0.33130672641	1.000	0.00
16	0.66700537657	0.00101612971	0.33197849371	1.000	0.00
17	0.66689289418	0.00067868255	0.33242842326	1.000	0.00
18	0.66681767274	0.00045301824	0.33272930902	1.000	0.00
19	0.66676742102	0.00030226306	0.33293031592	1.000	0.00
20	0.66673387346	0.00020162039	0.33306450615	1.000	0.00
21	0.66671148776	0.00013446328	0.33315404895	1.000	0.00
22	0.66669655476	0.00008966429	0.33321378095	1.000	0.00
23	0.66668659534	0.00005978602	0.33325361864	1.000	0.00
24	0.66667995391	0.00003986172	0.33328018438	1.000	0.00
25	0.66667552547	0.00002657642	0.33329789811	1.000	0.00
26	0.66667257283	0.00001771848	0.33330970870	1.000	0.00

Also, we considered an experiment with four characters: "*A*", "*L*", "*U*", "*I*". The associated HMM has four states and the recognition process works well.

On our opinion, it would be interesting the combination of the above described method of training with certain dynamic programming methods.

4. THE APPLICATION

The application is written in Microsoft Visual C++ 5.0 and implements the algorithms described in the previous sections.

Examples

1. For the sequence *100100100111* the application displays the following results:

   The entry is

   *
   *
   *
   * * *

   The maximum probability for the entry is 0.003906

   The character recognized for the given entry is:

   *
   *
   *
   *

2. For the sequence *000101111101* the application displays the following results:

   The entry is

   * *
   * * *
   * *

   The maximum probability for the entry *000101111101* **is** $2.2966411 \cdot 10^{-22}$

   The character recognized for the given entry is:

   * *
   * *
   * *
   * * *

5. Conclusions

In certain situations, the behavior of an intelligent agent can be modeled using an HMM. In such situations, it would be interesting to use mathematical methods for working on these models.

In the case of the proposed experiment, some future research would be:

- how could the proposed model be generalized for as many characters as possible;
- how could the structure of the HMM be generated dynamically;
- how would such probabilistic methods be more appropriate than others;
- how could the probabilistic methods be combined with others (that may also be heuristic) for obtaining a higher performance of the model;

- what would happen in certain "plateau" situations (where a given entry would have the same probability in more environments).

Anyway, which we wanted to emphasize in this paper is another way of working on problems of training intelligent agents.

References

[1] D.Jurafsky, James H. Martin : "Speech and language processing", Prentice Hall., 2000.
[2] E. Charniak: "Statistical language learning", MIT Press, 1996.
[3] S.J.Russell, P.Norvig: "Artificial intelligence. A modern approach", Prentice-Hall International,1995.
[4] D.Tatar, G.Serban: "Training probabilistic context-free grammars as hidden Markov models", Studia Universitatis "Babes-Bolyai", Series Informatica XLV (2), 2000, 69–78.

"Babeş-Bolyai" University, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

SOME PARALLEL NONDETERMINISTIC ALGORITHMS

VIRGINIA NICULESCU

Abstract. Nondeterminism is useful in two ways. First, it is employed to derive simple and general programs, where the simplicity is achieved by avoiding unnecessary determinism; such programs can be optimized by limiting the nondeterminism. Second, some systems are inherently nondeterministic; programs that represent such systems have to employ some nondeterministic construct. Nondeterministic programs can be mapped more easier on parallel machine, since parallelism brings some nondeterminism by itself.

In this article, there are constructed some nondeterministic programs, for some numerical methods, using the UNITY notation[3]. The correctness of the algorithms is proven, and some possible mappings are discussed.

1. Introduction

Nondeterminism is useful in two ways. First, it is employed to derive simple and general programs, where the simplicity is achieved by avoiding unnecessary determinism; such programs can be optimized by limiting the nondeterminism. Second, some systems are inherently nondeterministic; programs that represent such systems have to employ some nondeterministic construct.

There is a variety of parallel architectures, though parallel programs have to be developed such that they can be mapped in different ways, on different architectures. A solution is to specify little in the early stages of design, and specify enough in the final stages to ensure efficient execution on target architecture. Specifying little about program execution means that the programs may be nondeterministic.

To express the nondeterministic programs, the model used for the developing the programs is UNITY [3]: "Unbounded Nondeterministic Iterative Transformations", which is briefly described in the next section.

2. A Programming Notation

The UNITY program structure is

2000 *Mathematics Subject Classification.* 68N19.

1998 *CR Categories and Descriptors.* G.1.3. [**Mathematics of Computing**] : Numerical Analysis – *Numerical Linear Algebra*; G.4. [**Mathematics of Computing**] : Mathematical Software; D.1.3 [**Software**] : Programming Techniques – *Concurrent Programming.*

$$
\begin{array}{lcll}
program & \rightarrow & Program & program-name \\
 & & declare & declare-section \\
 & & always & always-section \\
 & & initially & initially-section \\
 & & assign & assign-section \\
end & & &
\end{array}
$$

The $declare-section$, names the variables used in the program and their types. The syntax is similar to that used in Pascal. The $always-section$ is used to define certain variables as function of others. This section is not necessary for writing UNITY programs, but it is convenient. The $initially-section$ is used to define initial values of some of the variables; uninitialized variables have arbitrary initial values. The $assign-section$ contains a set of assignment statements.

The program execution starts in a state where the values of variables are as specified in the initially-section. (A state is characterized by the values of all variables.) In each step, any one statement is executed. Statements are selected arbitrarily for execution, though in an infinite execution of the program each statement is executed infinitely often. A state of a program is called a *fixed point* if and only if execution of any statement of the program, in this state, leaves the state unchanged. A predicate, called FP, characterize the fixed points of the program. Once FP holds, continued execution leaves values of all variables unchanged, and therefore it makes no difference whether the execution continues or terminates.

The termination of a program is regarded as a feature of an implementation. A program execution is an infinite sequence of statement executions and an implementation is a finite prefix of the sequence.

2.1. **Mapping Programs to Architectures.** One way to implement a program is to halt it after it reaches a fixed point.

A mapping to a von Neumann machine specifies the schedule for executing assignments and the manner in which a program execution terminates.

In a synchronous shared-memory system, a fixed number of identical processors share a common memory that can be read and written by any processors. The synchronism inherent in a multiple-assignment makes it convenient to map such a statement to this architecture.

A UNITY program can be mapped to asynchronous shared-memory system, by partitioning the statements of the program among the processors. In addition, a schedule of execution for each processor should be specified that guarantees a fair execution for each partition. If the execution for every partition is fair, then any fair interleaving of these executions determines a fair execution of the entire program. Two statements are not executed concurrently if one modifies a variable that the other uses.

Other architectures can be considered for mappings.

2.2. **Assignment Statement.** It is allowed that a number of variables to be assigned simultaneously in a multiple assignment, as in

$$x, y, z := 0, 1, 2.$$

Such an assignment can also be written as a set of assignment-components separated by $\|$, as in

$$x, y := 0, 1 \| z := 2$$

or

$$x := 0 \| y := 1 \| z := 2.$$

The variables to be assigned and the values to be assigned to them may be described using quantification, rather than enumeration:

$$< \| i : 0 \leq i < N :: A[i] := B[i] > .$$

A notation like the following is used for a conditional assignment:

$$\begin{array}{rl} x := -1 & if\ y < 0\ \sim \\ 0 & if\ y = 1\ \sim \\ 1 & if\ y > 0\ . \end{array}$$

2.3. **Assign-section.** The symbol ‡ acts as a separator between the statements. A quantified-statement-list denotes a set of statements obtained by instantiating the statement-list with the appropriate instances of bounded variables; if there is no instance, quantified-statement-list denotes an empty set of statements. The number of the instances must be finite. The boolean expression in the quantification should no name program variables whose values may change during program execution.

2.4. **Initially-section.** The syntax of this section is the same as that of the assign-section except that symbol $:=$ is replaced with $=$. The equations defining the initial values should not be circular.

2.5. **Always-section.** An always-section is used to define certain program variables as function of other variables. The syntax used in the always-section is the same as in the initially-section.

3. Nondeterministic Gauss Elimination

We consider the Gaussian elimination scheme for solving a set of linear equations,

$$A \cdot X = B,$$

where $A[0..n-1, 0..n-1]$ and $B[0..n-1]$ are given and the solution is to be stored in $X[0..n-1]$. Gaussian elimination is presented typically as a sequence of n pivot steps. The following UNITY program allows nondeterministic choices in the selections of the pivot rows.

3.1. **A Solution.** Let $M(A;B)$ (or M for short) the matrix with n rows and $n+1$ columns, where the first n columns are from A and the last column is from B. In the Gaussian elimination $M(A;B)$ is modified to $M(A^{'};B^{'})$ by certain operations such that

$$A \cdot X = B$$

and

$$A^{'} \cdot X = B^{'}$$

have the same solutions for X. The goal of the algorithm is to apply a sequence of these operations to convert $M(A;B)$ to $M(I_n;X_F)$, where I_n is the identity matrix; then X_F is the desired solution vector. This goal can be realized if the rank of A is n, which we assume to be the case.

The program consists of two kinds of statements:

(1) Pivot with row u, provided that $M[u,u] \neq 0$; this has the effect of setting $M[u,u]$ to 1 and $M[v,u]$ to 0, for all $v, v \neq u$
(2) Exchange two rows u and v, provided that both $M[u,u]$ and $M[v,v]$ are zero and at least one of $M[u,v], M[v,u]$ is nonzero; this has the effect of replacing a zero diagonal with a nonzero element.

Due to the fact that there are some possible exchanges between the rows, the elements of the solution vector will be exchanged also. The permutation of the elements is stored in an array p.

Program Gauss
  declare
    $M : array[0..n-1, 0..n]\ of\ real$
    $p : array[0..n-1]\ of\ integer$
  initially
    $< i : 0 \leq i < n : p[i] = i >$
  assign
    {pivot with row u if $M[u,u] \neq 0$}
    $< \ddagger u : 0 \leq u < n ::$
      $< \| v,j : 0 \leq j < n \wedge 0 \leq v < n \wedge v \neq u ::$
        $M[v,j] := M[v,j] - M[v,u] \cdot M[u,j]/M[u,u]\ if\ M[u,u] \neq 0$
      $>$
      $\| < \| j : 0 \leq j < n ::$
        $M[u,j] := M[u,j]/M[u,u]\ if\ M[u,u] \neq 0$
      $>$
    $>$

```
‡{exchange two rows if both have zero diagonal elements and the
   exchange results in at least one of these elements being set to nonzero}
  < ‡u, v : 0 ≤ u < n ∧ 0 ≤ v < n ∧ u ≠ v ::
    < ||j : 0 ≤ j < n :: M[u, j], M[v, j], p[u], p[v] := M[v, j], M[u, j], p[v], p[u]
      if M[u, u] = 0 ∧ M[v, v] = 0 ∧ (M[u, v] ≠ 0 ∨ M[v, u] ≠ 0)
    >
  >
end{Gauss}
```

3.2. **Correctness.** Let M^0 denote the initial Z matrix. Since each statement in the program modifies M such that the solutions to the given linear equations are preserved, we have the following invariant:

$$invariant \quad M^0, M \text{ have the same solution.}$$

In the following, A refers to the $n \times n$ matrix in the left part of M, and B, to the last column of M. First, it is proven that the program *Gauss* reaches a fixed point and that at any fixed point, A is an identity matrix. Then, from the invariant, B is the desired solution vector. In the following, a *unit column* is a column in which the diagonal element is 1 and all other elements are 0. That is, column u is a unit column means that

$$M[v, u] = 0 \ if \ u \neq v \ \sim 1 \ if \ u = v.$$

To show that a fixed point is reached, it is proven that the pair (p, q), where

$$p = \text{number of unit columns in } A$$
$$q = \text{number of nonzero diagonal elements in } A,$$

increases lexicographically with every state change.

We consider each statement in turn. Pivoting with row u, where column u is a unit column, cause no state change. A state change results from a pivot operation with row u only if column u is not a unit column; the effect of the pivot operation is to set u to a unit column, thus increasing p.

Two rows u and v are exchanged only when $M[u, u] = 0 \wedge M[v, v] = 0 \wedge (M[u, v] \neq 0 \vee M[v, u] \neq 0)$. Hence neither of the columns u or v is a unit column. The exchange preserves all the unit columns, also preserving p. In addition, at least one diagonal element, $M[u, u]$ or $M[v, v]$ is set to nonzero. Since both of these elements were previously zero, q increases. Therefore, every state change in program *Gauss* increases (p, q) lexicographically. Since each of p, q is bounded from above by n, *Gauss* reaches a fix point.

Now, it must be proved that A is an identity matrix at any fix point. The proof is as follows. Lemma 1 proves that if any diagonal element $M[u, u]$ is nonzero at a fixed point, u is a unit column. Lemma 2 proves that if some diagonal element is zero at a fix point, all elements in the row are zero. This contradicts the assumption

that the determinant of A is nonzero. (Note that execution of any statement in *Gauss* preserves the determinant.) Therefore every diagonal element is nonzero and, using Lemma 1, A is an identity matrix.

Lemma 1. *At any fixed point of program Gauss,*

$$M[u,u] \neq 0 \Rightarrow u \text{ is a unit column.}$$

Proof: Consider the statement for a pivot corresponding to row u. At any fix point, given that $M[u,u] \neq 0$, for any j and $v, u \neq v$,

$$M[v,j] = M[v,j] - M[v,u] \cdot M[u,j]/M[u,u]$$

and

$$M[u,j] = M[u,j]/M[u,u].$$

In particular, with $j = u$,

$$M[v,u] = M[v,u] - M[v,u] \cdot M[u,u]/M[u,u] = 0$$

and

$$M[u,u] = M[u,u]/M[u,u] = 1.$$

Therefore u is a unit column.

Lemma 2. *At any fixed point of program Gauss,*

$$M[u,u] = 0 \Rightarrow M[u,v] = 0, \forall v \neq u.$$

Proof: Consider two cases: $M[v,v] = 0$ and $M[v,v] \neq 0$.

In the first case, consider the exchange statement for rows u, v. At any fix point, given that $M[u,u] = 0 \wedge M[v,v] = 0$:

$$(M[u,u] = 0 \wedge M[v,v] = 0) \vee (\wedge j :: M[u,j] = M[v,j]).$$

Consider the particular case, $j = v$. Then,

$$(M[u,u] = 0 \wedge M[v,v] = 0) \vee (M[u,v] = M[v,v]).$$

Using the fact that $M[v,v] = 0$ we conclude that $M[u,v] = 0$.

In the second case, if $M[v,v] \neq 0$ from Lemma 1, $M[u,v] = 0$.

3.3. **Mappings.** Program *Gauss* can be implemented in a variety of ways on different architectures. For a sequential machine, it may be more efficient to choose the pivot rows in a particular order. The correctness of this scheme is obvious from the proof because it is obtained from the given program by restricting the nondeterministic choices in statement executions. For an asynchronous shared-memory or distributed architecture, the given program admits several possible implementations; the simplest one is to assign a process to a row. To facilitate the exchange operation, it is possible to allow the row number at a process to be changed. Two rows can be then exchanged simply by exchanging their row numbers. A parallel synchronous architecture with $O(n)$ processors can complete

each exchange operation in a constant time and each pivot operation in $O(n)$ steps; with $O(n^2)$ processors, a pivot operation takes constant time.

4. The Inverse of a Matrix

The method we use for the computation of the inverse matrix use Gauss-Jordan steps. A Gauss-Jordan step with the pivot element $a[u,v] \neq 0$ transforms the matrix A elements, in the following way:

$$a[i,j] = \begin{cases} \frac{1}{a[u,v]} & , i = u \wedge j = v \\ \frac{-a[i,j]}{a[u,v]} & , i = u \wedge j \neq v \\ \frac{a[i,j]}{a[u,v]} & , i \neq u \wedge j = v \\ \frac{a[i,j] \cdot a[u,v] - a[i,v] \cdot a[u,j]}{a[u,v]} & , i \neq u \wedge j \neq v \end{cases} .$$

If we apply a Gauss-Jordan step n times on matrix $A[0..n-1, 0..n-1]$ we obtain the inverse matrix A^{-1} [2]. We assume that the rank of matrix A is n.

4.1. **A Solution.** The choice of the pivot element it is done in nondeterministic way, provided that it is nonzero. Since, a pivot operation have to be done only one time for a particular row u and a particular column v, after the execution of a pivot operation with the pivot element $a[u,v]$ we set $ind1[u] = 1$ and $ind2[v] = 1$. The $ind1$ and $ind2$ are two arrays which indicate the possible pivot steps. An elimination step with the pivot $a[u,v]$ can be executed only if $ind1[u] = 0 \wedge ind2[v] = 0$.

