UNIVERSITY OF CRAIOVA
Faculty of Mathematics and Computer Science
RESEARCH CENTER FOR ARTIFICIAL INTELLIGENCE

501

Student Notes in Computer Science

Edited by RCAI Junior Members

Volume 1, Issue 1
Electronic version

December 2003
Craiova, ROMANIA

SUPERVISORS:

Professor Nicolae Țăndăreanu, Ph.D.
Assistant Prof. Ruxandra Gorunescu
Assistant Prof. Cătălin Stoean

EDITORIAL BOARD

Mihai Dobrin
RCAI Junior Member
Third Year Student
E-mail: famdobrin@rdslink.ro

Orlando Gabriel Pîrlog
RCAI Junior Member
Third Year Student
E-mail: orlando@k.ro

Andrei-Constantin Sandu
RCAI Junior Member
Third Year Student
E-mail: andreisandu2000@yahoo.com

Gabriel Voinescu
RCAI Junior Member
Third Year Student
E-mail: gabbryel@k.ro

Contents

A graphical approach to Lindenmayer systems

ABSTRACT

This application belongs to the category of Lindenmayer system representation applications. The Lindenmayer systems are used in the computer game industry to draw certain textures and 3D elements (flowers, trees, etc.) being useful in the drawings which repeat at a smaller scale elements presented at a bigger scale of that figure.

1. Introduction

The central concept of a Lindenmayer system is that of rewriting. Rewriting is a technique for building complex objects by successively replacing parts of simple initial ones using a set of rewriting rules or productions.

The idea of rewriting is not new, it was firstly introduced by Chomsky for the generating of strings.

In 1905 Koch proposed the same mechanism for generating curves.

2. Fundamental theoretical concepts of a Lindenmayer system

Let V be an alphabet and $V^* = V^+ \cup \{\lambda\}$ the set of all strings (words) over V, where λ is the zero-lengthed word. Any subset of V^* is called a language.

Definition 1.1 : An OL system is a tidy triplet $G = (V, \omega, P)$ where:

- V is an alphabet
- $\omega \in V^+$ and this symbol is called the axiom of the system
- $P \subset V \times V^k$ is a finite set of elements which are called productions
- for each $a \in V$ there is at least one word χ for which $(a, \chi) \in P$ [1].

A production (a, χ) is denoted by: $a \to \chi$ since this represents the replacing of *a* with the symbol χ.

The last condition from definition 1.1 is not restrictive. If for a symbol $a \in V$ there is not any production $(a, \chi) \in P$ then the production (a, a) for *P* can be added.

Definition 1.2: An OL-system is deterministic (or DOL-system) if for each $a \in V$ there is one and only one word $a \in V$ for which $a \to \chi \in P$.[2]

Definition 1.3: Let $\mu = a_1 ... a_k \in V^*$ be an arbitrary word over *V*. It is said that the word $\mu = a_1 ... a_k \in V^*$ is directly *derivative* from μ and it is denoted by $\mu \Rightarrow \chi_1 ... \chi_k \in V^*$ if and only if $a_i \to \chi_i \in P$ for each $i \in \{1, ..., k\}$. The reflexive and transitive closing of the "$\Rightarrow$" relation is "$\Rightarrow^*$" [3]; e.g. the development of the Anabaena Catenula filament can be described by a DOL system. The cytologyc states of a corpuscle are denoted by *u* and *v*. Using *l* and *r* to indicate the polarity of a corpuscle (left and right respectively), the development of the microorganism can be written by the DOL system below:

$\omega : u_r$

$p_1 : u_r \to u_l v_r$

$p_2 : u_r \to v_l u_r$

$p_3 : v_r \to u_r$

$p_4 : v_l \to u_l$

Starting with the ω axiom the following are obtained:

u_r

$u_l v_r$

$v_l u_r u_r$

$v_l u_l v_r u_l v_r$

……

Definition 1.4: A word *v* is generated by the *G* OL-system, $G = (V, \omega, P)$, if there is derivation of the *v* word from $\omega, \omega \Rightarrow^* v$ in *G*.

The language generated by *G* is: $L(G) = \{v / \omega \Rightarrow^{*} v\}$. [4]

A word *v* is directly derivative from μ using a process chraracterized by a parallel replacing. This means that all the symbols from μ are replaced simultaneonsly with some words, which are defined using some productions.

The Lindenmayer systems have many implications in the theory of formal languages in the moulding of the development of some microorganisms.

Generating drawings using a Lindenmayer system

The "turtle" interpretation of a word is introduced. First of all, a turtle state is a triplet (x, y, α), where *(x,y)* are the cartezian coordinates of a point and α is the measure of the defined angle. The (x, y, α) triplet is interpreted as follows: one can imagine a virtual pencil (or a turtle) which can draw a trace; *(x,y)* are the coordinates of the point where the virtual pencil is placed, and the angle α is the direction straight forward.

d denotes the length of a segment and δ a value which is used as a value of incrementation for the angle that defines the direction forward of the virtual pencil.