Because we not choose every time pivot elements from the diagonal, a permutations of the rows of the inverse matrix results. The permutation p depends of the choices of the pivot elements.

Program inverse
declare
  $a : array[0..n-1, 0..n-1]$ *of real*
  $ind1, ind2 : array[0..n-1]$ *of integer*
  $p : array[0..n-1]$ *of integer*
initially
  $< u : 0 \leq u < n :: ind1[u], ind2[u] = 0, 0 >$
  assign
    {pivot operation with the element u, v if $a[u,v] \neq 0$}
    $< \ddagger u, v : 0 \leq u < n \wedge 0 \leq v < n ::$
      $< \| i, j : 0 \leq i < n \wedge 0 \leq j < n ::$

$$
\begin{array}{lll}
a[i,j] & := 1/a[u,v] & if\ i=u \wedge j=v \ \sim \\
 & := -a[u,j]/a[u,v] & if\ i=u \wedge j\neq v \ \sim \\
 & := a[i,v]/a[u,v] & if\ i\neq u \wedge j=v \ \sim \\
 & := (a[i,j]\cdot a[u,v]-a[u,j]\cdot a[i,v])/a[u,v] & if\ i\neq u \wedge j\neq v
\end{array}
$$

$>$

$\|ind1[u], ind2[v], p[u] := 1, 1, v$

$if \quad a[u,v]\neq 0 \wedge ind1[u]=0 \wedge ind2[v]=0$

$>$

$end\{inverse\}$

4.2. **Correctness.** If we denote by p the following sum $p=(\sum u : 0\leq u<n : ind1[u])$, and by q the sum $q=(\sum u : 0\leq u<n : ind2[u])$, it can be easy proved that the for the pair (p,q) the equality $p=q$ holds at any moment of the execution. So, we can write:

$$invariant \quad p=q.$$

The number $p(p=q)$ increase after the execution of any statement. The values for p and q are bounded from above by n, hence the program $inverse$ reaches at a fix point, where $p=q=n$.

The equality $p=q=n$ which holds at any fix point shows that there are executed exact n Gauss-Jordan steps with pivot elements from different rows and columns. Therefore the matrix A at any fix point is the inverse matrix of the initial matrix, possible with the rows permuted.

To transform the result to the true inverse matrix the following program can be used.

$Program\ transform$
$declare$
$a : array[0..n-1, 0..n-1]\ of\ real$
$p : array[0..n-1]\ of\ iuteger$
$assign$
$< \ddagger u, v : 0\leq u<n \wedge 0\leq v<n ::$
{rows exchange }
$< \|j : 0\leq j<n :: a[u,j], a[v,j] := a[v,j], a[u,j] >$
$\| \ p[u], p[v] := p[v], p[u]$
$if\ p[u]=v \vee p[v]=u$
$>$
$end\{transform\}$

4.3. **Mappings.** On a sequential architecture the program $inverse$ can be mapped by choosing the first pivot element founded; the search of the element is made depending in $ind1$ and $ind2$.

The program can be implemented on an asynchronous shared-memory system, by assigning a processor to a row, or by assigning a processor to each matrix element (and so the operations associated with it), provided that there are enough processors.

On a parallel synchronous architecture with n^2 processors the execution of the program takes $O(n)$ time.

4.4. **Other Applications.** The program *inverse* can be used to find the rank of a matrix. The rank it will be equal to $p = q$, which represents the number of the Gauss-Jordan steps, which were executed.

With slight modifications, this program can be used to resolve a system of linear equations. The matrix A is replaced with the matrix M defined for the Gauss program $M = [A|B]$ and finally the result (the solution vector)is the last column of the matrix at the fix point. A permutation of the elements it is done in this case also.

The application of n Gauss-Jordan steps represents also the second stage of the algorithm SIMPLEX.

5. Conclusions

There are presented some nondeterministic algorithms from numerical analysis. Their correctness was proved, and different mappings are discussed.

Nondeterministic programs can be mapped more easier on parallel machine, because the parallelism brings some nondeterminism by itself.

Interesting algorithms can be developed using the concept of nondeterminism. Nondeterministic programs can be implemented on different architectures, in efficient ways.

References

[1] G. E. Blelloch, B. M. Maggs , *Parallel Algorithms*, ACM Computing Surveys, Vol. 28, No. 1, March 1996, pg. 51-54.
[2] W.W. Breckner, Operational Research, "Babeş-Bolyai" University, Cluj-Napoca, 1981 (in Romanian).
[3] K.M. Chandy, J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.
[4] Gh. Coman, *Numerical Analysis*, Libris, Cluj-Napoca, 1995 (in Romanian).
[5] I. Foster, Designing and Building Parallel Programs, 1995.
[6] Carrol Morgan, *Programming from Specifications*, Prentice Hall, 1990.

Department of Computer Science, "Babeş-Bolyai" University, RO-3400 Cluj-Napoca, 1 Kogălniceanu St., RO-3400 Cluj-Napoca, Romania
E-mail address: gina@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

ON PROGRAMMING STYLE – PROGRAM CORRECTNESS RELATION

M. FRENŢIU

Abstract. There is little empirical information about the relation between the quality of the programs and the style of the programmer. One experiment in this direction is presented in this paper.

It is considered that the style of the programmer affects his efficiency, and the correctness of his programs. To sustain this hyphotesis, the papers at a written examination were analysed and the conclusions are presented.

Key words: programming methodology, style, quality, software metrics, education

1. Introduction

The need to measure various attributes met in software engineering is underlined in [4]. Certainly, it is very important to asess the time needed to realize a software project, or to evaluate the quality, maintainability, reliability, or usability of a program, or the productivity of a programmer. Also, we think it is very useful to assess the effect of programming style on the above mentioned attributes.

Is there a relation between the style of a programmer and the quality of his work? We need some definitions of the concepts we use. What is a style? In [2] the word style is considered to be the general way in which something is done, "the general attitudes and usual ways of behaving", "the style of a product is its design", and the style of writing is "the choice of words and the way in which sentences and paragraphs are structured". It is somebody's manner of speaking, acting, writing, for expressing his thought.

In Software Engineering when we define the style we can think only to how the programs look [15, 17]. Therefore, in a narrow sens we have:

Definition 1. Programming Style consists of the ways in which the programer writes programs easy to read, and easy to understand, the ways in which these qualities are achieved.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques* D.2.7 [**Software**] : Software Engineering – *Distirbution, Maintenance and Enhancements* .

Readability is considered to be the main attribute of style [11]. And readability depends on indentation, good names, and on the comments present in the texts of the programs.

We will briefly describe the elements of style. But we must say that they may differ from person to person, although each programmer must have and think to his own style. Citing Gries, "Whatever conventions you use, use them consistently" [10].

Therefore, the elements of style (in a narrow sens) are:

- Comments;
- Text Formatting (Indentation, White Spaces);
- Good Names for Entities of the Program.

Comments are very important for internal documentation. Every program ought to have documentation in it. Comments must be used:

- to state the specification of the problem solved by the program, to precise the author, date, and other useful information for the reader;
- to show the purpose of each variable;
- to explain what a procedure does: to show the specification of the procedure, and the meaning of the parameters;
- to write the loop invariants in those places of the program where they hold;
- to explain the conditions in which some parts of the programs are reached, and the role of these parts;
- and to transmit other useful information to the reader.

Indentation rules are used to enrich the clarity of the program. There are a variety of suggestions for such rules. Gries suggests the following indentation rules [10]:

- succesive short commands can be written on the same line provided that, logically, they belong together;
- commands of a sequence that appear on succesive lines should begin in the same column;
- subcommands of a comand must be indented 3 or 4 spaces from the column where the command begins;
- the pre- and postcondition of a command should begin in the same column as the command;
- a loop should be preceded by an invariant and a bound function; these should begin in the same column as the beginning of the loop.

Then, the names of variables, functions, types, contribute to the clarity of programs [13, 14]. Here are some rules in this direction:

- choose meaningful names for all entities;

- do not use a single name for two variables (i.e. the same name has one meaning in a part of the program, and a second meaning in another part);
- define your variables before you use them, and then be sure to adhere to the definitions;
- when a name is composed of two words, start the second one with a capital letter.

But everybody admits that a badly conceived program remains a bad program. It may be well documented through comments, it may be nice indented, and it may use meningful names, if it is not correct it is useless. And, also, if it cannot be maintained, it is not of a good quality.

Therefore, we consider a larger aspect of programming style.

Definition 2. Programming Style consists of all means taken by the programmer in his activity for producing reliabile products easy to read, and easy to understand, the ways in which these qualities are achieved [13, 10], [8, page 137].

This definition sets in the main frame the way in which the programmer respects the general accepted rules for good programming. It starts with the specification of the program, with the way in which the design is done, with the clarity of documentation for all stages of work. As Floyd [5] said, we must permanently fight to acquire better programming methods for producing correct and easily maintainable programs. The style depends on how the general accepted rules for good programming are respected. And there are many books and papers that contain such rules [13, 14, 10, 6, 7, 8, 9].

2. The experiment

The opportunity to do this study was offered by the Graduate Licence Examination held in June 2001. 94 students took part in this exam (for B.Sc. in Computer Science). The subject consisted of two parts: theoretics (the first two subjects), and programming (next two subjects). Here are the subjects:

(1) Sorting. Quicksort;
(2) Merging;
(3) Specify, design, and implement an Abstract Data Type ***SET***;
(4) Write a program which prints the longest sequence of consecutive primes from a given sequence of natural numbers.

The results are given in Table 2. Since the results for the theoretical subjects are not directly used in the analysis, only the total number of points (for all four subjects) are given in the fourth column (denoted by T). The P3, and P4 columns contain the points given for the subject (3), and (4), respectively. Then, the elements of style are measured by grades from 0 to 10. The grade 0 is given when the corresponding attribute is not present at all, and 10 if it is considered perfectly.

No	P3	P4	T	P3	P4	No	P3	P4	T	P3	P4
				I C N G	I C N G					I C N G	I C N G
1	23	22	73	8 0 6 7	8 0 8 5	48	24	21	80	6 5 5 6	5 4 4 5
2	19	18	66	7 2 7 5	6 3 4 3	49	18	23	75	4 0 4 4	5 0 3 5
3	25	8	73	8 1 3 6	7 0 8 2	50	25	25	90	9 6 6 9	8 5 5 8
4	15	18	57	7 3 5 7	8 2 6 2	51	24	19	79	7 0 4 6	8 8 5 7
5	22	15	70	7 0 7 4	9 0 5 4	52	20	18	72	7 2 6 7	6 2 4 6
6	15	19	69	8 1 6 5	6 6 8 7	53	22	23	85	6 1 5 6	7 2 3 7
7	8	19	57	7 0 5 2	7 1 8 4	54	11	18	64	4 1 4 4	5 3 2 4
8	15	20	71	5 1 5 2	8 3 7 6	55	23	20	78	5 4 5 6	5 1 4 5
9	22	25	81	7 5 5 5	8 5 8 6	56	20	21	75	3 1 2 3	4 3 4 4
10	25	22	79	7 6 5 6	7 8 6 8	57	20	0	52	4 0 3 4	0 0 0 0
11	25	20	84	8 4 6 7	7 0 4 6	58	10	20	52	5 4 4 5	6 1 3 6
12	22	19	78	8 0 4 8	8 2 5 5	59	25	22	87	6 7 6 8	6 0 4 6
13	20	20	77	6 6 5 8	4 8 7 7	60	20	5	65	7 5 5 7	6 4 3 5
14	9	5	42	0 0 0 0	2 0 2 0	61	6	10	42	2 2 1 2	5 0 5 5
15	18	17	75	6 0 3 5	5 4 3 4	62	18	15	64	5 0 3 4	4 1 3 3
16	22	20	82	5 0 4 5	5 1 4 5	63	23	20	82	4 2 5 5	5 3 4 5
17	0	2	28	0 0 0 0	2 0 0 0	64	20	20	80	6 0 5 6	6 7 5 7
18	23	22	80	6 0 4 6	5 0 4 5	65	24	21	82	7 8 7 8	7 7 6 7
19	24	20	82	9 7 7 8	8 6 6 8	66	20	18	74	6 0 4 5	6 9 6 8
20	13	17	63	6 1 3 3	7 0 4 7	67	21	20	73	6 0 4 5	7 3 4 6
21	15	15	57	6 2 3 5	6 3 5 5	68	9	21	65	2 0 3 2	3 4 5 4
22	17	18	71	7 7 6 7	7 6 5 6	69	14	20	67	3 0 2 2	5 0 2 4
23	5	13	42	3 0 1 1	2 4 2 2	70	25	15	78	6 3 5 6	5 2 3 4
24	12	20	66	5 4 5 4	5 6 6 5	71	12	10	55	2 0 3 3	4 0 2 2
25	25	23	88	6 8 6 9	7 7 7 8	72	20	19	76	4 1 3 3	4 2 3 3
26	10	19	49	2 0 1 3	6 1 3 6	73	23	15	76	7 3 5 6	4 3 4 4
27	0	0	15	0 0 0 0	0 0 0 0	74	9	15	57	4 3 3 4	5 3 4 4
28	13	15	61	2 7 4 4	7 0 4 4	75	20	20	72	5 3 4 5	5 2 4 5
29	18	21	78	6 4 5 5	7 5 5 7	76	22	13	75	6 7 5 6	5 5 3 4
30	24	18	81	8 6 6 7	5 5 3 5	77	22	25	76	7 3 6 7	8 9 5 6
31	24	21	85	7 1 6 6	5 4 4 5	78	17	22	79	4 0 2 3	6 6 4 6
32	25	23	85	7 6 6 8	7 6 5 8	79	13	18	64	4 0 3 3	7 5 5 7
33	24	15	74	6 3 6 7	5 3 6 5	80	23	19	82	6 0 4 5	5 4 5 5
34	23	22	65	5 7 7 7	6 0 5 6	81	25	23	85	5 0 4 5	6 1 4 5
35	18	20	77	6 2 6 6	7 5 4 5	82	22	21	82	2 4 5 5	5 3 5 5
36	24	20	79	9 9 8 9	5 6 6 6	83	18	22	73	6 4 6 6	6 4 5 6
37	19	18	76	6 5 4 5	3 4 3 5	84	11	18	68	3 2 3 3	4 0 2 3
38	20	4	53	5 1 5 5	2 0 1 1	85	8	18	46	4 2 3 3	5 2 3 4
39	20	18	66	8 5 4 7	5 7 5 5	86	12	19	59	5 0 4 4	5 0 3 5
40	22	19	79	7 0 5 6	6 0 4 4	87	25	20	83	6 0 5 6	6 4 3 5
41	20	22	65	6 0 4 6	7 1 4 7	88	9	18	54	4 0 3 3	5 1 4 4
42	11	3	46	2 1 1 1	1 0 1 1	89	25	20	82	7 0 6 6	6 6 5 6
43	9	11	30	2 0 1 2	2 2 1 2	90	23	22	82	8 7 6 8	8 7 6 8
44	21	20	61	3 0 4 4	3 5 3 3	91	24	20	79	7 5 6 7	7 6 6 7
45	22	13	72	5 2 3 5	4 1 3 4	92	17	21	72	6 5 5 6	6 6 6 6
46	24	21	80	6 5 5 6	6 4 4 5	93	25	23	88	7 4 5 7	6 7 5 7
47	21	20	76	5 5 4 5	4 5 3 4	94	24	15	77	5 3 4 4	5 4 3 4

TABLE 1. Primary data for attributes of style

Line no.	X1	X2	Y	C(X1,Y)	C(X2,Y)
1	P3	T	P3/I	0.70	0.67
2	P3	T	P3/C	0.37	0.37
3	P3	T	P3/N	0.70	0.70
4	P3	T	P3/G	0.79	0.73
5	P4	T	P4/I	0.64	0.57
6	P4	T	P4/C	0.42	0.45
7	P4	T	P4/N	0.56	0.50
8	P4	T	P4/G	0.76	0.66
9	P3	P4	P3/I + P4/I	0.63	0.61
10	P3	P4	P3/C + P4/C	0.43	0.40
11	P3	P4	P3/N + P4/N	0.60	0.63
12	P3	P4	P3/G + P4/G	0.72	0.69
13	P3/I + P4/I	P3/N + P4/N	T	0.68	0.67
14	P3/C + P4/C	P3/G + P4/G	T	0.48	0.77

TABLE 2. The correlation coefficients for various attributes

For example, in column P4/C the grades for comments in the program corresponding to the problem 4 are given. The minimum amount of comments required to obtain 10 is formed from the statement of the problem, the precondition and the postcondition for each procedure, the meaning of each variable, and, in some important places, the situation in which that part of the procedure is reached.