The basic controls for the virtual pencil are symbolized like this:

F: move and draw forward with a step of length d. Thus, if the position of the virtual pencil is *(x,y)* then the new state is (x', y', α), where $x' = x + d\cos\alpha$ and $y' = y + d\sin\alpha$. A segment of line from the point *(x,y)* to the point *(x',y')* will be drawn.

f: move forward with a step of length *d*, without drawing anything

+: rotation to the left with the δ angle; if the actual turtle state is (x, y, α), then the new turtle state will be $(x', y', \alpha + \delta)$.

-: rotation to the right with the δ angle; if the actual turtle state is (x, y, α), then the new turtle state will be $(x', y', \alpha - \delta)$.

[: remember the current position

]: return to the last remembered position

A snapshot of commands for the virtual pencil is a word $v \in \{F, f, +, -\}^*$. Let it be supposed that the initial turtle state is (x_0, y_0, α_0). The angle α_0 defines the direction forward. If an angle δ and a word over the alphabet $\{F, f, +, -\}$ are taken then the road drawn by the virtual pencil is established.

A Lindenmayer system where $v \in \{F, f, +, -\}^*$ and each production is either $F \to v$ or $f \to v$ with $v \in V^+$ is considered.

A segment of line *F* with the length *d* is transformed into broken line using a procedure which is described below.

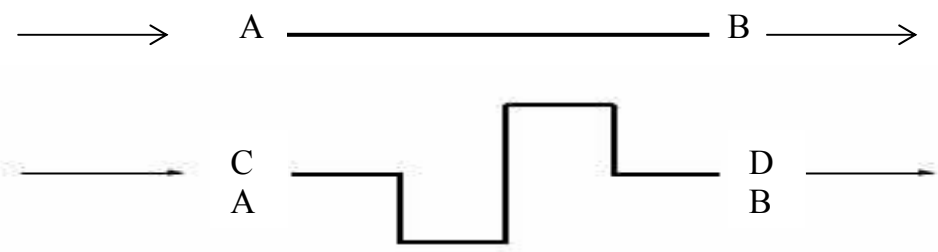


Figure 1: The decodification of snapshot: F-F+F+FF-F-F+F in a broken line

Let A and B be the extremities of the segment. A production $p \in P$ is chosen i.d. $p : F \to v$, where $v \in V^*$. Interpreting the snapshot of commands *v* with a step d_1 a broken line with the extremities C and D is obtained. d_1 is calculated for the distance which has the extremities A and B and is replaced with the broken line obtained after recalculating the next step. For example consider the production:

$p : F \to F - F + F + FF - F - F + F$.

Decoding the snapshot of commands $F \to F - F + F + FF - F - F + F$ using a step d_2 and $\delta = 90$ the line drawn with C and D as its extremities in Figure 1 is obtained. Taking $d_2 = d / 4$, where *d* is the length of the segment F denoted as AB, if the extremity coverlaps A then the extremity D overlaps B. This is a one step transformation so far. The same transformation is used for each segment of the polygonal line obtained

after the first step. Then this transformation for each segment of the polygonal line obtained next is repeated.

Now, interpreting the axiom as a polygonal line, the initiator is obtained.

For each segment of the the above discussed initiator transformation is used, then the procedure for each polygonal line obtained is repeated. This procedure can be repeated as many times as desired and the method obtained is called *Koch's Construction*.

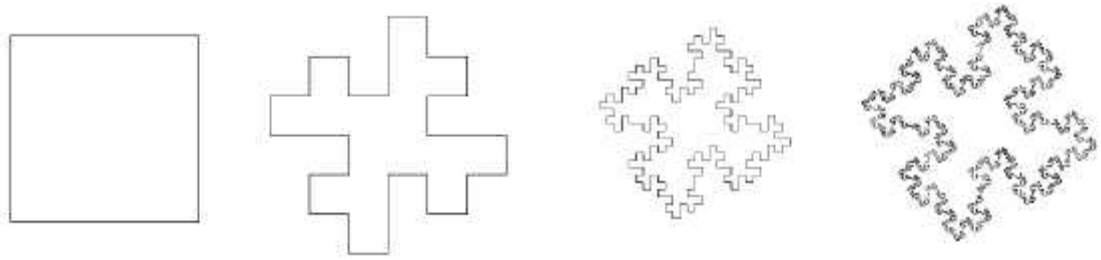


Figure 2: Koch's Construction

For example, take into consideration the Lindenmayer system:

$$\omega : F - F - F - F$$

$$p : F \to F - F + F + FF - F - F + F$$

and let $\delta = 90$. The initiator is a square. Using Koch's Construction, the drawing in Figure 2 is obtained.

3. Implementation remarks

The application that was developed uses Java. To generate the drawing for a step bigger than 0, recursivity is used. Any expression which must be generated for "*n*" steps is reduced to "*n-1*" steps. Then, the expression becomes "*n-2*" reduced and so on until it reaches step 0.

To represent the entire drawing, a scaling of it must be made. Before drawing, the maximum real value of the X coordinate, the maximum real value of the Y coordinate, the minimum real value of the X coordinate, the minimum real value of the Y coordinate

have to be calculated. The window where it will be drawn which is between 200 and 500 is chosen. This is how the scalar factor will be found. Each point which will be shown on the screen will be multiplied with this factor, and after word if will be rounded to the closest integer and will be shown.

The application has 3 options: Turtle, Tree (uses "[" and "]" in writing) and Combination (shows the expression of ω , starting from step 0 to the specified step). Firstly the expression of F is introduced, then the expression of ω (Start) then the angle and the step and finally one of the options is chosen. If Tree is chosen, ω (start) can deprive because only the expression of F will it be interpreted.

4. The algorithm

The input data is made of:

ω : the initial figure;

F: the transformation; (for the AB segment, F must reach B)

δ : the winding angle;

The algorithm contains 5 steps:

Step 1. Calculate the relation between the segment in step n and the one in step n + 1

Step 2. Determine the real display window

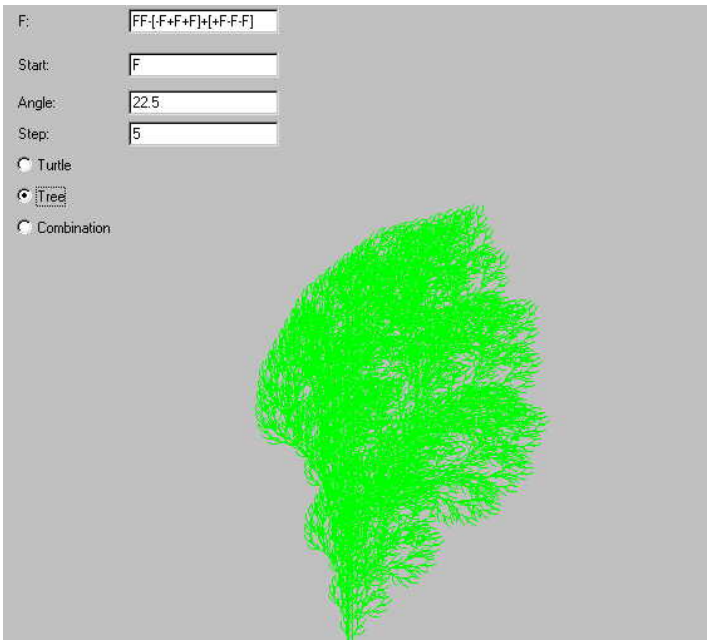
Step 3. Determine the screen display window

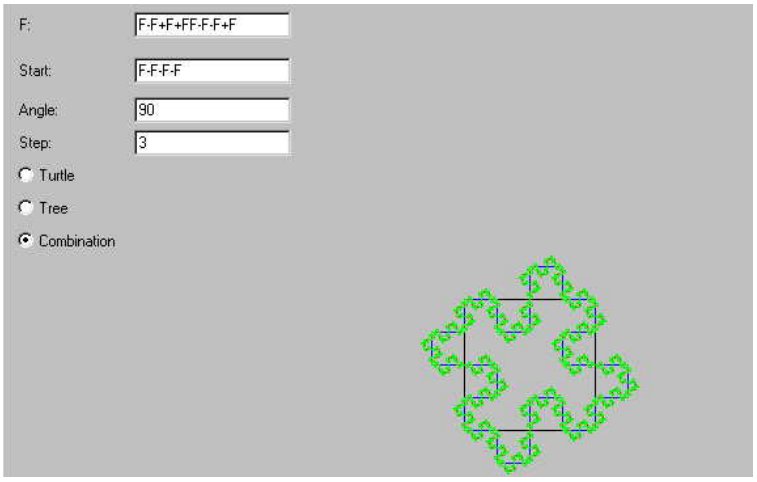
Step 4. Determine the scaling factor from real space into screen space

Step 5. The effective drawing of the picture

5. Experimental results:

Results are shown below:





6. Conclusions and future work:

This application is not the fastest, but it can be optimized by pre-calculating the values of the sinuses and co sinuses. Further work can be added in order to extend this program to represent not only the 2D Lindenmayer systems but also the 3D Lindenmayer systems.