The papers were independently analysed by two teachers, and the points were given for the global correctness of programs. It was similar to an inspection of the program, such that the given number of points reflects the measure of program correctness (columns P3, and P4, respectively), and acquired knowledges (column T). Although everybody knew that the correctness of programs is important, and this was watched carefully, the students also knew that the teachers look at their style of programming.

The columns marked by (C), (I), and (N) contain the points for the measures in which the rules connected to comments, indentation, and good names are respected, as explained above. The column (G) contains the points (from 0 to 10) for the way in which all the general accepted programming rules are respected, starting with the specifications of the problem and of all used modules, analysing the design and the documentation of all activities. The points contained in the columns (C), (I), (N), and (G) were given by the author of this paper.

3. CONCLUSIONS

It is known that the measure of linear dependence between two characteristics is given by the correlation coefficient of these characteristics. Therefore, the correlation coefficients for various attributes were computed. They are given in Table 3, where C(X,Y) denotes the correlation coeficient of the attributes X and Y.

As we expected, these coefficients are positive, and show that there is a strong dependence between the corresponding attributes. This confirms the idea that programming style has an important impact on program correctness. Also, it was expected that the largest coefficients are between the correctness and the way in which the general rules are satisfied (column G).

Moreover, we must observe that these correlation coeficients are stable for both problems, i.e. $C(P3, A)$ is closed to $C(P4, A)$ for all attributes $A \in \{I, C, N, G\}$. This confirms that the students have been convinced of the necessity to respect the above mentioned rules, and have acquired an acceptable programming style.

Nevertheless, we must observe some anomalies, and, for educational purposes, take some measure to eliminate them. First, we can observe that the smallest coefficients correspond to the column C: $C(P3, P3/C) = 0.37$ is the smallest of all. Therefore, students do not like writing comments. In this direction we must observe that there are 56 programs (from $188 = 2 \times 94$) that have no comments at all!

This is in contrast with the case of the other attributes, where the presence of zeros is an exception, only the lines with $P3 = 0$, or $P4 = 0$, having the grades for these attributes equal to zero.

The indentation rules are much better respected. There is one more reason for this. At all lectures, when the teachers write algorithms or code, they respect these rules in all lines. But only sometimes they write comments.

We may conclude that a good programming style and a correct programming habit must be taught in parrallel. As can be seen [6, 7, 8, 9] there were many important programming style rules in my lectures, but they were not compulsory, as is the case of many universities [1, 3, 11, 12, 15, 17]. As a consequence of this analysis, I think such rules must become compulsory.

References

[1] Adams, David, and Dan Beckett, Programming Style, http://www.island-data.com/downloads/ papers/programmingstyle.html, 2001.

[2] BBC English Dictionary, HarperCollins Publishers, 1993.

[3] Craig E.Wills, Programming Assignments, http://www.cs.wpi.edu/∼cew/courses/2005/style/style.html

[4] Fenton, N.E., Software Metrics. A Rigorous Approach, Int. Thompson Computer Press, London, 1995.

[5] Floyd, R.W., The Paradigms of Programming, Comm.ACM, 22(1979),8, 455-460.

[6] Frenţiu M., B.Prv, Programming Proverbs Revisited, Studia Univ. Babeş- Bolyai, Mathematica, XXXVIII (1993), 3, 49-58.

[7] Frenţiu M., On Program Correctness and Teaching Programming, Computer Science Journal of Moldova, vol.5 (1997), no.3, pp.250-260.

[8] Frenţiu M., Lazar I.(Romanian), Programming Fundamentals. Algorithms Design, Ed.Univ."Petru-Maior", Târgu-Mureş, 2000.

[9] Frentiu M., Verifying Program Correctness (Romanian), Ed.Univ."Petru-Maior", Târgu-Mureş, 2001.

[10] Gries, D., The Science of Programming, Springer Verlag, Berlin, 1981.
[11] Haahr, P., A Programming Style for Java, http://www.webcom.com/~haahr/essays/java-style
[12] Keith Gabryelski, Wildfire C++ Programming Style, http://www.cs.umd.edu/users/cml/cstyle
[13] Kernigham, Brian W., and P.J.Plauger, The Elements of Programming Style, McGraw-Hill Book Company, New York, 1974.
[14] Ledgard H.F., Programming Proverbs for Fortran Programers, Hayden Book Company, Inc., New Jersey, 1975.
[15] McCann, Toward Developing Good Programming Style, http://www.comsc.ucok.edu/~mccann/style_p.html
[16] Meyer, B., Object Oriented Software Construction, Prentice Hall, Englewood Cliffs, 1988.
[17] David R.Tribble, Notes About Programming Style, http://www.flash.net/~dtribble/src/sys/style.htm, 1998-04-16

Department of Computer Science, "Babeş-Bolyai" University, 1, M. Kogălniceanu, RO-3400 Cluj-Napoca, Romania
E-mail address: mfrentiu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

USING SCALABLE STATECHARTS FOR ACTIVE OBJECTS INTERNAL CONCURRENCY MODELING

DAN MIRCEA SUCIU

Abstract. In the last two decades, the design of object models having concurrent features has represented a constant concern for many researchers. The fundamental abstractions used in this methodology are concurrent (or active) objects and protocols for passing messages between them. Statecharts seem to be one of the most appropriate ways of modeling the behavior of concurrent objects. Based on statecharts we will define an executable formalism, called *level 2 scalable statechart* (SS^2), for modeling of intra-concurrency in object-oriented concurrent applications.

Key words: object-oriented concurrent programming, reactive systems, statecharts.

1. Introduction

In the last two decades, the design of object models having concurrent features has represented a constant concern for many researchers. This was happening for mainly two reasons. On the one hand, as an effect of the obtained technological progress, many object-oriented programming languages having concurrent features have been designed during this time (over 100 such languages have been discussed and systemized in [10]).

On the other hand, the fact is known that object-oriented programming has been developed having as a model our environment (seen as a set of objects among which several relationships exist and which communicate between them by message transmission). However, in the real world these objects are naturally concurrent, which leads to the normal trend of transposing this thing into programming.

It is interesting how two distinct criteria, the first one objective (determined by the rise of performances and complexities of the calculus systems), and the second one subjective (actually determined by "decency", which urges us to solve different abstract problems looking for similitude with the real world), have finally led to

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques* D.2.7 [**Software**] : Software Engineering – *Distirbution, Maintenance and Enhancements* .

the development of some concepts, some programming techniques and implicitly of some efficient analysis and design methods for developing applications.

The concurrent programming has occurred before the object-oriented programming. It has been applied for the first time within the framework of procedural languages. Here the main problems studied have been concerned to the synchronization of the parallel execution of some instruction sequences and to the information transmission among many other concurrent activities.

Once with the appearance of object-oriented programming software development has met a qualitative and meaningful leap. In this way, the development of these programs (or applications) does not involve the decomposition of problems into algorithmic procedures, but independent objects that interacts among them. An evaluation of the coordinating primitives of these interactions will be achieved in a concurrent system.

In the same time, a great interest was accorded to object oriented technology, especially to the analysis and design methods. The analysis and design methods may be defined as coherent approaches used to describe a system. Due to the complexity of the systems, different models are built, each of them containing another view of the system. Any model emphasize an aspect and neglect all the others. For instance, the entity- relation model describes the dates involved in the system and indicates nothing about their processing. In order to cover all the aspects connected with the design, every method uses more than one model.

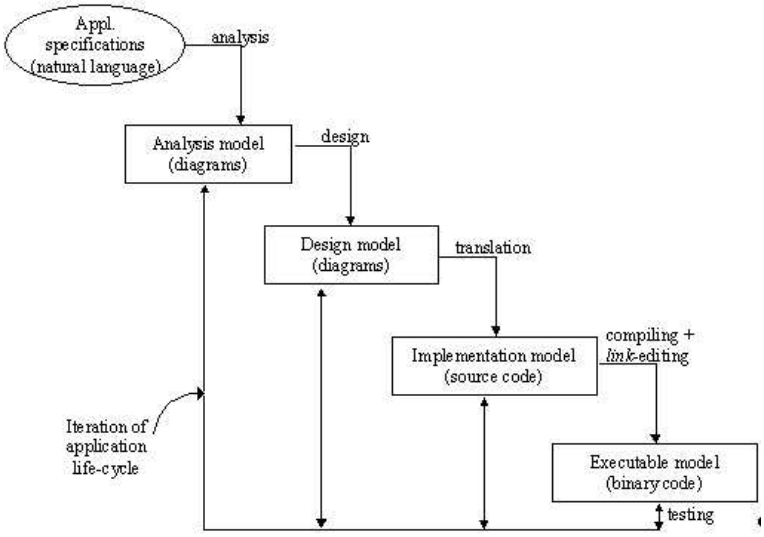


FIGURE 1. Iterative model of applications development using an object-oriented analysis/design method

The life cycle of an application, represents the stages that are go through in the process of developing that application. The most important stages are:

Analysis: where are identified the main characteristics of all possible correct solutions,

Design: that add to analysis models new elements that define a particular solution, based on some criteria optimizations,

Implementation: where an executable design is built for the particular solution modeled in design phase,

Testing: where is verified the equivalence of the implementation with the designed model and validates the fact that the implementation respects the correctness criteria identified in the analysis phase.

The object oriented analysis and design methods allow an iterative approach of the phases from applications life cycle (Figure 1).

CASE (*Computer Aided Software Engineering*) tools are software products able to support medium or large application development. This support is realised by automating some of the activities made in an analysis and design method. If we agree that one of the main goals of an analysis and design method is code generation and that we should obtain automatically a high rate of application code, it is obvious that an efficient use of a method cannot be made without an associated CASE tool.

Typically, the translation of a complex analysis/design model into a programming language takes a long period. A model is called *executable* if this translation can be made automatically. The automatization of the translation process allows running a prototype of an application immediately after building its model.

This paper captures aspects regarding concurrent object oriented application modeling. We analyzed the main object models developed in literature, insisting on concurrency aspects. In the center of this analysis is UML (*Unified Modeling Language*) version 1.3 [8].

The obtained results and the similarities between active object and reactive systems drive us to the idea of modeling their behavior through statecharts formalism. We extended the *scalable statecharts* formalism, introduced in [13], which allow developing executable models and offers support for automatic source code generation and for simulation of active objects behavior.

The executability is an important feature of *scalable statecharts* [13], allowing the automatization of active objects implementation based on their behavioral models. Furthermore, the executability offers support for simulation, testing and debugging of active object execution at the same level of abstraction like the built model.

2. Level 2 scalable statecharts (SS^2)

SS^1 statecharts defined in [13] do not allow parallel triggering of transitions. Thus SS^1 statecharts cannot be used to model intra-object concurrency. Furthermore, SS^1 statecharts do not provide mechanisms for modeling conditional synchronization and synchronization constraints.

We will extent SS^1 with new elements that allow us to specify state invariants, conditions for transition triggering and to handle more than one message from queue.

Definition 1. *A level 2 scalable statechart of a class K is a tuple:*

$$SS^2_K = (M, S, O, P, E, s_R, S_F, (stSucc, stInit, ortSucc,), inv, T; eval, par, S_a, C)$$

where:

- *M is a finite set of messages,*
- *S is a finite, non-empty set of states,*
- *O is a finite, non-empty set of orthogonal components,*
- *P is a finite set of properties,*
- *$s_R \in S$ is the* root *of the states hierarchy,*
- *S_F is a finite set of final states. To preserve the consistency of our model we will presume that all the final states will be successors of orthogonal components from the root state s_R. Thus we will eliminate the termination transitions proposed in UML* [8] *without affect the modeling power of the statecharts.*
- *functions that defines the states hierarchy:*
  - *$stSucc : O \to \mathcal{P}(S \cup S_F)$, where $stSucc(o) = \{s_1, s_2, \ldots, s_n\}$ is the set of sub-states of the orthogonal component o, with the restriction that $\forall o_1, o_2 \in O$ we have $stSucc(o_1) \cap stSucc(o_2) = \emptyset$;*
  - *$stInit : O \backslash \{o : stSucc(o) = \emptyset\} \to S, stInit(o) = s_0 \in stSucc(o)$, the initial sub-state of the orthogonal component o ($stSucc$ is defined only for non-empty orthogonal components);*
  - *$ortSucc : S \to \mathcal{P}(O) \backslash \{\emptyset\}$, where $ortSucc(s) = \{o_1, o_2, \ldots, o_m\}$ is the set of the orthogonal components owned by state s, with the restriction that $\forall s_1, s_2 \in S$ we have $ortSucc(s_1) \cap ortSucc(s_2) = \emptyset$ (a state has at least one orthogonal component);*
- *$T \subseteq \mathcal{P}(S \backslash \{s_R\}) \times M \times \mathcal{P}(S \backslash \{s_R\})$ is a finite set of transitions. A transition $(\{s'_1, \ldots, s'_i\}, m, \{s''_1, \ldots, s''_j\}) \in T$ means that if an object is in source states $s'_1, \ldots, s'_i \in S \backslash \{s_R\}$ (each source state is located in distinct orthogonal components of a state from S) and receives a message m then, after executing the operation associated to m, the object will enter in* destination *states $s''_1, \ldots, s''_j \in S \backslash s_R\}$. The root state can not be source nor destination for a transition and the sets of source states and destination states not contain states that includes each other.*

- $S_a \subseteq S \cup S_F$ *is the set of* active states *of the statechart in a given moment with the restriction that* $\forall s_a \in S_a, ortSucc(s_a) = \emptyset$,
- $C \in M^\star$ *is a finite sequence of messages, and models the messages queue of an active object.*

Figure 2 contains an example of a SS^0 statechart and its visual representation. The structure of the modeled class (*Bottle*) is defined in the same figure using UML notation.

Based on *stSucc* and *ortSucc* functions we will define another two functions that return the parent of a state or orthogonal component.

Definition 2. *The function* $stPred : O \to S$, *where* $stPred(o) = s \in S$ *if* $o \in ortSucc(s)$, *determines the parent state of an orthogonal component* $o \in O$. *The function* $ortPred : S \cup S_F \backslash \{s_R\} \to O$, $ortPred(s) = o \in O$ *if* $s \in stSucc(o)$ *determines the orthogonal component that is parent of a state* $s \in S \cup S_F \backslash \{s_R\}$.

The restrictions stated in definition 1:

$$\begin{aligned} \forall o_1, o_2 \in O, stSucc(o_1) \cap stSucc(o_2) &= \emptyset \text{ and} \\ \forall s_1, s_2 \in S, ortSucc(s_1) \cap stSucc(s_2) &= \emptyset, \end{aligned}$$

ensure that *stPred* and *ortPred* are well defined.

To complete the formal definition of SS^1 statecharts we will give a formal specification for *valid* transitions. For this reason, we will define first the *nesting relation* between states and/or orthogonal components.

Definition 3. *Two elements* $so_1, so_2 \in S \cup O$ *are in* nesting *relation, denoted by* $so_1 \prec so_2$, *iff one of the above affirmations is true:*

a) $so_1 = so_2$,

b) $so_1 \in S \wedge so_2 \in S \Rightarrow \exists n \in \mathbb{N}^+ : so_2 = \underbrace{stPred(ortPred(\cdots}_{n \text{ times}} so_1 \cdots))$,

c) $so_1 \in O \wedge so_2 \in O \Rightarrow \exists n \in \mathbb{N}^+ : so_2 = \underbrace{ortPred(stPred(\cdots}_{n \text{ times}} so_1 \cdots))$,

d) $so_1 \in S \wedge so_2 \in O \Rightarrow \exists n \in \mathbb{N}^+ : so_2 = \underbrace{ortPred(stPred(\cdots}_{n \text{ times}} ortPred(so_1) \cdots))$

e) $so_1 \in O \wedge so_2 \in S \Rightarrow \exists n \in \mathbb{N}^+ : so_2 = \underbrace{stPred(ortPred(\cdots}_{n \text{ times}} stPred(so_1) \cdots))$.

Proposition 1. *The nesting relation is partial order over* $S \cup O$.

Proof. The reflexivity is assured by the affirmation a) from nesting relation definition.

Let $so_1, so_2, so_3 \in S$ be three states such that $so1 \prec so_2$ and $so_2 \prec so_3$. From definition 7 we have that $\exists n \in \mathbb{N}^+ : so_2 = \underbrace{stPred(ortPred(\cdots}_{n \text{ times}} so_1 \cdots))$ and $\exists m \in \mathbb{N}^+ : so_3 = \underbrace{stPred(ortPred(\cdots}_{m \text{ times}} so_2 \cdots))$. This implies that $\exists r = n + m \in \mathbb{N}^+ : so_3 = \underbrace{stPred(ortPred(\cdots}_{r=n+m \text{ times}} so_1 \cdots))$, so $so_1 \prec so_3$. This means that the nesting relation is transitive over S. Analogous it can be proved that the nesting relation is transitive over $S \cup O$ for so_1, so_2, so_3 belonging to S and/or O.