7. Acknowledgements

We would like to thank Professor Nicolae Tandareanu, Professor Ruxandra Gorunescu and Professor Catalin Stoean who were very close to us and helped us through the making of this article. Without their help this article wouldn't have existed.

8. References

[1] Csima Judit Lindenmayer systems http://sziami.cs.bme.hu/~csima/phd1/node13.html

[2] Arto Salomaa: Developmental Models for Artificial Life: Basics of L Systems, in Artificial Life-Grammatical Models, Ed. Gh. Paun, Black Sea University Press, 1995, p.22-32

[3] Jurgen Dassow: Cooperating Grammar Systems (Definitions, Basic Results, Open Problems), in Artificial Life-Grammatical Models, Ed. Gh. Paun, Black Sea University Press, 1995, p.40-52

[4] Gheorghe Paun: New Variants of Lindenmayer Systems with Biological Motivation, in Artificial Life-Grammatical Models, Ed. Gh. Paun, Black Sea University Press, 1995, p.129-136

Dobrin Mihai, student in the 3rd year at the Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova. Recently, I was appointed junior member at the Research Center for Artificial Intelligence (RCAI). My hobbies are: computer (especially programming), beekeeping, classical music. At

this moment I am working on a program for topography at a firm in Craiova. I spend my spare time as a scout at the Rabon-Cfr group or… between the bees…beekeeping them!…

Sandu Andrei-Constantin was born in Craiova in 1982. In the present, he is a student at the Faculty of Mathematics and Computer Science at the University of Craiova and he is a junior member in a group of students from the same university who dedicate their time to finding new ways of improving the knowledge they acquired from their teachers, by participating to local, national and international contests where he himself obtained good results. He also studied in Spain in 2003, at the Autonoma University of Madrid .

Admission decisions for candidates to a competition through Minsky frames

Abstract

In this paper some theoretical concepts of Minsky frames are presented and a practical illustration of these concepts is intended. The application could be used as a means of deciding acceptance of candidates at a competition.

1. Introduction

Proposed paper presents some of Minsky's concepts and shows a practical implementation of this theory in a decision-making process regarding acceptance of candidates at a competition.

Marvin Minsky is one of the inventors of the field of Artificial Intelligence. In fact, some call him the "father of Artificial Intelligence". He has contributed many important ideas and concepts to the field of AI. He was one of the first people to explore the area and he still remains as one of the leading researchers today.

In the 1970's he invented a form of knowledge representation known as "frames". These "frames" were a lot like forms that could be filled in with data. He could represent knowledge in many areas including language understanding and visual perception. These were not only useful to Artificial Intelligence, but they are considered to be some of the earliest forms of object oriented programming. But Minsky's largest contribution was yet to come.

What are frames? What problems are frames supposed to solve? How?

Minsky tackles the problem of knowledge representation by creating a model of how we represent and store knowledge([2],[3]). It states that we organize and store our knowledge of the world symbolically through frame representation. A frame is a set of terminals containing characteristics of specific information. A system identifies information based upon the characteristics of the information held in the frame. Frames are supposed to solve the mystery of the organizational structure of knowledge in humans and other cognitive machines by solving problems such as efficiency, adaptability and sufficiency. Frames are sufficient to hold enough characteristics to identify the desired knowledge, they are highly organized and therefore efficient, and they are also adaptable because the information held in the frames is able to be manipulated based upon knew knowledge.

2. Fundamental theoretical concepts of Minsky frames

2.1 The intuitional aspect of the concepts

In this paragraph it is considered that a frame $\boldsymbol{F}$ is characterized by the following elements([1]):

- a symbolic name which represents the *name of the frame;* each frame has a unique name; the name of the frame is used for its identification.

- a finite set of symbolic names; each name represents *the parent* of the frame; a frame can have zero or more parents.

- a finite set of *slots;* a slot is an orderly couple of (*attribute, value*), where *attribute* is the name of the frame's property and *value* is the adequate value hereof.

A frame is represented by a diagrammatic drawing as in Figure 1, where $attribute_i \neq attribute_j$ for $i \neq j$. The number of parents and the number of slots can differ from frame to frame. Presume that the value V of attribute A of frame F is wanted. The following cases are possible:

- There exists a slot S formally denoted by (A,V) or (A,P), where V is the direct value of the attribute and P is the name of a procedure; in the first case V is the value of attribute A; in the second case, the value of the attribute is a value returned by the procedure P.
- There is no slot of F so that A be the first component of the slot; in this case, what is tried is to find the value of the attribute with the help of the respective object parents, i.e. the value of the attribute A is hereditary; if none of the parents of F contain the respective attribute, their parents are used and so on.

name_of_F	
name_of_parent_1	
……...	
name_of_parent_k	
$attribute_1$	$value_1$
……..	…….
$attribute_n$	$value_n$

Figure 1. The graphical representation of a frame.