We will prove that the nesting relation is anti-symmetrical over S.

Let $so_1, so_2 \in S$ be two states for which $so_1 \prec so_2$ and $so_2 \prec so_1$. This implies that:

$$so_1 = so_2,$$

or

$$\exists n, m \in \mathbb{N}^+ : so_2 = \underbrace{stPred(ortPred(\cdots}_{n \text{ times}} so_1 \cdots))$$

and

$$so_1 = \underbrace{stPred(ortPred(\cdots}_{m \text{ times}} so_2 \cdots)).$$

Let us suppose that $so_1 \neq so_2$. Then

$$\exists r = n + m \in \mathbb{N}^+ : so_1 = \underbrace{stPred(ortPred(\cdots}_{r=n+m \text{ times}} so_1 \cdots)).$$

From definition 1 we have that the above statement is true only for $r = 0$. This is obviously impossible because $r \in \mathbb{N}^+$. We deduce that $so_1 = so_2$. The other three cases ($so_1, so_2 \in O$, $so_1 \in O$ and $so_2 \in S$, $so_1 \in S$ and $so_2 \in O$) are analogous.

Thus, $\forall so_1, so_2 \in S \cup O$, $so_1 \prec so_2 \wedge so_2 \prec so_1 \Rightarrow so_1 = so_2$, i.e. the nesting relation is anti-symmetrical over $S \cup O$.

Because the relation $(S \cup O, \prec)$ is reflexive, transitive and anti-symmetrical we deduce that the nesting relation is partial order over $S \cup O$. □

Definition 4. *For a state or orthogonal component $so \in S \cup O$, $\{so' : so' \in S \cup O, so \prec so'\}$, denoted by $PRED_{so}$, is the set of all its predecessors.*

Proposition 2. *For all $so \in S \cup O$, $(PRED_{so}, \prec)$ is total order.*

Proof. Corresponding to proposition 1, the relation $(PRED_{so},)$ is partial order. Let $so', so'' \in PRED_{so} \cap S$ be two predecessor states of so. According to definition 8 we have:

$$\exists n' \in \mathbb{N}^+ : so' = \underbrace{stPred(ortPred(\cdots}_{n' \text{ times}} so \cdots))$$

and

$$\exists n'' \in \mathbb{N}^+ : so'' = \underbrace{stPred(ortPred(\cdots}_{n'' \text{ times}} so \cdots)).$$

We suppose that $n' > n''$. We have:

$$\exists n'' \in \mathbb{N}^+ : so'' = \underbrace{stPred(ortPred(\cdots}_{n'-n\text{" times}} so' \cdots))$$

that implies $so' \prec so''$. The other three cases ($so', so'' \in PRED_{so} \cap O$, $so' \in PRED_{so} capO$ and $so'' \in PRED_{so} \cap S$, $so' \in PRED_{so} \cap S$ and $so'' \cap PRED_{so} \cap O$) are analogous.

Thus, $\forall so', so'' \in PRED_{so}$, $so' \prec so''$ or $so'' \prec so'$, which implies $(PRED_{so}, \prec)$ is total order. □

Definition 5. *Let $(X, \prec)$ be a partially ordered set and let Y be a subset of X. An element $x \in X$ is a* lower bound *for Y iff $x \prec y$ for all $y \in Y$. A lower bound x for Y is the* greatest lower bound *for Y iff, for every lower bound x' for Y, $x' \prec y$. Whet it exists, we denote the greatest lower bound for Y by $\sqcap Y$.*

In the paper we use the following three well known results [9]:

- if x is a lower bound for Y and $x \in Y$ then $\sqcap Y = x$;
- if $\sqcap Y$ exists then it is unique;
- if $(Y, \prec)$ is total order and Y is finite then $\sqcap Y$ exists and $\sqcap Y \in Y$.

Because $(PRED_{so}, \prec)$ is total order and $PRED_{so}$ is a finite set, we deduce that the greatest lower bound for $PRED_{so}$ does exist, and $\sqcap PRED_{so} \in PRED_{so}$. We will prove that $\sqcap PRED_{so}$ is the parent of so.

Proposition 3. *Let $so \in S \cup O$ be a state or orthogonal component. One of the following affirmations is true:*

1) $so \in S \Rightarrow ortPred(so) = \sqcap PRED_{so}$,
2) $so \in O \Rightarrow stPred(so) = \sqcap PRED_{so}$.

Proof. a) Let $so \in S$ be a state. It is obvious that $so \prec ortPred(so)$, and based on the definition of set $PRED_{so}$ we have that $ortPred(so) \in PRED_{so}$.

Let $so' \in PRED_{so}$ be an arbitrary predecessor of the state so. From definition 8 we have that $so \prec so'$. If so' is an orthogonal component ($so' \in O$) then:

$$\exists n \in \mathbb{N} : so' = \underbrace{ortPred(stPred(\cdots}_{n \text{ times}} ortPred(so) \cdots)),$$

which implies that $ortPred(so) \prec so'$. The case when so' is a state ($so' \in S$) is analogous. Because so' was arbitrary selected from PREDso we will have:

$$\forall so' \in PRED_{so}, ortPred(so) \prec so'$$

. that implies $ortPred(so) = \sqcap PRED_{so}$.

The proof for statement b) is analogous. □

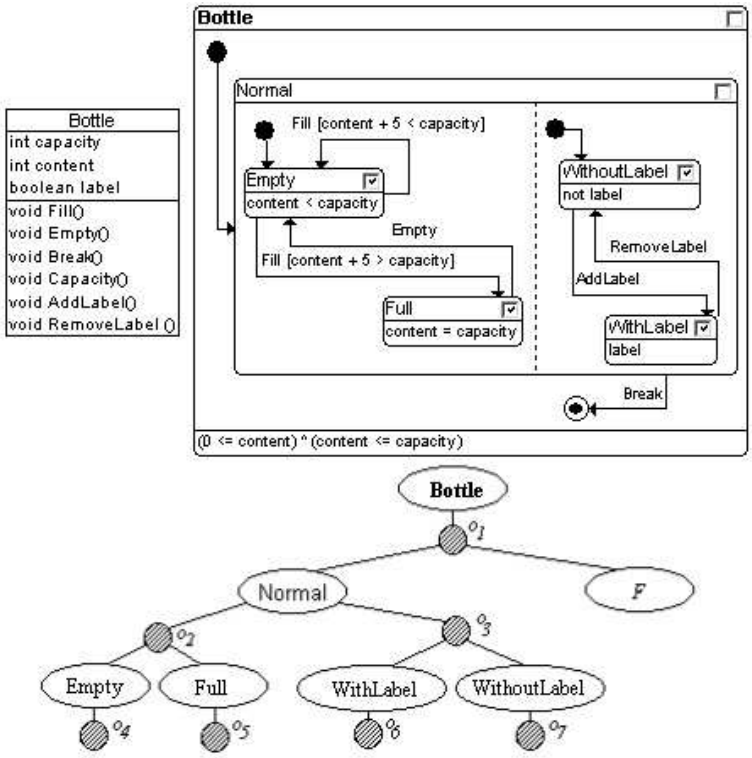


FIGURE 2. Graphical representation of SS^2 statechart

Definition 6. *Two states or orthogonal components* $so', so'' \in S$ *are* orthogonal *iff* $so' \not\prec so''$, $so'' \not\prec so'$ *and* $\sqcap(PRED_{so'} \cap PRED_{so''}) \in S$.

In other words, two states or orthogonal components are *orthogonal* if they are not in nesting relation and the closest common ancestor is a state.

Definition 7. *Let* $t = (\{s'_i \in S : i = 1, \ldots, n\}, m, \{s''_j \in S : j = 1, \ldots, m\}) \in T$ *be a transition. We say that* t *is a valid transition if all the following affirmations are true:*

a) $Ps' = \sqcap \bigcap_n^{i=1} PRED_{s'_i} \in S$ *(the source states are orthogonal),*

b) $Ps'' = \sqcap \cap_m^{j=1} PRED_{s''_j} \in S$ *(the destination states are orthogonal),*

c) $Ps' \not\prec Ps''$, $Ps'' \not\prec Ps'$ *and* $\sqcap(PRED_{Ps'} \cap PRED_{Ps''}) \in O$ *(source and destination states are not orthogonal).*

We will call $dom_t = \sqcap(PRED_{Ps'} \cap PRED'_{Ps'}) \in O$ *the* domain *of transition* t.

The domain of a transition represents the "smallest" orthogonal component that contains all transition's source and destination states.

In definition 1 function *par* characterizes the algorithm of choosing a set of messages from message queue. The specification of *par* function is not important in this phase of formalization and is imposed by particular mechanisms implemented in various concurrent object oriented languages. We consider that this function will return the maximal set of messages that can be handled concurrently.

Definition 8. *Two transitions* $t', t'' \in T$ *are textslindependent iff their domains are orthogonal, i.e.,* $\sqcap(PRED_{domt'} \cap PRED_{domt''}) \in S$.

Definition 9. *A* configuration *of a* SS^2 *statechart is a tuple* $(S_a, par(C), C_r)$, *where* $S_a \subseteq S$ *is the finite set of active states,* $par(C)$ *is the set of messages from queue which will be processed in parallel and* $C_r \in M^\star$ *the rest of messages queue* C *after removing messages from* $par(C)$. *The* initial configuration *of a* SS^2 *statechart if given by* $(active(s_R), \perp)$.

Definition 10. *The* interpretation *of a* SS^2 *statechart configuration is a function:*

$$\delta^2 : \mathcal{P}(S) \times \mathcal{P}(M) \times M^\star \to \mathcal{P}(S \cup S_F) \times M^\star,$$

$$\delta^2(S_a, \{m_1, \ldots, m_n\}, C_r) =$$

$$= \begin{cases} (Activ(\bigcup_n^{i=1} S_i''), C_r'), & \text{if } \forall i \in \{1, \ldots, n\} \exists (S_i' \subseteq S_a \cup S_{pa} \text{ and } eval(e_i) = true \\ (S_a, C_r'), & \text{if } \forall i \in \{1, \ldots, n\} \nexists S_1, S_2, S_2 \subseteq S_1, e \in E : (S_1, m_i, e, S_2) \in T \\ (S_a, C_r' \wedge m_1 \wedge \cdots \wedge m_n), & \text{else} \end{cases}$$

Definition 11. *The* execution *of a* SS^2 *statechart is a sequence finite or infinite of configuration interpretations, starting from the initial configuration, and is denoted:*

$$(active(s_R), \emptyset, \perp) \xrightarrow{\delta^2} (S_1, par(C), C_{r1}) \xrightarrow{\delta^2} \cdots \xrightarrow{\delta^2} (S_k, par(C), C_{rk}) \xrightarrow{\delta^2} \cdots$$

where $S_1, \ldots, S_k, \ldots \subseteq S$, $m_1, \ldots, m_k, \ldots \in M$ *and* $C_{r1}, \ldots, C_{rk}, \ldots \in M^\star$. *The execution is* finite *if the set of activated states contains at least a final state.*

3. Conclusions

We extended the statecharts formalism [7] with new semantically and graphical elements, in order to allow the specification of active objects behavior with respect of a general concurrent object model. The extensions are: allowing scalability, executability and the definition of a precise semantic.

The formalism that is proposed in section two of this paper is called *level two scalable statechart*. The scalability of states minimizes the effort of modeling objects with a complex behavior. In this way, the active objects behavior models can be analyzed at different levels of detail.

Because the semantic of scalable statecharts was defined regarding a general concurrent object model, they allow source code generation in various concurrent

object-oriented languages that use various modalities and mechanisms for specification of concurrency and interaction between concurrent activities. This thing confers a better flexibility in translation of behavioral models in source code.

References

[1] F. Barbier, H. Briand, B. Dano, S. Rideau, "The Executability of Object-Oriented Finite State Machines", Journal of Object-Oriented Programming, SIGS Publications, 4 (11), pp. 16–24, jul/aug 1998

[2] Michael von der Beeck, "A Comparison of Statecharts Variants", Formal Techniques in Real-Time and Fault-Tolerant Systems, L. de Roever and J. Vytopil (eds.), Lecture Notes in Computer Science, vol. 863, pp. 128–148, Springer-Verlag, New York, 1994

[3] S. Cook, J. Daniels, "Designing Object Systems - Object-Oriented Modelling with Syntropy", Prentice Hall, Englewood Cliffs, NJ, 1994

[4] Bruce Powel Douglas, "UML Statecharts", Embedded Systems Programming, jan. 1999, available at http://www.ilogix.com/fs_prod.htm

[5] D. Harel, A. Naamad, "The STATEMATE Semantics of Statecharts", ACM Transactions on Software Engineering and Methodology, 5 (4), pp. 293–333, 1996

[6] D. Harel, E. Gery, "Executable Object Modeling with Statecharts", IEEE Computer, 30 (7): 31–42, Jul. 1997

[7] David Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, vol.8, no. 3, pp. 231–274, June 1987

[8] Object Management Group, OMG Unified Modeling Language Specification, ver. 1.3, June 1999 available on Internet at http://www.rational.com/

[9] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, 1974

[10] Michael Phillipsen, Imperative Concurrent Object-Oriented Languages, Technical Report TR-95- 049, International Computer Science Institute, Berkeley, Aug. 1995

[11] Marian Scuturici, Dan Mircea Suciu, Mihaela Scuturici, Iulian Ober, Specification of active objects behavior using statecharts, Studia Universitatis "Babes Bolyai", Informatica, Vol. XLII, no. 1, pp. 19–30, 1997

[12] Dan Mircea Suciu, Reuse Anomaly in Object-Oriented Concurrent Programming, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLII, no. 2, pp. 74–89, 1997

[13] Dan Mircea Suciu, Extending Statecharts for Concurrent Objects Modeling, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLIV, No. 1, pp. 37–44, 1999

Department of Computer Science, "Babeş-Bolyai" University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

E-mail address: tzutzu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

TERM REWRITING SYSTEMS IN LOGIC PROGRAMMING AND IN FUNCTIONAL PROGRAMMING

DOINA TĂTAR, GABRIELA ŞERBAN

Abstract. Automated theorem proving and term rewriting system are fields with big interest since some years. Often these fields have a common development. Is it not amazingly that logic programming and functional programming, which belongs to both these fields, offers simple solutions to problems arising at the frontier of them. In [8], the author submitted a challenge for "finding an optimum way to implement the rewriting systems ". This paper presents the way in that the logic programming and functional programming offer their concision to realize a sound implementation of the TRS.

1. Introduction

In the first section we will presents shortly the equation systems, the TRS, the "critical pair" idea and the completion algorithm [1, 5, 7, 10]. In the following sections we will outline some problems and their solution in our implementation in Prolog (section 2) and in Lisp (section 3).

Definition 1 An equational theory (F,V,E) consists of:

- a set F of function symbols (with the same sort, for simplicity).
- a set V of variables.

Let T(F,V) be the set of terms build from F and V.

- a set of pairs of equations, s=t,s,t $\in$ T(F,V).

The set of equations E defines a syntactical equality relation $==_E$ on T(F,V), usually defined as "replacing equals by equals".

The fundamental problem in an equational theory is the "validity" or "word problem", which is undecidable:

"Give s and $t \in$ T(F,V), does $s ==_E$ t ?"

The undecidability (more precisely, the semidecidability) of the "word problem" is transferred on the approach by the TRS, but this approach is, on the our opinion, more algorithmically.

2000 *Mathematics Subject Classification.* 68T15.

1998 *CR Categories and Descriptors.* D.1.6. [**Software**] : Programming Techniques – *Logic Programming*;I.2.3. [**Computing Methodologies**] : Artificial Intelligence – *Deduction and Theorem Proving.*

Definition 2. A TRS is a set of rules: $R = \{l \to r \mid l, r \in \mathrm{T(F,V)}$, every variables occurring in term r also occurs in term l $\}$.

A TRS defines a rewrite relation $\to_R$:

Definition 3. $s \to_R t$ iff there is a rule $l \to r \in R$ and an occurrence p in s such that the subterm of occurrence p, noted $s \mid_p$ and the term t have the property:

$$s \mid_p = \sigma(l), t = s[p \leftarrow \sigma(r)]$$

for some substitution σ. Here notation $s[p \leftarrow \sigma(r)]$ represents the term obtained from s by replacing the subterm of occurrence p by the term $\sigma(r)$.

We denote by $\to_R^*$ and $\leftarrow\!\!\to_R^*$ the reflexive-transitive and reflexive-transitive-symmetric closure of $\to_R$.

In order to solve the "word problem" for an equational theory E, compute an TRS R_E such that $s =\!=_E t$ is a relation equivalent with $s \leftarrow\!\!\to_R^* t$. Let us denote R_E as *associated with* E.

The TRS R_E is the canonical (terminating and confluent) TRS *associated with* E, obtained as output of the completion procedure Knuth -Bendix. This algorithm has as input the set E and a reduction order over T(F,V).

Definition 4 The normal form of a term t , denoted $t \downarrow_R$, is a term with the followings properties:

$$1. t \to_R^* t \downarrow_R$$
$$2. t \downarrow_R \textit{ irreducible.}$$

Observations:

1. If a TRS R has the property that every term has a unique normal form, then:

$s \leftarrow\!\!\to_R^*$ iff $s \downarrow_R = t \downarrow_R$, because $s \leftarrow\!\!\to t_R^*$ is $s \to_R^* s \downarrow_R$ and $t \to_R^* \downarrow_R$. Thus, testing $s \leftarrow\!\!\to t_R^*$ is is the same as testing that $s \downarrow_R = t \downarrow_R$.

2. In a canonical TRS ,R, every term has a unique normal form.

We won't describe the well known Knuth-Bendix algorithm. Instead, we will survey the critical pair idea, staying on the ground of this algorithm.

Definition 5 Let $l_1 \to r_1$ and $l_2 \to r_2$ be two rules in R. By renaming the variables we may assume that they do not share common variables. If $\sigma_1(l_1) = \sigma_2(l_2)$, then the pair of terms $(\sigma_1(l_1), \sigma_2(l_2))$ is a critical pair for R.

The Knuth-Bendix algorithm computes, for every critical pair (t_1, t_2) of R', the normal forms $t_1 \downarrow_{R'}$ and $t_2 \downarrow_{R'}$. If this normal forms are different, then a rule $t_1 \downarrow_{R'} \to'_R t_2 \downarrow_{R'}$ or converse, (depending of the case $t_1 \downarrow_{R'} > t_2 \downarrow_{R'}$ or the converse), is added to R'. Let observe that the procedure fails if neither $t_1 \downarrow_{R'} > t_2 \downarrow_{R'}$ nor the converse is true.

2. Implementation in Prolog

A set of problems for implementation in Turbo Prolog derives from the fact that in this language does not exist the standard predicates **functor**, ==, and **op**. This fact lead as construct two specific domains in section **domains** of our programs as follows:

```
domains
  term=var(symbol);con(symbol);cmp(symbol,terml)
  terml=term*
  termll=terml*
```

For example, if we must introduce the term **f(x,y,a)** , we will write:

cmp(f,[var(x),var(y),con(a)]), respecting the conventions for syntax of formulas in first-order logic. Also,if we must introduce the formula **p(x,f(y,z))** we will write:

atom(p,[var(x),cmp(f,[var(y),var(z)])]) .

A TRS R of I rules, as in definition 2, is done by a couple of predicates **l(t,N)** and **r(t,N)** where **t** is a term and N=1, $\cdots$,I is the index of the rule. We worked in this program with the three starting rules associated with the theory E of groups.

l(cmp("f",[con(e),var(a)]),1).
l(cmp("f",[cmp("g",[var(a)]),var(a)]),2).
l(cmp("f",[cmp("f",[var(a),var(b)]),var(c)]),3).
r(var(a),1).
r(con(e),2).
r(cmp("f",[var(a),cmp("f",[var(b),var(c)])]),3).

The predicates which realizes the rewriting relation $X \to Y$ with a rule N in definition 3 is the predicate **rewrite (X,Y,N)** .

```
rewrite(X,Y,N):-l(X,N),r(Y,N),!.                              (1)
rewrite(X,Y,N):-member_left(X,L1,L2,N),                       (2)
                list_var(X,L_var),                            (3)
                lg_list(Lnou,K),                              (4)
                l(M_stg,N),                                   (5)
                aplic_subst(M_stg,Nou_m_stg,L1,L2),           (6)
                tr_term_str(Nou_m_stg,St_stg),                (7)
                aplic_subst(X,NouX,L1,L2),                    (8)
                list_var(NouX,L_var_n),                       (9)
                lg_list(Lnoun,K),                            (10)
                tr_term_str(NouX,St),                        (11)
                r(M_dr,N),                                   (12)
                aplic_subst(M_dr,Nou_m_dr,L1,L2),            (13)
                tr_term_str(Nou_m_dr,St_dr),                 (14)
                strsr_first(St,St_stg,St_dr,Nou_string),     (15)
                tr_str_term(Nou_string,Yinterm),             (16)
                sc_lista(L2,L1,L2nou,L1nou),                 (17)
                aplic_subst(Yinterm,Y,L2nou,L1nou),!.        (18)
```

The predicate **member-left** (denoted by (1)) is defined as follows:

```
/*  member_left(X,L1,L2,N):-the rule  N-th has the property that
    his left side unifies with a subterm of term X, and the unifier
    has the domain L1 and the codomain L2.    */
```

One of the clauses for **member-left** must be:

```
member_left(X,L1,L2,N):-subterm(S,X),
                        l(Z,N),
                        unify(S,Z,L1,L2).
```

The predicate **aplic-subst(t,s,L1,L2)** denoted by (6) applies the substitution σ =(L1/L2) to t obtaining s. The predicates **tr-term-str** transforms a term (e.g. f(a,x)) in a string (f2ax). The reason for this transformation is to provide to predicate:

strsr-first (St,St-stg,St-dr,Nou-string), denoted by (15),
his first three arguments (the lines (7),(11),14)). Thus, one step of the realization of the relation $\rightarrow$ is accomplished by the predicate **strsr-first**. This is defined as:

```
/* strsr-first(S1,S2,S3,S):- the string S is obtained by
   replacing in the string S1 the first occurrence of the
   substring  S2 by the string S3. */
```

The converse transformation of a string into a term is realized by the predicate **tr-str-term** (16). A clause for this one must be:

```
tr_str_term(X,Y):-str_len(X,L),L>0,frontstr(1,X,Z,U),
                  frontstr(1,U,N,W),
                  str_int(N,N1),
                  frontstr(N1,W,WW,WWW),
                  tr_str_terml(WW,V),
                  tr_str_terml(WWW,V1),
                  append(V,V1,V2),
                  Y=cmp(Z,V2),lg_list(V2,N1).
```

The relation $\rightarrow_R^*$ defined as the reflexive -transitive closure of $\rightarrow_R$ is realized by the predicate **rewrite***. The clauses for this predicate are:

```
rewrite*(X,Y):-rewrite(X,Y,N).

rescrie*(X,Y):-rewrite(X,Z,N),!,rewrite*(Z,Y).
```

The predicates **critical-pair** and **normal-form** are defined as:

```
critical-pair(X,Y):-l(X,N),member_left(X,L1,L2,M),l(Z,M),
                    aplic-subst(Z,Y,L1,L2).
normal-form(X,Y):-rewrite*(X,Y),not(rewrite(Y,_,_)).
```

At the end of the application of the Knuth-Bendix algorithm, the canonical TRS is given as usually by 10 rules. (Some intermediary rules are deleted because they have been rewritten in the same terms.) The obtained canonical TRS can be used for demonstrate some theorem in group theory. For example, if we want to prove that $t_1 = i((i(a) + a) + (b + i(b)))$ is equal with $t_2 = b + (i(a + b) + a)$, then we run the program with **normal-form**(t_1,X) and **normal-form**(t_2,Y). We will obtain X=Y.

3. Implementation in Lisp

In this section our aim is to present how the rewriting relations could be defined in LISP.

3.1. **LISP representations.** First, we have to establish the way in which the terms are represented in LISP.

- a variable **x** is represented as a list **(var x)**;
- a constant **a** is represented as a list **(con a)**;
- a functional symbol **f** is represented as a list **(cmp f)**;
- a function **f(LA)** where **f** is a functional symbol and **LA** is a list of arguments, is represented as a list **((the list corresponding to f) (the list of arguments))**; for example, **f(a,x)** is represented as a list **((cmp f) ((con a) (var x)))**.

With the above considerations, if we must introduce the term **g(x,f(y,z))** we will write ((cmp g) ((cmp f) ((var y) (var z)))).

A rule **l→r** from a TRS is represented as a list **(list-l list-r)**, where **list-l** and **list-r** are the representations in LISP of the terms **l** and **r**. For example, a rule $\mathbf{f}(\mathbf{a},\mathbf{x})\rightarrow\mathbf{x}$ is represented as the list **(((cmp f) ((con a) (var x))) (var x))**.

A TRS R of N rules is represented as a list of rules **(rule-1 rule-2 . . . rule-N)**, each rule is represented as we described above.

In the followings, we work with the three starting rules associated with the theory of groups. The list of rules is denoted by **LR** and is the following:

```
(setq LR '(
    ( ((cmp f) ((con e) (var a)))
      (var a)
    )
    (
      ((cmp f) (((cmp g) ((var a))) (var a)))
      (con e)
    )
    (
      ((cmp f) (((cmp f) ((var a) (var b))) (var c) ))
      ((cmp f) ((cmp f) ((var a) ((cmp f) ((var b) (var c))))))
    )
  )
)
```

3.2. **Functions defined for rewriting rules.** The functions which realize the rewriting relation **X→Y** with a rule N in definition 3 is the function **(rewrite X N LR)** which returns Y.

```
(defun rewr (X N LR)
; LR represent the list of rules
  (prog (RN)
```

```
    (setq RN (rule-N N LR))
    (cond
       ((equal (car RN) X) (return (cadr RN)))
       (t (setq Y (cadr RN))
          (setq UNIF (member-left X (car RN)))
          (cond
            ((null UNIF) nil)
            (t
                (setq L1 (car UNIF))
                (setq L2 (cadr UNIF))
                (return (apply-subst L1 L2 Y))
            )
          )
       )
    )
  )
)
```

The function **(rule-N N LR)** returns the N-th rule from the list of rules LR.

```
(defun rule-N (N LR)
  (cond
      ((null LR) nil)
      ((= N 1) (car LR))
      (t (rule-N (- N 1) (cdr LR)))
  )
)
```

The function **(member-left X Y)** is defined as follows:

- if Y (the left side of a given rule) unifies with a sub-term of X, and the unifier has the domain $L1$ and the codomain $L2$, then the function returns the list **(L1 L2)** (this list is calculated by the function **(unify X Y)**);
- else the function returns **NIL**.

```
(defun member-left (X Y)
  (cond
      ((not (equal (length X) (length Y))) nil)
      (t (unify X Y))
  )
)
```

The function **(apply-subst L1 L2 Y)** applies the substitution $\sigma = (L1/L2)$ to Y and returns the result.

```
(defun apply-subst (L1 L2 Y)
   (subst Y L1 L2)
)
```

The function **(rewrite X)** is defined as follows:

- returns a list of elements having the form **(N Y)**, where Y is the right side of the rewriting relation **X**→**Y** with the rule N (if it is possible) - this list is calculated by the recursive function **(rewrite-rule X N LR)** which returns the result of rewriting X with the N-th rule of LR;
- returns NIL, if no rewriting relations for X are possible.

```
(defun rewrite-rule(X N LR)
  (cond
    ((> N (length LR)) nil)
    (t
       (setq RN (rewr X N LR))
       (cond
         ((not (null RN)) (cons (list N RN)
                                (rewrite-rule X (+ N 1) LR)
                          )
         )
         (t (rewrite-rule X (+ N 1) LR))
       )
    )
  )
)

(defun rewrite (X)
  (rewrite-rule X 1 LR)
)
```

The relation defined as the reflexive-transitive closure of the rewriting relation R is defined as the function **(rewrite* X)**.

```
(defun rewrite* (X)
  (setq Y (rewrite X))
  (append Y (rewr* Y))
)

(defun rewr* (Y)
 (cond
   ((null Y) nil)
   (t (append (rewrite (cadar Y)) (rewr* (cdr Y))))
 )
)
```

The **normal-form** is defined as a function **(normal-form X)**.

```
(defun normal-form (X)
  (n-form (rewr* X))
)
```

```
(defun n-form (Y)
  (cond
    ((null Y) nil)
    ((null (rewrite (cadar Y))) (append (car Y) (n-form (cdr Y))))
    (t (n-form (cdr Y)))
  )
)
```

Examples

(1) if X is ((cmp f) (((cmp g) ((var b))) (var b))), then the result of rewriting X este ((2 (CON e)));

(2) if X is ((cmp f) ((con e) (var b))), then the result of rewriting X este ((1 (VAR b)));

(3) if X is ((cmp f) ((con e) (var a))), then the result of rewriting X este ((1 (VAR a))).

(4) if X is ((cmp f) (((cmp f) ((var a) (var b))) ((cmp g) ((var c))))), then the result of rewriting X este (3 ((cmp f) ((cmp f) ((var a) ((cmp f) ((var b) ((cmp g) ((var c)))))))).

References

[1] Avenhaus J., Madlener K. : "Term rewriting and Equational Reasoning" in Formal Techniques in A.I., A coursebook, R.B.Banerdji (ed) 1990.

[2] K.H. Blasius, H.J. Burkert: "Deduction systems in Artificial Intelligence", Ellis Horwood Ltd.,1989.

[3] Buchberger B.: "History and basic features of the Critical-Pair Completion Procedure", J. of symbolic Computation 3, 1987, pp. 3–38.

[4] W.F. Clocksin, C.S. Mellish : Programming in Prolog, Springer-verlag, 1984.

[5] Huet G., Oppen D.D.: "Equations and rewrite rules: A survey", in "Formal languages: theory, perspectives and open problems", ed. R. Book, 1980.

[6] Jouannaud J.P., Lescanne P.: "Rewriting Systems", in Technology and Science of Informatics, 1987, pp. 181–199.

[7] Knuth D.E., Bendix P.P.: "Simple word problem in Universal Albgebra", Comp. prob. in Abstr. Alg. (ed. J. Leech), 1970.

[8] Lescanne P.: "Current trends in rewriting techniques and related Problems", IBM int. symp. on Trends in Computer Algebra, Germany, 1987.

[9] Rusinowitch M.: "Demonstration automatique. Techniques de reecriture" Inter. Edition, Paris, 1989.

[10] Tatar D.: "A new method for the proof of theorems", Studia Universit. Babes-Bolyai, Mathematica, 1991, pp. 83–95.

[11] Tatar D.: "Term rewriting systems and completion theorems proving: a short survey", Studia Univ. Babes-Bolyai, Mathematica, 1992, pp. 117–125.

Department of Computer Science, Faculty of Mathematics and Computer Science, "Babeş-Bolyai University, 1, M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

FORMAL MODEL FOR SOFTWARE SYSTEMS COMPOSITION

IUGA MARIN

Abstract. In this paper we have provided a formal model for software systems specification and for the software systems composition operation. Using the notion of information system as a basis, we can model any information system using both software services and software interfaces. Doing this, we can develop a formal model for software systems composition. This formal model may be used both in formal specification of software systems (structure, functionality, requirements) and in software systems composition expressions.

1. An overview of software system notion

The history of "software system" notion is full of controversies and debates over what is central in the process of defining a software system. At first, a software system was identified with an executable program, but this definition has been enlarged later when a software system was associated with an executable program and its modules. Sooner, this definition has proven to be incomplete because the notion of software system has a larger range than that given by any program, no matters how large or complex this program is.

As a consequence, the definition of a software system has changed its center from the notion of executable programs and modules to the notion of software services and software systems inter-relations.

A radical change of perspective over the software systems is presented in [9]:

"Large software systems are non-algorithmic, open and distributed:

non-algorithmic: they model temporal evolution by systems of interacting components

open: they manage incremental change by local changes of accessible open interfaces

distributed: requirements as well as components are locally autonomous."

A system is generally considered to be a collection of components organized to fulfill a certain function or a certain set of functions. A software system is viewed as an entity that requests software services from the external environment and exports other software services to this environment. We will try to describe a software system without any need of information about its internal construction.

2000 *Mathematics Subject Classification.* 03B70,68N30.

1998 *CR Categories and Descriptors.* C.0. [**Computer Systems Organization**] : General.

It suffices to say that a software system has an internal state, represented by a set of abstract values, but we don't need to know how the state and the mechanism of state changing is implemented inside the system.

The classical software system concept is now replaced by the concept of *extensible system* (see [10]). An extensible system is considered to be a kind of software system whose functionality may be freely extended by replacing existing components with new ones. Smalltalk is an extensible language/system, and new additions to Java make it possible to create extensible systems in Java. Extensible systems cannot be created in more traditional languages such as Simula and C++. However, Active X from Microsoft, allows programming of extensible systems in C++, Visual Basic and other languages.

2. Modeling software systems using software services

We can observe now the fact that the definition of a software system is centered over the notion of software service, thus making the definition of software service the key to define the notion of software system. We will define the *software service* as a set of *operations* grouped under the same identifier. This identifier is the software service's identifier.