2.2 The formal aspect

Syntactically, it is considered that a frame is a formal string([1])

frame(frame_name, list_of_parent_names, list_of_attributes) (1)

where

- *frame_name* is the name of the frame;

- *list_of_parent_names* is a list of frame names which represent the parents;
- *list_of_attributes* is a list of elements having one of the next forms:
  - *attr(name_of_attribute, value);*
  - *attr(name_ of_attribute, proc(name_of_proc));*
  - *attr(name_ of_attribute, daemon).*

The entity *value* represents the direct value of the attribute; *name_of_proc* represents the name of a procedure which calculates the value of the attribute; the word *daemon* means that there is an abstract attribute whose value can be calculated through a procedure or specified as the direct value of a particular object.

Let a *Knowledge Representation and Processing System(KRPS)* based on frames be defined below. The elements of language L_{Repr} will be words of the form (1). As soon as we define the rules of constructing names of frames, names of attributes, names of procedures and attribute values, the elements of L_{Repr} can be constructed. V_{Attr} denotes the set of all direct attribute values. The elements of V_{Attr} can by homogeneous or heterogeneous. Hereby, V_{Attr} can be a subset of the natural numbers set or can contain as well numbers (real, integer), alphabetical strings and number strings. L_{fr_name} denotes the set of names of frames, L_{Attr} denotes the set of names of attributes and L_{parent} denotes the set of names of parents. Because any name of a parent is a frame name, in general $L_{parent} \subseteq L_{fr\_name}$ and in the case there are certain rules concerning the frames which can be parents then there is the strict inclusion $L_{parent} \subset L_{fr\_name}$. L_{proc} denotes the set of names of procedures and consider the set $Q_{Attr} = V_{Attr} \cup \{proc(x) \mid x \in L_{proc}\}$. The sets L_{fr_name}, L_{Attr}, L_{parent} and L_{proc} are formal languages and accordingly for each of them one of the methods that define these languages are applied.

Definition 2.1 *The representation language* L_{Repr} is the set of all words of the form:

$$frame(name, [p_1 \ldots\ldots p_n], [attr(a_1, v_1), \ldots\ldots, attr(a_k, v_k)]) \qquad (2)$$

where $s \geq 0$, $k \geq 0$, *name* □ *Lfr-name,* {*p*1, . . . , *ps*} □ *Lparent,* {*a*1, . . . , *ak*} □ *LAttr* ¸*si vi* □

QAttr □{*daemon*} *for* *i* □ {1, . . . , *k*}

In relation with *Definition 2.1* it can be said that the elements $p_1 \ldots\ldots p_n$ are two by two distinct and the same thing can be told about the elements $a_1 \ldots\ldots a_k$. But then, the same condition for the values of the attributes does not hold and consequently can have the case $v_i = v_j$ and $i \neq j$. In the syntactic structure of any word from L_{Repr} the entity *frame* and *attr* are key words.

Definition 2.2 A finite set of frames syntactically represented by the form (2) is a knowledge base with frames.

K, a knowledge base with frames is considered. *Name(K)* denotes that subset of the language L_{fr_name} having all elements *X* if there is a frame in *K* with the name *X*.

Having a knowledge base with frames *K*, an oriented graph $G_k = (Name(K), \Gamma_K)$ is considered where the nodes are names of frames from K and from node f_u to node f_v there exists an arc, that is $(fu, fv) \in K\Gamma_K$, if f_u belong to the list of f_v parents.

The set of all paths from node x to node y in G_k is denoted by *Path(x,y)*. The element *f* is an *predecessor* of *g* if $f \neq g$ and there exists a path from *f* to *g*. *Pred(g)* denotes the set of all predecessors of g. If f∈Pred(g) then *dist(f,g)* denotes the length of the shortest path from *f* to *g*. If *dist(f,g)=k* then it is said that the "distance" from f to g is k.

Definition 2.3 A predecessor *f* of *g* in the graph G_k is the nearest predecessor with the α property if the next two conditions are realized:

- f satisfies the α property;
- there is any predecessor of *g*, namely *h*, so that *h* satisfy the α property and *dist(h,g)<dist(f,g)*

The nearest predecessor of *g*, N.P $\{\alpha\}(g)$, denotes the set of all nodes that are the nearest predecessors of *g* and satisfy the α property.

Generally, to obtain an admissible knowledge base of a *KRPS*, certain restrictions about components are suggested. Following up, the conditions which are imposed to a knowledge base with frames for being an admissible base are specified.

A knowledge base K with frames is considered. The first condition that is imposed to *K* is named the **C1 condition:**

$$frames(f,X,Y) \in K, frames(f,U,V) \in K \Rightarrow X = U, Y = V$$

This condition imposes the requirement that all the parents and all the attributes of the frame to be defined in a single string of the form (2).