An *operation* is defined by a name, and a textual, rather informal, description of it.

Considering this, an *operation* could be represented as:

$$operation = (operation\_signature, operation\_description)$$

where:

operation_signature: is the operation's signature;
operation_description: is the operation's description.

We will provide a formal representation for an operation in this paper.

Once we can specify an operation, we are able to represent a *software service* as:

$$service = (service\_name, \{service\_operation_i, i \in 1, \ldots, num\_operations\})$$

where:

service_name: is the name of the software service,
service_operation$_i$: is the i-th operation of the software service
num_operations: is the number of operations associated with this software service.

A software service could be easily identified as a contract between a provider and a client. It specifies the terms of information exchange between the provider and the client, it specifies a protocol that makes the service provider and the client to understand each other and it specifies the conditions that must be met for the information exchange process.

As an example let's consider the process of a COM object serialization. The serialization is defined as "the ability of an object to write its state to a persistent

storage" [6]. So, if we want a persistent COM object then this object must implement the service specified by *IPersistStorage* (at least). We will call this service as the *Persist_Storage* service, and it is characterized by the following operations:

IsDirty: indicates whether the object has changed since it was last saved to its current storage;

InitNew: initializes a new object, providing a handler to the storage to be used for the object;

Save: saves an object, and any nested objects that it contains, into storage;

SaveCompleted: notifies the object that it can revert from NoScribble or HandsOff mode, in which it must not write to its storage object, to Normal mode, in which it can;

HandsOffStorage: instructs the object to release all storage objects that have been passed to it by its container and to enter HandsOff mode, in which the object cannot do anything and the only operation that works is a close operation.

We can define a software system by the following quadruple:

$$(IN\_STATUS, OUT\_STATUS, IN, OUT)$$

where:

IN_STATUS: represents the system's internal status;

OUT_STATUS: represents the external environment's status that is accessed or modified by the system's services;

IN: represents the set of imported services that are needed by the system in order to fulfill its functionality;

OUT: represents the set of the exported services that are used by the software system to express its functionality.

As a synthesis of what we have exposed until now, we will consider a software system to be characterized by the following features:

- a series of software services exported to an external software environment;
- a series of software services imported from an external software environment;
- an internal state which could be changed as a result of a software service fulfillment;
- a software service execution could change the status of the external environment.

We are close to the model for a software component, introduced in [7], where the component is characterized by a service interface, a client interface and an implementation. Since the black-box model is adopted for a software component (excluding any information about internal implementation and imported services), we find the essence of this model applicable to software systems.

We denote by $OUT_1, \ldots, OUT_n$ the exported software services, where n is the number of exported services and we denote by $IN_1, \ldots, IN_m$ the imported software

services where m is the number of the imported software services. Also we will denote by:

$$IN\_STATUS = \{IN\_STATE_1, \ldots, IN\_STATE_p\}$$

the set of the values of the software system status affected by the software services execution, and by:

$$OUT\_STATUS = \{OUT\_STATE\_1, \ldots, OUT\_STATE_q\}$$

the set of the values of the external software environment status affected by the software services execution.

We consider the external software environment to be divided into two parts, the first part denoted by IN exports software services to the software system, denoted by $SYSTEM$, and the second denoted by OUT is the part which imports the software services exported by $SYSTEM$. Both parts could be identified as a standalone software system. The first representation of the interaction of a software system with the external software environment, using software services, is given in Figure 1:

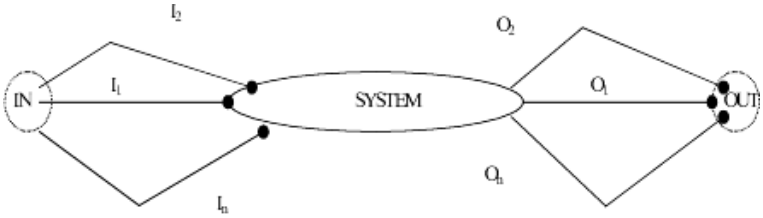


FIGURE 1. Representation of the interaction between a software system and its external software environment using software services

Let's consider, as an example, a software system, called *DataProcessor*, which receives data from an external data source, process it, and displays it to a display.

The imported services for this system are *DataProvider* service (imported form a data source system) and *DisplayRenderer* (imported from a graphical device system).

DataProvider service is characterized by the following operations:

OpenConnection: opens a connection with the data source;
CloseConnection: closes the connection with the data source;
GetData: obtains the raw input data.

DisplayRender service is characterized by the following operations:

ClearDevice: clears the content of the graphical device;
RenderImage: renders a graphical image.

The *DataProcessor* system exports the *DataProcessing* service, which is characterized by the following operations:

CheckValidity: checks the validity of input data;

ProcessData: processes the input data.

The internal status for the system is:

$$IN\_STATUS = \{idle, operation\_completed, operation\_ready\}$$

where:

idle: DataProcessor system is idling;
operation_completed: DataProcessor system has just finished an operation and is ready to provide output data;
operation_ready: DataProcessor system has received valid input data and is ready to begin a processing session.

The external status for the system's external environment is:

$$OUT\_STATUS = \{(connected), (not\_connected)\}$$

where:

connected: the data source has accepted connection and is ready to provide input data;
not_connected: the data source has not accepted a connection, the connection is closed or it is not ready to provide any input data.

As we may see from this example, the software service is only a feature that characterizes a software system and a software system could be viewed as a node that imports some services and exports other services.

3. Modeling software systems using software interfaces

By using a formal specification for a software service (as a interface implementation) we can obtain a formal representation for a software systems (as a set of interface implementations). In this kind of specification we must represent how the status of the external software environment and the status of the software system are affected by the software services execution.

The contract between a software entity and its external environment must be specified in a neutral language and there is needed a contract that will stipulate the terms and limits of the information transaction. In [5] we have specified the fact that the contract that supervises the information transaction should be based on the notion of software interface and the software interface must be specified in a programming language neutral manner.

However, other authors have different points of view about the neutrality of an element specification. They consider the specification of an element (type, interface, class, component, ...) as an abstract description of it, and a program (or module) as the concrete description of this element. In [2] it is requested that any software specification must be in an executable format, but it is hard to agree with this.

For a long time, a software service has been modeled as an interface. This kind of model ignores the fact that an interface can be identified only with the specification of a protocol for a set of operations (the syntactic part) and cannot

capture the meaning of this operation (the semantic part). So, it is properly to discuss a service by the means of the implementation of an interface.

So, we will propose to use the interface implementations as a model for a software service, rather then using only interfaces. The interfaces are sets of method signatures and carry only the syntactic information, while the interface implementations are sets of methods and carry semantic information (behavioral specifications). There are many ways to specify a method by using predicate calculus, functional methods and non-functional methods.

We will propose here a specification model that is based on the predicate calculus. We will specify a method as:

$$(signature, precondition, postcondition)$$

where:

signature: is the method's signature;
precondition: is the method's precondition predicate;
postcondition: is the method's postcondition predicate.

The method's signature is represented as:

$$return\_typemethod\_name(in\_status, out\_status, [par\_rolepar\_name : par\_type])$$

where:

return_type: is the method's return type;
method_name: is the method's name;
in_status: represents the IN_STATUS for the software system to whom the method are bounded to, via its associated interface;
out_status: represents the OUT_STATUS for the software system to whom the method are bounded to, via its associated interface;
par_role: is the parameter's role (could be in, out, inout);
par_name: is the parameter's name;
par_type: is the parameter's type.

For a method m, we will consider the following sets:

- $IN(m)$ = {the set of all in or inout parameters}∪{in_status, out_status};
- $OUT(m)$ = {the set of all parameters}∪{in_status, out_status}∪{result – the value returned by this method}.

The precondition predicate is defined over values from $IN(m)$ and it is true if these values represents valid input data, and false otherwise.

The postcondition predicate connects the input data with the output data, and is true if the returned values are those expected (if valid input data is considered for the actual parameters of the method).

All that we have to remember is the fact that an interface implementation specification must consider the mechanism of state changing associated with the system that implements the interface. As a consequence of this thing, not all interface implementations could be attached to any software system. A software system that implements this interface must accept the values of the state changed

by this interface implementation. We will denote by I_j the interface that has its implementation specified by the software service IN_j and with O_i the interface that has its implementation specified by the software service OUT_i. Using the name of the interface to designate the interface implementation associated with the service, we will have another representation of the interaction between a software system and its associated external environment, as can be seen in Figure 2.

The way an interface implementation is specified in has no critical importance. Thus, we have provided a functional specification, but it also can be non-functional (using message sending/receiving for example). This specification must take into consideration the modification of the state of the software system and its external environment.

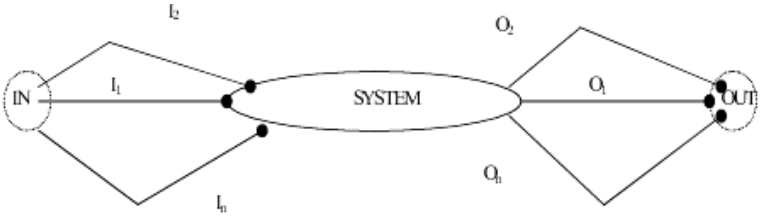


FIGURE 2. Representation of the interaction between a software system and its external software environment using interface implementation

Finally, we may synthetize the definition of a software system by using the following quadruple:

$$SYSTEM = (IN\_STATUS, OUT\_STATUS, \{FI_k, 0 \leq k \leq ni, ni \in \mathbb{N}\}, \{FO_k, 0 \leq k \leq no, no \in \mathbb{N}\})$$

where the notions involved are:

SYSTEM: the software system to be defined;
IN_STATUS: the set of the values of the software system status affected by the software services execution;
OUT_STATUS: the set of the values of the external software environment status affected by the software services execution;
FI_l: the interface implementation associated with the software service IN_l;
FO_l: the interface implementation associated with the software service OUT_l;
ni: number of the imported services;
no: number of the exported services.

This quadruple can capture the entire description of a software service. It is an open model though, because of the way an interface implementation is specified in (but is not fixed because one can choose an alternate way to specify a software

interface implementation). Any kind of specification (predicative, functional or non- functional) can be used, the only restriction is that the specification must consider the mechanism of status changing for a software system and its associated environment.

4. Software systems composition

The idea that a software system must be decomposed in smaller subsystems, for the purpose of a better handling, is an old idea and it is frequently argued in [1]. But building a software system form simpler subsystems is an idea embraced from the beginning of 90s, and the advantages of this method is presented in papers like [8, 4, 3]. We will specify a formal model, based on our software system specification, for the operation of software systems composition.

In the previous paragraphs we have provided a formal model for software systems, model based on services and interfaces. Using this model we will propose a formal model for the operation of composition of two software systems. In an informal manner, we will consider the result of the composition of two software systems S_1 and S_2 as a new software system that follow these rules:

- the group of IN services for the result system is obtained by putting together the IN services of both software systems. From this group we will eliminate all those services that are IN services for one system and OUT services for the other system;
- the group of OUT services for the result system is obtained by putting together the OUT services of both software systems. From this group we will eliminate all those services that are OUT services for one system and IN services for the other system;
- the IN_STATUS is the set of all values of the software system status which appear in all of the service descriptions from IN and OUT groups;
- the OUT_STATUS is the set of all values of the external environment status which appear in all of the service descriptions from IN and OUT groups.

For a software system S, we will consider the following functions:

- the $IN(S)$ function as the function that returns all the interface implementations associated with the imported services of this system;
- the $OUT(S)$ function as the function that returns all the interface implementations associated with the exported services of this system;
- the $SpecStatus_{IN}(spec)$ function as the function which returns all the system's status values which appear in the interface implementation from specification set *spec*;
- the $SpecStatus_{OUT}(spec)$ function as the function which returns all the external environment's status values which appear in the interface implementation from specification set *spec*.

By using the interface-based model, we can define the software systems composition by considering the set named $SYSTEMS$ as the set of all software systems.

The operation of composition, denoted by "+":

$$+ : SYSTEMS \times SYSTEMS \rightarrow SYSTEMS$$

will be defined for any software systems:

$$\begin{array}{ll} S_1 = (in\_status^1, out\_status^1, & \{(i_l^1, fi_l^1), 0 \leq l \leq ni^1, ni^1 \in \mathbb{N}\}, \\ & \{(o_l^1, fo_l^1), 0 \leq l \leq no^1, no^1 \in \mathbb{N}\}), \\ S_2 = (in\_status^2, out\_status^2, & \{(i_l^2, fi_l^2), 0 \leq l \leq ni^2, ni^2 \in \mathbb{N}\}, \\ & \{(o_l^2, fo_l^2), 0 \leq l \leq no^2, no^2 \in \mathbb{N}\}). \end{array}$$

as:

$$\begin{array}{rll} S_1 + S_2 = & (& SpecStatus((IN(S_1) \backslash S_2 \cap S_1) \cup (IN(S_2) \backslash S_1 \cap S_2)), \\ & & SpecStatus((OUT(S_1) \backslash S_1 \cap S_2) \cup (OUT(S_2) \backslash S_2 \cap S_1)), \\ & & (IN(S_1) \backslash S_2 \cap S_1) \cup (IN(S_2) \backslash S_1 \cap S_2), \\ & & (OUT(S_1) \backslash S_1 \cap S_2) \cup (OUT(S_2) \backslash S2 \cap S_1)) \end{array}$$

This formal definition of the software systems composition captures the entire meaning of the informal definition, previously presented. The expression:

$$(IN(S_1) \backslash OUT(S_2)) \cup (IN(S_2) \backslash OUT(S_1))$$

is the formal expression of the imported services, and the expression:

$$(OUT(S_1) \backslash IN(S_2)) \cup (OUT(S_2) \backslash IN(S_1))$$

is the formal expression of the exported services of the $(S1 + S2)$ information system.

The expressions:

$$\begin{array}{ll} SpecStatus_{IN} & ((IN(S_1) \backslash OUT(S_2)) \cup (IN(S_2) \backslash OUT(S_1)) \cup \\ & \cup (OUT(S_1) \backslash IN(S_2)) \cup (OUT(S_2) \backslash IN(S_1))) \\ SpecStatus_{O}UT & ((IN(S_1) \backslash OUT(S_2)) \cup (IN(S_2) \backslash OUT(S_1)) \cup \\ & \cup (OUT(S_1) \backslash IN(S_2)) \cup (OUT(S_2) \backslash IN(S_1))) \end{array}$$

defines the IN_STATUS and, respectively, OUT_STATUS attributes of the result system.

The composition operation for two software systems models the process of the tight coupling between these systems. All the similar services exported by one system and imported by the other system are hidden in the obtained system, along with the corresponding status values. One can use this operator if he wishes to obtain an expression for a tight interaction between two software systems. The composition operation is characterized by the following proprieties:

- the composition operation is commutative;
- the system $\theta = (\emptyset, \emptyset, \emptyset, \emptyset)$ is the neutral element for the composition operation;
- if we consider the software system:

$$\begin{array}{rcl} S & = & (IN\_STATUS, OUT\_STATUS, \\ & & \{I_k, 0 \leq k \leq ni, ni \in \mathbb{N}\}, \{O_k, 0 \leq k \leq no, no \in \mathbb{N}\}) \end{array}$$

then the following system:

$$CLOSE(S) = (OUT\_STATUS, IN\_STATUS, \{O_k, 0 \leq k \leq no, no \in \mathbb{N}\}, \{I_k, 0 \leq k \leq ni, ni \in \mathbb{N}\})$$

is the inverse element of S for the composition operation;

- the composition operation is not generally associative.

The proof of these proprieties, due to its extent, it is not discussed here. We have only wished to enumerate them.

The software systems specification and composition may be used for many purposes, ranging from checking of software systems compatibility to methods for software applications design and generation. CASE tools can use them as a support for software systems representation and interaction models. They might also be the basis for other different formal models in programming.

References

[1] Dahl O.J., Dijkstra E. W., Hoare C.A.R., Structured Programming, Academic Press, 1972
[2] Fucs N. E., "Specifications Are (Preferably) Executable", Software Engineering Journal, September, 1992
[3] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994
[4] Hölzle U., Integrating Independently-Developed Components in Object-Oriented Languages in LNCS 707, pp. 36–56, 1993
[5] Iuga Marin, A Graphical Representation for Software Component Systems, Faculty of mathematics and Computer Science, Research Seminars, pp. 107–110, 1999
[6] MSDN Library Visual Studio 6.0, Visual C++ Programmers guide, Serialization (Object Persistence)
[7] Allen Parrish, Component Based Software Engineering: A Broad Based Model is Needed, Brandon Dixon, David Hale in International Workshop on Component-Based Software Engineering proceedings, pp. 43–46, 1999
[8] Jan Udell, ComponentWare, Byte Magazine, pp. 46–56, 1994
[9] Wegner Peter, Models and Paradigms of Interaction, in Object-Based Distributed Programming, ECOOP'93 Workshop, Vol. 791, pp. 1–32, Springer-Verlag, 1994
[10] Szyperski Clemens, Pountain Dick, Extensible Software Systems, in BYTE May 1994, pp. 57–62, 1994

Babeş-Bolyai University, Faculty of Mathematics and Computer Science
E-mail address: marin@cs.ubbcluj.ro, iuga_marin@yahoo.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

AUTOMATIC SUPPORT FOR IMPROVING INTERACTION WITH A WEB SITE

ALINA CÂMPAN AND DARIUS BUFNEA

Abstract. In this paper we describe a method to make a Web site easier navigable by its users. In the same time, this method provides support for the Webmaster to raise the quality of the site with minimal effort. These goals are achieved by automatic creation of *orientation Web pages*. The new adds-on to the site are generated by exploiting the data accumulated in Web server access logs, being thereby a feedback to the users' "footprint". Web mining techniques are used in order to extract the meaningful information from log data.

1. Introduction

For a Web site, to be appreciated by its visitors, it is significant not only its visual aspect, the interest of the information or/and the quality of services it offers! It also counts, in a great extent, how easy the users retrieve within the site the information they are interested in. If this retrieval involves searching in long chains of unclearly linked documents, it is very likely that the user will give up searching, leave the site and possibly never come back. In the case of a company that sales its products or services on the Internet, this will mean loosing clients, clearly an undesired effect.

As defined in [7], the quality of a Web site is lower as the user's effort to find the pages that match his area of interest is growing. Most often this effort is measured as a count of links followed by the visitor until he finds the desired information. It is more realistic to assume this effort as being a trade-off between the effort to choose in every visited page the link to follow next, and the number of visited pages. We are interested in reducing this effort.

We must also note that maintaining a complex Web site can be a difficult task. The Webmaster has to face several challenges:

- The site has to contain up-to-date information;

2000 *Mathematics Subject Classification.* 68T05,68T10,68U35.

1998 *CR Categories and Descriptors.* I.5.2. [**Computing Methodologies**] : Pattern Recognition – *Design Methodology*; I.5.3. [**Computing Methodologies**] : Pattern Recognition – *Clustering*; H.3.3. [**Information Systems**] : Information Storage and Retrieval – *Information Search and Retrieval.*

- The users may seek different information at different times, and the site must be structured in a way to permit easy access, whatever the visitors' goals may be.

So, a Web site is a dynamic structure, its design may be object to changes in time. These changes materialize in new pages and links, added sometime in unlikely places.

Our purpose is to come in response to the Webmaster needs, by helping him to maintain a good quality site for the users (quality as we talked about few paragraphs above). We are entitle to sustain that the solution we shall describe does help to improve the interaction with the Web site, both for the Webmaster and, consequently, for the visitors.

Previous work. The problem of *adaptive Web sites* — "sites that automatically improve their organization and presentation by learning from visitor access patterns" — was enounced in the AI community ([5]). There are known two ways of addressing this problem. One is the *customization* of the Web site (we will not refer to this). The other, more recent, approach is the *optimization* of the site's structure to make it easier to use for all visitors. This is the trend followed by the authors in [5, 6, 7]. More precisely, they investigate the data accumulated in Web server access logs and identify a number of cohesive, possibly overlapping clusters of pages that they conclude, based on users access patterns detected in logs data, that are related to a particular topic. For some of these clusters are synthesized index pages which contain a link for every page in that cluster. The methods used in the above mentioned papers are AI traditional clustering techniques adapted to the specific of the problem.

In this paper, we proceed similarly as in [5, 6, 7]. Namely, we want to create *orientation pages* with links to the related-by-content pages of the site. But we propose a different manner to partition the site: using Web mining methods instead of AI clustering techniques. Also, this partitioning will be made with more accuracy, as we shall see.

The point to start from is raw Web server data. Taking into account a user browsing behavior model (proposed in [3]), we separate important *content page* references from references used for *navigational* purposes. Than we construct *content transactions* that correspond to the content pages visited by a user in one session. The obtained transaction repository is than mined for association rules with Web mining algorithms. Pages in every association rule give us a cluster for which we can synthesize orientation pages. Keeping in view that we try to facilitate the access to the site, and not to overhead it, we develop orientation pages only for those clusters which pages are not already linked in the site [5].

2. Problem description

We reach our goal of synthesizing orientation pages in three steps, as we said above. Each one of these steps is detailed next in one paragraph.

2.1. **Content Transactions Identification.** In order to group into transactions the elementary page references which a Web server access log contains, we consider the following user behavior model. First, we make a visit coherence assumption, which states that the pages a user visits during one visit *session* tend to be *conceptually related* [5]. Even if this is not always a valid assumption, accumulating statistics over long periods of time and for many users will reduce the noise till extinguish. Secondly, during a visit of a site, a user treats the pages either as *content* pages, either as *navigational* pages. We accept as content pages those pages with information the user is interested in. The pages where he looks for links to the desired data are considered navigational ones.

To meet our goals we shall need to identify *content transactions*, from the log data. By a content transaction we mean all of the content references a user makes in one session. Mining these content transactions will produce the associations between the content pages of the site, therefore the *clusters of pages within the site related by their content*; so, we do more than just find the most popular navigational paths and the pages these paths consist in.

We introduce now the notions we need and describe formally how to find content transactions.

Let L be the set of Web server access log entries completed with user identification information. An entry $l \in L$ includes the client IP address $l.ip$, the client user id $l.uid$, the URL of the accessed page $l.url$ and the time of access $l.time$. Local browser cache, masquerading and proxy servers can distort the accuracy of the data collected by the Web server and make user identification a difficult to accomplish task. Some solutions to user identification problem are given in [8].

We order the log entries after $l.uid$ and $l.time$ and we develop first a repository of general transactions.

Definition 1. *A* general transaction t *is a triple:*

$$t = \langle ip_t, uid_t, \{(l_1^t.url, l_1^t.time), \ldots, (l_m^t.url, l_m^t.time)\}\rangle$$

where $l_k^t \in L, l_k^t.ip = ip_t, l_k^t.uid = uid_t, k = 1, \ldots, m.$

From the general transactions, we identify the set of reference length transactions contained in the log data.

Definition 2. *A* reference length transaction t *is a triple:*

$$tr = \langle ip_{tr}, uid_{tr}, \{(l_1^{tr}.url, l_1^{tr}.time, l_1^{tr}.length), \ldots, (l_m^{tr}.url, l_m^{tr}.time, l_1^{tr}.length)\}\rangle$$

where $l_k^{tr} \in L, l_k^{tr}.ip = ip_{tr}, l_k^{tr}.uid = uid_{tr}, k = 1, \ldots, m,$
and $l_k^{tr}.lenght = l_{k+1}^{tr}.time - l_k^{tr}.time, k = 1, \ldots, m-1.$

We make some observations regarding the above definition.

Obviously, the last reference in each general transaction has no next time to use in determining the reference length. We assume that all of the last references are content ones and their length is, say, one hour (in [3], they are also excluded in the process of calculating the cut-off time). From one general transaction, we can generate one or more reference length transactions, as follows. During a user visit may appear large amounts of time between two page references. In case of such interruptions in user's navigation, longer than a threshold $TMax$ we establish, we decide to break the initial general transaction in two or more smaller reference length transactions, for which every reference (except the last one) is shorter than $TMax$. This makes sense, because resuming a visit after a long inactivity may be very well interpreted as the beginning of a new *session*. Therefore, we complete definition 2 with the following condition:

$$l_k^{tr}.length < TMax, k = 1, \ldots, m-1 \text{ and } l_m^{tr}.length \geq TMax.$$

Different users can use the same page in different manners, which are for navigational or for content purposes. We need to differentiate between these two alternatives. In most cases it is not possible to categorize a page based on its content; it is more realistic to make the distinction based on how much time the visitor spends on the page. A cut-off threshold between the medium time associated with the navigation references and the content references can be assumed or calculated — one possibility is mentioned in [3]. We denote this cut-off time by C. Having this cut-off time, we define a content transaction as follows:

Definition 3. *A* content transaction *t is a triple:*

$$tc = \langle ip_{tc}, uid_{tc}, \{(l_1^{tc}.url, l_1^{tc}.time, l_1^{tc}.length), \ldots, (l_m^{tc}.url, l_m^{tc}.time, l_m^{tc}.length)\}\rangle$$

where $l_k^{tc} \in L, l_k^{tc}.ip = ip_{tc}, l_k^{tc}.uid = uid_{tc}, k = 1, \ldots, m,$
and $C < l_k^{tc}.length < TMax, k = 1, \ldots, m-1, l_m^{tc}.length \geq TMax.$

From every reference length transaction we obtain one content transaction by removing the references shorter than the cut-off time C.

2.2. **Mining for Large Content-page Sets.** We properly format the content transactions from the repository R we obtained as described in paragraph 2.1, to be suited for the type of data mining we want to perform. Because temporal information is not needed for the mining of association rules, we exclude it from our set of transactions. We do this next.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of pages within the site. Every such p_i has a unique corresponding *url* that appears in the Web server access log entries and, consequently, in the content transactions in R, and which uniquely identifies the page within the site.

Definition 4. *We define an application f over R, which transforms a content transaction in a* mining transaction*, corresponding to the relation below:*

$$f(\langle ip, uid, \{(l_1.url, l_1.time, l_1.length), \ldots, (l_m.url, l_m.time, l_m.length)\}\rangle) = \\ = \{p_{k1}, \ldots, p_{km}\},$$

where

$$tc = \langle ip, uid, \{(l_1.url, l_1.time, l_1.length), \ldots, (l_m.url, l_m.time, l_m.length)\}\rangle \in R$$

and p_{ki} is the page that corresponds to $l_i.url$.

Each mining transaction is uniquely identified by a *tid* in the resulting set of mining transactions (we denote this set by D).

Every mining transaction (we refer to it simply as transaction from now on) tm is therefore a set of pages such that $tm \subseteq P$.

Definition 5. *Let X be a set of pages. A transaction tm is said to* contain *X if and only if $X \subseteq tm$.*

a) An association rule *is an implication of the form $X \Rightarrow Y$, where $X \subset P$, $Y \subset P$ and $X \cap Y = \emptyset$.*

b) The rule $X \Rightarrow Y$ holds in the transaction set D with confidence *c if $c\%$ of transactions in D that contain X also contain Y.*

c) The rule $X \Rightarrow Y$ has support *s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$.*

Given the set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than a user-specified minimum support (*mins*) and minimum confidence (*minc*) respectively.

This problem of mining association rules can be decomposed into two subproblems. First, find all sets of pages (page sets) that have transaction support above minimum support — which means that they are contained in a sufficient number of transactions such that the page set to have its support larger than *mins*. We call *large page sets* those page sets with minimum support condition satisfied. Once all large page sets are obtained, we use them to generate the desired rules.

There are proposed various algorithms for solving the mining association rules problem ([1, 2, 4]). For what we want to do it is sufficient to limit ourselves at finding the large page sets. We find suitable for this the algorithms described in [2].

2.3. **How We Synthesize Orientation Pages.** Since we choose to mine the mining transactions obtained from *content transactions* in D, we are entitled to say that we will obtain *large content-page sets*. What signifies, in practice, such a large content-page set? Discovering an association rule $X \Rightarrow Y$ in D means that is very frequent the situation when if a user visits the pages in X, he will also visit pages in Y. The support $X \cup Y$ of that rule gives us therefore a cluster of pages

grouped together based on certain common feature. In our case, they are grouped together corresponding to the criteria that they are related by their content. As we said, the association rules are determined from the calculated large page sets. The large content-page set from which $X \Rightarrow Y$ is derived is $X \cup Y$ (however, from $X \cup Y$ it is possible to obtain more that one association rule!). The discussion above justifies why we found sufficient to determine the large page sets, and not to continue with identifying the association rules.

For every set of currently unlinked content-related pages we want to generate an orientation page, comprising one link for every page in that set. Two pages are considered linked if there exists a link from one to the other, or if there exists a page that links to both of them. It certainly wouldn't make sense to include, in an orientation page, links to two pages that are already pointed by a common parent, because we would create a redundant structure equivalent with the existing parent.

We make use in constructing the orientation pages of the following obvious property that stands for large page sets. In fact, the algorithms that discover the large page sets in a transaction repository are based on this property. We enunciate it and then we introduce another concept we will use.

Remark 1. *Any subset of a large page set is also a large page set.*

Definition 6. *We call a* maximal page set *a large page set that is not contained in any other large page set.*

We explained before that every large content-page set comprises pages related by their content, and which are often visited together. Obviously, we wouldn't have any advantage in generating an orientation page for every large page set! This because every large page set that is not maximal will retrieve itself in a series of other large and maximal page sets fact that would cause many redundant orientation pages! So, we are interested in knowing only the maximal content-page sets because they give us the maximal, complete clusters of pages related by their content. To obtain them from the large content-page sets that we have previously determined is a straightforward task.

However, we are not yet at the end of our task. As we affirmed, we want to generate orientation pages for groups of related-by-content pages that are not already linked in the site So, it remains to detect, from every maximal content-page set, the subsets of pages that, two by two, are currently unlinked. We describe bellow, in an algorithmic form, a method to do this by working on a graph model.

Algorithm *ConnComp* is
Set $H = \emptyset$;
For every maximal content-page set previously determined, M=$\{p_{i1}, \ldots, p_{ij}\} \subset P$, Do
    Associate to M a graph $G = (M, U)$; $U \subseteq M \times M$ where there is an edge $(p_i, p_j) \in U$ **iff** the pages p_i and p_j are unlinked (meaning that they are not linked in the sense we specified above) in our site;

Find all the connected components of G and add them to H;
End For;

End *ConnComp*

The algorithm *ConnComp* supplies us the set H of all the connected components determined for all maximal content-page sets. For every connected component in H there are two properties that follow from the way we defined G:

- All the pages in a connected component in H are content related;
- Every connected component in H has the property that each two of its pages are not linked.

We must note that it is not realistic to offer to the visitors of the site orientation pages with hundreds of links, because this wouldn't be of any help. Similarly, orientation pages with just a few links should be excluded, as not being significant enough and only burdening the site's structure. So, we agree to reasonably choose two thresholds (min and Max, $min < Max$) to limit the number of links in an acceptable orientation page.

Eventually, we generate one or more orientation pages for every connected component in H like this:

- If the connected component has between min and Max pages, we generate one orientation page containing a link for every page in the component;
- If the connected component has more than Max pages we break it into parts of Max pages each (excepting the last one, which may have between 1 and Max pages). For each part we construct one orientation page and we create a parent index to point to the orientation pages corresponding to all these parts. So, we have a two-level orientation structure. We will not take into consideration the connected components with more than Max^2 pages; we think that such cases have little chance to appear in practice for a common site. For $Max = 10$, imagine what it means "hot" access pattern that imply more than 100 pages!

We point out that every possible improvement to the site (add-on orientation pages) is reported to the Webmaster to be accepted or not. Only the content of pages is automatically supplied; the Webmaster will have to integrate the orientation pages in the overall design of the site and place them where he thinks adequate. So this approach brings in only non-destructive transformations: changes of the site that leave existing structure intact.

3. Heuristic Comparative Study

In [5] grouping the pages above their common topic simply consisted in finding collections of pages that tend to co-occur in visits. This process didn't make difference between pages that were visited only for navigational purposes and those

pages that really interested the user by their content. This fact can erroneously introduce some pages used for navigation, into one cluster of related-by-topic pages, only because they are placed on a frequently used path between two content related pages.

In turn, we identified the pages that *really* share a particular topic by making use of the concept of content transaction [3]. We explain now why this is true.

In the context of classifying a reference made by a user as either a navigational or a content one, there are two ways of defining transactions *3*. One is to define a transaction as all of the navigation references up to and including each content reference for a user session — *navigation-content transactions*. The other is to compound a transaction as we did in this paper, from all of the content references within a user session — *content transactions*. Mining the repository of navigation-content transactions calculated from a log and the one of content transaction determined for the same log will take us to different results. The first approach is, in a way, similar to what is described in [5] — mining navigation-content transaction would essentially give the common traversal paths trough the Web site towards content pages. Mining the content transactions produces, in turn, associations between the content pages of a site, without any information about the path followed between the pages. We point out other significant fact that devolves from our approach: Web mining on content transactions does not produce association rules that might be erroneously determined if we would consider all page references in a log. Imagine this situation: users that treat page A as a navigation page do generally go on to the page B, but users that visit A as a content page do not go on to B. In this case, including navigational references into the data mining process will classify A in the same cluster as B. Mining content transactions will not produce this fake association, because rule $A \Rightarrow B$ will not have minimum support. So, in this way, we grouped the pages related by their content with more accuracy than previously has been done.