Let us consider a certain frame

frame f,[p1 ...,ps],[attr(a1, v1) ,...,attr(ak, vk)]) (3)

of a knowledge base K with frames who satisfies the **C1** condition.

Because *all* attributes of frame *f* are explicited in (3), so there are no other attributes for *f* besides the attributes represented in (3) and the same property goes for the parents of that frame, there are the following notations:

$Slot(f) = \{(a_1, v_1), ..., (a_k, v_k)\}$
$Parent(f) = \{p_1 ..., p_s\}$

Through language abuse it is said that the name of the attribute a_i belongs to the frame *f* and it is written $a_i \in f$ if there exists $u \in Q_{Attr} \cup \{daemon\}$ so that $(a_i, u) \in Slot(f)$. Sometimes it is said, equivalently, that *f* includes attribute a_i.

For any attribute $a \in L_{Attr}$ let it be denoted that:

$Near_a(f) = \{f\}$ if $a \in f$;
$Near_a(f) = N.P_{\{include\ a\}}(f)$ if $a \notin f$;

Below let it be denoted by **C2,** respectively **C3,** the following conditions:

Condition C2: For each name of frame f and for each name of attribute a wherefore $(a, daemon) \in Slot(f)$:

$g \in \Pr ed(f) \Rightarrow Near_a(g) = \Phi$.

Condition C3: For any $a \in L_{Attr}$ the set $Near_a(f)$ has at most one element.

The requirement of the **C2** condition is bound by the mode that the *daemon* value of an attribute is used and interpreted.

Definition 2.4. Let it be considered a knowledge base with frames *K*. The base *K* is called an **admissible base** if it satisfies the conditions **C1, C2, C3**.

A procedure name *P*, $P \in L_{\Pr oc}$, is considered. The expression $P(v_1, \Lambda, v_t)$ is used in the following to denote the returned value of the *P* procedure when the actual parameters of the procedure are the values v_1, Λ, v_t. Generally, the v_1, Λ, v_t values are the values of the a_1, Λ, a_t attributes. Let it be denoted by $Dom_{Attr}(P) = (a_1, \Lambda, a_t)$.

The value of an attribute can be an element of the set $V_{Attr} \cup \{daemon, unknown\}$.

The value returned by the procedure *P* can be:

1) $P(v_1, \ldots, v_t) \in V_{Attr}$ if $v_1, \ldots, v_t \in V_{Attr}$
2) $P(v_1, \ldots, v_t) = unknown$ if there exists $i \in \{1, \ldots, t\}$ so that $v_i = unknown$;
3) $P(v_1, \ldots, v_t) = daemon$ if for each $i \in \{1, \ldots, t\}$, $v_i \in V_{Attr} \in \{daemon\}$
and $j \in \{1, \ldots, t\}$ so that $v_j = daemon$;

Let it be considered that:
$N_K = \{(f, a) \in Name(K) \times L_{Attr} / \exists g \in Near_a(f), \exists P \in L_{\Pr oc} : (a, proc(P)) \in Slot(g)\}$

If $f \in Name(K)$ and $Near_a(f) = \varnothing$ then $(f,a) \notin N_K$.

Definition 2.5 Let $K \in L_{KB}$ be considered. The application $\Omega_K : N_K \to L_{\Pr oc}$ is defined so that : $\Omega_K(f, a) = P$, where $(a, proc(P)) \in Slot(g)$, $g \in Near_a(f)$.
The application Ω_K is called the **extractive function** for the knowledge base K.
For each $K \in L_{KB}$ the following set is considered:

$M_K = \{(f, a) \in Name(K) \times L_{Attr} \mid (f, a) \notin N_K, Near_a(f) \neq \Phi\}$

The sets N_K and M_K are disjunctive sets.

Proposition 2.1 If $(f, a) \in M_K$ then there exists an element and only one $u \in V_{Attr} \cup \{daemon\}$ so that $(a, u) \in Slot(g)$, where $\{g\} = Near_a(f)$.

Definition 2.6 Let *K*, an arbitrary element in L_{KB}, be considered. The application $Comp_K : L_{fr\_name} \times L_{Attr} \to V_{Attr} \cup \{unknown, undefined, daemon\}$ is defined so that:

- If $(f, a) \in M_K$ then $Comp_K(f, a)=u$, where u is the only one element from the set $V_{Attr} \cup \{daemon\}$ so that $\{g\}=Near_a(f)$ and $(a, u) \in Slot(g)$
- If $(f, a) \in N_k$ and $(b_1, \ldots, b_t) = Dom_{Attr}(\Omega_K(f, a))$, then:
  - $Comp_K(f, a)= \Omega_K(f, a)(Comp_K(f, b_1),...,Comp_K(f, b_t))$, if $Comp_K(f, b_1),....,Comp_K(f, b_t)$ can be calculated.
  - $Comp_K(f, a)=undefined$, if $Comp_K(f, b_1),....,Comp_K(f, b_t)$ cannot be calculated.
- If $(f, a) \in (L_{fr\_name} \times L_{Attr}) \setminus (M_K \cup N_K)$ then $Comp_K(f, a) = unknown$.