4. Conclusions

In this paper we presented an approach to the problem of adaptive Web sites — synthesizing new pages to be added to the site in order to facilitate the retrieval of information was already proposed before. What is new is the way we establish the content of the orientation pages. We identified the clusters of pages that really share a particular topic by making use of the concept of content transaction. We outlined above the advantage of this technique.

References

[1] Agrawal R., Srikant R., *Fast Algorithms for Mining Association Rules*, In Proc. of the 20th VLDB Conference, pp. 487–499, Santiago, Chile, 1994 (http://citeseer.nj.nec.com/agrawal94fast.html).

[2] Chen M.-S., Park J. S., Yu P.S., *Data Mining for Path Traversal Patterns in a Web Environment*, In Proc. of the 16th International Conference on Distributed Computing Systems, pp. 385–392, 1996 (http://citeseer.nj.nec.com/128354.html).

[3] Cooley R., Mobasher B., Srivastava J., *Grouping Web Page References into Transactions for Mining World Wide Web Browsing Patterns*, In Knowledge and Data Engineering Workshop, pp. 2–9, Newport Beach, CA, 1997 (http://citeseer.nj.nec.com/cooley97grouping.html).

[4] Park J. S., Chen M.-S., Yu P.S., *An Effective Hash-Based Algorithm for Mining Association Rules*, In Proc. 1995 ACM-SIGMOD, pp. 175–186, 1995 (http://citeseer.nj.nec.com/park95effective.html).

[5] Perkowitz M., Etzioni O., *Adaptive Web sites: Automatically Synthesizing Web Pages*, In 15- th National Conference on Artificial Inteligence, 1998 (http://citeseer.nj.nec.com/perkowitz98adaptive.html).

[6] Perkowitz M., Etzioni O., *Towards adaptive Web sites: Conceptual framework and case study*, Artificial Intelligence 118 (2000), pp. 245–275, 2000 (http://citeseer.nj.nec.com/326006.html).

[7] Perkowitz M., *Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis*, Ph.D. Dissertation, 2001 (http://www.perkowitz.net/mike/research/papers/phd.pdf).

[8] Pitkow J., *In search of reliable usage data on the WWW*, In Proc of the Sixth International World Wide Web Conference, pp. 451–463, Santa Clara, CA, 1997 (http://citeseer.nj.nec.com/242362.html).

Department of Computer Science, Faculty of Mathematics and Computer Science, "Babeş-Bolyai University, 1, M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania
E-mail address: alina@cs.ubbcluj.ro

Comunications Center, "Babeş-Bolyai University, 1, M. Kogălniceanu St., RO- 3400 Cluj-Napoca, Romania
E-mail address: bufny@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

SPATIAL DATA CAPTURE IN GIS ENVIRONMENT

A. M. IMBROANE

Abstract. Traditionally geographical data is presented on maps using symbols, lines and colours. A map is an effective medium for presentation and a database for storing geographical data. But herein lies some limitations. The stored information is processed and presented in a particular way and for a particular purpose. A map provides a fixed, static picture of geography that is almost always a compromise between many different users. Compared to maps GIS (Geographical Information System) has the inherent advantage that data storage and data presentation are separate and may be presented and viewed in various ways. The core of GIS environment is the matching of spatial data (digital maps) and attribute data (the meaningful of spatial data) together. The attribute data are in fact tables associated with geographical features stored in spatial database. The main problem is capturing and storing spatial data in digital form. In this paper we will approach just spatial data capture in vector format and not the database organization.

1. Introduction

The geographic entities or objects in GIS are based on two different types of data: spatial and attribute. Spatial data representation in GIS are classified into raster and vector, which are dual with regard to space bounding and space filling [6]. Parallel to representational duality, there is the duality of spatial concepts, namely entity-based and field-based concepts [4]. The fundamental difference between the representations as well as between spatial concepts causes problems in interoperability and multi-source fusion [8]. Attributes or descriptive, or aspatial data are alphanumeric data related to the graphic entities. They are also called thematic data because they contain themselves a theme of the graphic features. Attributes of vector units are stored in computer files as records or tuples that may be linked to them by pointers. Usually the attributes are stored in traditional relational database. This operation is called *geocoding* or *address matching*. The fact is that attributes data can be linked with other tabular data (external) and so the new data can be processed together with the others. This is one of the powerful specific operations in GIS. Spatial data and attribute data can be stored

2000 *Mathematics Subject Classification.* 91B99.

1998 *CR Categories and Descriptors.* J.2. [**Computer Applications**] : Physical Sciences and Engineering – *Earth and Atmospheric Sciences.*

separately or together depending of the database design performed by the GIS proprietary. A geographical database is a set of geometric entities (spatial features) and attributes. The major problems are to capture, store, manipulate, maintain and process the spatial data. In order to obtain a digital map which can be processed by GIS software it has to be depicted in separated thematic layers, like: topography, geology, hydrology, and so on. The next step is to separate points, lines and polygons associated with real entities from the layers above. And finally, each graphic feature, points, lines, polygons have to contain just a single theme, like towns, roads, administrative boundaries.

In this paper we will approach only vector format and related attributes. The vector format can be represented in two components: *geometric (graphic)* data and *topological* data. Geometric data have a quantitative nature and are used to represent coordinates, lines, areas etc. The two-dimensional vector format has three subtypes: *point*, *line* and *polygon*, named *features*. These features are called also 'graphics primitives' (definition borough from CAD/CAM software). Geometric, or non-topological data models are those in which any positional information are recorded. The recognition of individual spatial units as separate unconnected elements characterized the early data organization for digital cartography. Topological data describe the relationship between the geometric data. There are several types of topological relationship: *connectivity*, *adjacency*, and *inclusion*. Examples of queries concerning topological relationship are: which areas are neighbors of each other (adjacency), which lines are connected and form a network or a road (connectivity), and which lakes lie in a certain country (inclusion). Topological data are not always stored explicitly, because in principle they can be derived from geometric data. Note that this definition of topological data is slightly different from the stricter one used in mathematics.

To avoid the confusions and for simplicity it is better that every type feature represents only one type of an object in a layer. Example: in a point theme we have only towns and not other kind of point object. In a line theme we have only rivers and not roads.

2. Data capture programs in pseudo-code form for point feature

The core of the vector data storage is the capturing data from digitizer tablet. This procedure will be used for point feature storage, line feature storage, and polygon feature storage. We have different programs in pseudo-code for the three spatial features: point, line and polygon.

```
Program POINT                          {Captures and stores x,y coordinates}
open VECTOR_POINT
repeat
  read cod                               {cod =1 write in file, cod =-1 end}
  call DIGITIZE(x,y,id$)
  put id$,x,y
```

```
until cod =-1
close VECTOR_POINT
end
```

Subsequently we will illustrate the transfer of a string of data in ASCII format from a tablet and extract the (x,y) coordinate. The procedure DIGITIZE calls the port repeatedly looking for the data, assembles the (x,y) coordinates.

```
procedure DIGITIZE(x,y,id$)                                {Get x, y and id$}
call GET_STRING(str$)
x=num(val(str$,1,4))
y=num(val(str$,5,4))
id$= num(val(str$,9,4)
endprocedure

procedure GET_STRING(str$)              {Extracts characters from the port}
str$="":port_signal=0
repeat
  call PORT(port_signal)
  call INKEY(key$)
  if key$ <> " " then
    char$=GET$
    str$=str$+char$
  endif
until (char$=chr$(10)) or (key = " ")
endprocedure
```

Val$ function extracts x, y values and the identifier from a specific part of string, **Num** function converts from string to numeric value. So the x coordinate is obtained at the numerical value of part of str$ string, starting at the character 1 and ending at character 4; y is starting at character 5 and ending at 8. The identifier feature (id$) is extracted from character position 9 and end at 12. The program may be used for control points and point theme as well.

The next step is to create attribute table linked together with VECTOR_POINT which contain descriptive data on point features. This mean to open VECTOR_POINT and read every identifier code and create for it a field (character or numeric). In subsequent we present the program in general form.

```
Program ATTRIBUTE_POINT                          {Creates the attribute table}
open VECTOR_POINT                          {assigned to point feature table}
open ATTRIB_POINT
repeat
  get id$,x,y
  repeat
    put id$
    read file_type                  {1 for char 0 for numeric -1 for finish}
    if file_type=1 then read chr_field$
    put chr_field$
```

```
    if file_type=0 then read num_field
    put num_field
  until file_type=-1
until endfile
close VECTOR_POINT
close ATTRIB_POINT
end
```

Usually just a few fields are generated in the attribute data file. More data can be joined from an external relational database. The join condition is that every file has a common field.

Examples. Assume that we must create a point theme, which is representing towns at sufficient small scale that the shape of the town is not important. In the VECTOR_POINT file we have the coordinates of every town and in ATTRIBUTE_POINT we can have the subsequent fields: town code (which must be unique), name, population and so on.

3. Data capture programs for line feature

For the line feature storage it is convenient to use multi-button cursor. The functionality of the buttons is pre-specified. Here we define one button (say button 1) to denote a digitized point, another (button 2) to mean 'start node – start of arc', and the other (button 3) 'end node end of arc'. The following code shows how a vector of points can be build up by correctly interpreting incoming data from the tablet using this principle. Such code normally is treated as a procedure or a sub-routine within the program controlling the data generation process.

```
Program LINE                                          {Captures line feature}
open VECTOR_LINE
repeat
  read cod                              {cod=1 write in file; cod=-1 for end}
  k=0
  repeat
    call GET_STRING (str$)
    k=k+1
    con$=val(str$,9,1)
    x=num(val(str$,1,4))
    y=num(val(str$,5,4))
    if con$="button2" then     begin
      line(k,1)=x: line(k,2)=y end
    if con$="button3" then     begin
      line(k,3)=x: line(k,4)=y end
    if con$="button1" then begin
      line(k,3)=x: line(k,4)=y
      line(k+1,1)=x: line(k+1,2)=y
    endif
```

```
  until con$="button3"
  read id$
  put id$
  kk=k-1                                    {kk = total number of segments}
  put line(1,1),line(1,2),line(1,3),line(1,4),kk
  for k=1 to kk
    put line(k+1,1), line(k+1,2)
  endfor
until cod=-1
close VECTOR_LINE
end
```

The attribute file contains description data on line feature. Usually line attribute contains the arc length, beside others. This characteristic is required in almost all spatial procedures analysis. The algorithms that follow use the next functions two compute the length of a segment and area of a triangle:

```
length(x1,y1,x2,y2) = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))
area(x1,y1,x2,y2,x3,y3) = (x1*y2+x2*y3+x3*y1-y1*x2-y2*x3-y3*x1)/2
```

An algorithm for the attribute file is given below.

```
Program  ATTRIBUTE_LINE          {Creates attribute table for line feature}
open VECTOR_LINE
open ATTRIB_ LINE
repeat
  get id$,xs,ys,xe,ye,n
  len =length(xs,ys,x(1),y(1))
  for i=1 to n-1
    len = len + length(x(i),y(i),x(i+1),y(i+1))
  endfor
  len = len + length(x(n),y(n),xe,ye)
  put id$
  put len
  repeat
    read file_type               {1 for char, 0 for numeric, -1 for finish}
    if file_type=1 then read chr_field$
    put chr_field$
    if file_type=0 then read num_field
    put num_field
  until file_type=-1
until endfile
close VECTOR_LINE
close ATTRIB_ LINE
end
```

Examples. If LINE_VECTOR refers to roads, the attribute data must be code of the road, name, length, quality and so on. If LINE_VECTOR refers to rivers,

attribute data must be code of the river, name, length, quality, mean debit and so on.

4. Data capture programs for polygon feature

The polygon theme is in a way synonym with the line theme. In fact a polygon is a closed line (start point and end point are the same). When the line is finished placing the cursor in the right location (exactly in the same location where the start point is) might be problematic. This is a real problem because we are not able to mark exactly the same point at different times (if we move a little bit the cursor). It is an "error" problem. So it is necessary to establish an error inside which the two points represent the same point. It is often named *snap node error* (or *snap node tolerance*). This is linked with the digitizer precision. Anyway the error is establish sufficiently small for the purpose of using the map. Operator specifies this error before the digitizing process is running. If the two nodes are inside error tolerance, the start node 'snap' the end node and therefore we have just one node (xe=xs, ye=ys).

Every polygon must contain an isolated point inside the polygon (not necessarily in centre) named centroid. Like for the line feature storage we use a multi-button cursor.

```
Program POLYGON                        {Captures and stores polygon features}
open VECTOR_POLY
read snap_point
repeat
  read cod              {if cod=1 then capture polygon; if cod=-1 then end}
10 k=0
  repeat
    call GET_STRING (str$)
    k=k+1
    con$=val(str$,9,1)
    x=num(val(str$,1,4))
    y=num(val(str$,5,4))
    if con$="button2" then begin
      line(k,1)=x: line(k,2)=y
    end else if con$="button3" then begin
      line(k,3)=x: line(k,4)=y
    end else if con$="button1" then begin
      line(k,3)=x: line(k,y)=y
      line(k+1,1)=x: line(k+1,2)=y
    end
  until con$="button3"
  xs=line(k,1): ys=line(k,2): xe=line(k,3): ye=line(k,4)
  dist = length(xs,ys,xe,ye)
  if dist>snap_point then goto 10
```

```
  kk=k-1                                        {kk = total number of segments}
  read id$
  put  id$
  put xs,ys, kk
  call DIGITIZE (x,y,id$)
  put x,y
  for k=1 to kk
    put line(k+1,1), line(k+1,2)
  endfor
until cod=-1
close VECTOR_POLY
end
```

A multi-polygon map often requires a hierarchical data structure because polygons will share common boundary and line segments will terminate at the common nodes. Moreover, to produce cartographic output, any combination of polygons may be required.

Usually the attribute table contains beside identifier, both perimeter and area of the polygon, followed by an arbitrary number of fields. For the perimeter we can use the same procedure as for arc length. The measurement of an irregular feature such a polygon can be done by calculating the areas of the trapezoids under the successive line segments which make up the polygon [1]. Another method for finding area of a polygon is to decompose the polygon in triangles using the centroid coordinate, and finally calculate area of each triangle.

For an effective evaluation of the area we depict the polygon in three parts: first triangle, intermediate triangles and last triangle. A pseudo-code program for attribute file is:

```
Program  ATTRIBUTE_POLY                          {Creates the attribute table}
open VECTOR_ POLY                               {associated with polygon feature}
open ATTRIB_ POLY
repeat
  get id$,xs,ys,n
  get xc,yc
  len = length(xs,ys,x(1),y(1))
  a = area(xs,ys,x(1),y(1),xc,yc)
  for i=1 to n-1
    len = len + length(x(i),y(i),x(i+1),y(i+1))
    a = a + area(x(i),y(i),x(i+1),y(i+1),xc,yc)
  endfor
  len = len + length(x(n),y(n),xs,ys)
  a = a + area(xs,ys,x(n),y(n),xc,yc)
  put id$,len,area
  repeat
    read file_type                {1 for char, 0 for numeric, -1 for finish}
    if file_type=1 then read chr_field$
```

```
    put chr_field $
    if file_type=0 then read num_field
    put num_field
  until file_type=-1
until endfile
close VECTOR_ POLY
close ATTRIB_ POLY
end
```

Data entry process, such as retrieval of thematic data from secondary sources, or topographic data capture from digitizing operations, are typically managed within a GIS. The task is to provide the user with techniques for interfacing with input device and file handling procedures.

References

[1] Bracken I., Webster C., *Information Technology in Geography and Planning*, Routledge, London, 1990.

[2] Buttenfield B.P., *Digital Definitions of Scale Dependent Line Structures*, in Proceeding Auto Carto M.J. Blakemore (ed), vol. I, 1986, 497–506.

[3] Foley J. D. and Van Dam A., *Fundamentals of Computer Graphics*, Addison-Wesley, Reading, Mass, 1982.

[4] Frank A.U., *Spatial Concepts, Geometric Data Models and Geometric Data Structures*, Computers & Geoscience, 18, 1992, 409-417.

[5] Imbroane, A.M., Moore D., *Iniţiere în GIS şi Teledetecţie*, Presa Univ. Clujeană, Cluj-Napoca, 1999.

[6] Peuquet D.J., *A Conceptual Framework and Comparison of Spatial Data Models*, Cartographica, 66, 1984, 113–121.

[7] Wallace V.L., *The Semantics of Graphics Input Devices*, Computer Graphics, 10, 1976, 1, 61–65.

[8] Winter S., *Bridging Vector and Raster Representation in GIS*, ACM, 11, 1998, 57–63.

"Babeş-Bolyai" University, Faculty of Geography
E-mail address: alex@geografie.ubbcluj.ro