The value $Comp_K(f, a)$ of the $Comp_K$ application in the (f, a) argument is the value of the *a* attribute for the frame *f*.

Let $L_Q = L_{fr\_name} \times L_{Attr}$ be the query language and $L_{Ans} = V_{Attr} \cup \{unknown, undefined, daemon\}$ be the answer language.

Definition 2.7 Let the inference relation $\subseteq L_{KB} \times L_Q$ be defined as follows:

$K \in (f, a)$ only if $Comp_K(f, a) \in V_{Attr} \cup \{daemon\}$.

Definition 2.8 The answer function $Ans : L_{KB} \times L_Q \to L_{Ans}$ is defined hereby as follows:

$$Ans(K, (f, a)) = Comp_K(f, a).$$

3. Proposed application

For the illustration of the above concepts let the problem of deciding admission of candidates at a competition.

Let a knowledge base *K* be considered with the following elements:

```
frame("candidate",[],[attr("accepted","proc(calcAccepted)"),
                      attr("age","proc(calcAge)"),
                      attr("birthYear","proc(calcBirthYear)")]).

frame("veronica",["candidate"],[attr("english","very good"),
                      attr("economy","9"),attr("birthYear","1974")]).
frame("mihai",["candidate"],[attr("economy","10"),attr("age","28"),
                      attr("domicile","Craiova")]).
frame("maria",["candidate"],[attr("english","satisfying"),attr("economy","9"),
                      attr("birthYear","1976")]).
frame("ionel",["candidate"],[attr("english","good")]).
```

```
frame("aurel",["candidate"],[attr("english","very
good"),attr("economy","10"),attr("age","14")]).
frame("mirela",["candidate"],[attr("english","very good"),attr("economy","9")]).
frame("simona",["candidate"],[attr("english","good"),attr("economy","8"),
                              attr("age","29"),attr("maried","yes")]).
```

3.1. Implementation remarks

The application that was developed uses Turbo Prolog Language and realises a system based on frames for the determination of the results at a competition.

A candidate is declared accepted if:

1. have 25-30 years;
2. english qualificative-at most good.;
3. economy note-at most 8.

The system takes the date from the computer and define the fact that Satisfying< Good< Very Good.

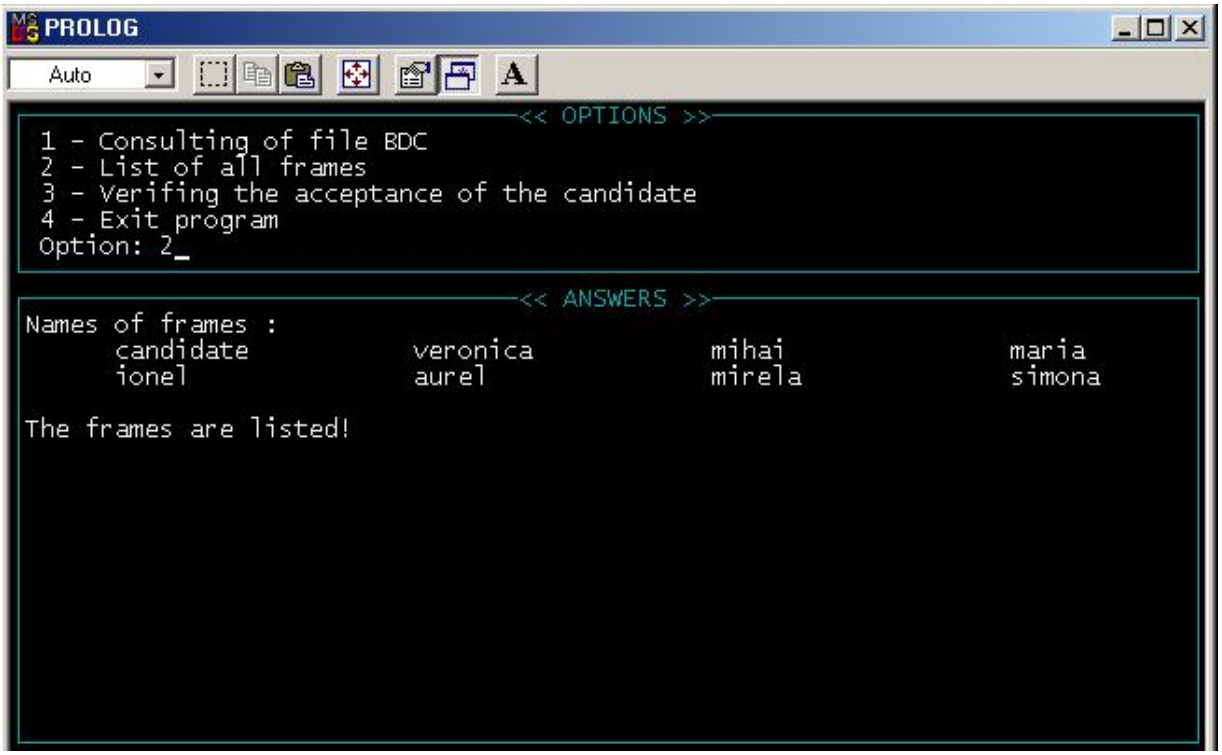
3.2. Experimental results

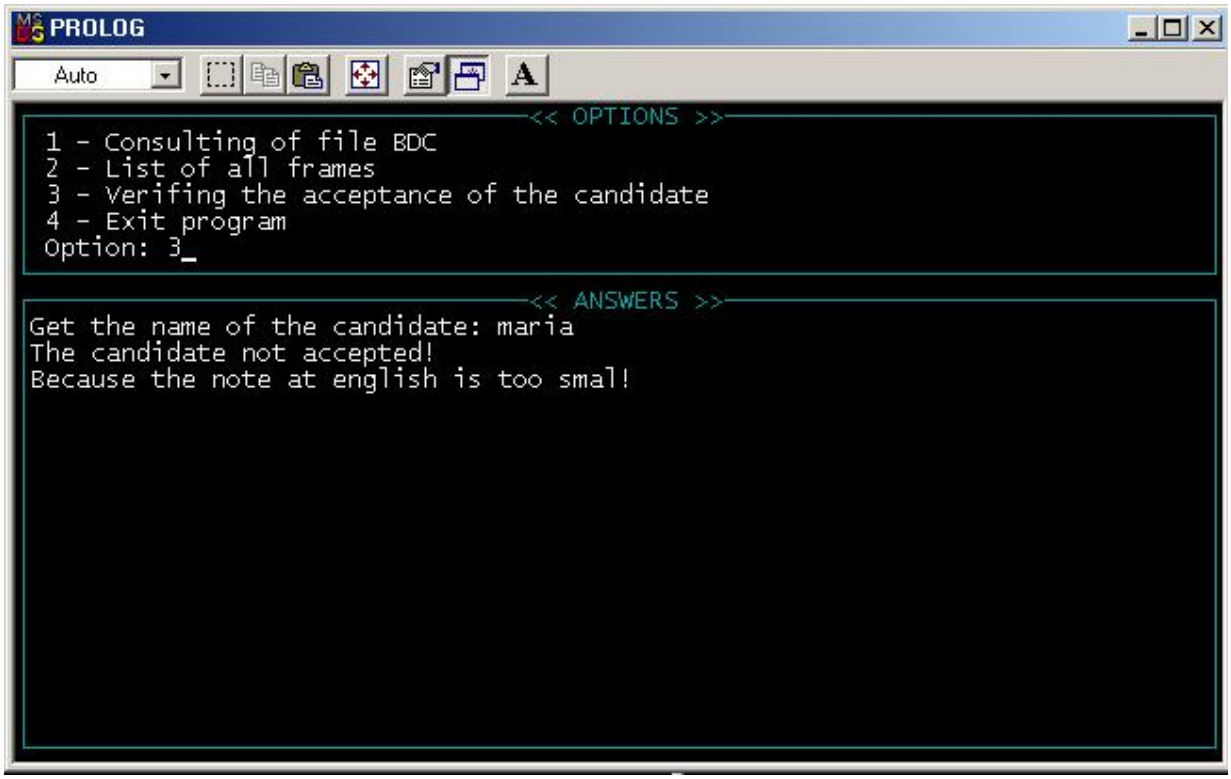
```
domains
    attr = attr(symbol,symbol)
    latribute = attr*
    lparinti = symbol*
database
    frame(symbol,lparinti,latribute)

predicates
    giveVarstaAccept(symbol,latribute)
    varsta(latribute)
    economie(latribute)
    giveEconomieAccept(integer)
    giveEnglishAccept(symbol)
    english(latribute)
    scriuRezAdmis(latribute)
    concatattr(latribute,latribute,latribute)
    apartine(symbol,lparinti)
    apartineattr(attr,latribute)
    findneamu(lparinti,lparinti,lparinti)
    findallattr(lparinti,latribute,latribute)
    scotdubluriattr(latribute,latribute,latribute)
    meniu
    citimOptiune
    execut(symbol)
    scrieFrame(lparinti,integer,integer)
    calcCoord(integer,integer,integer,integer)
    verifAdmis(symbol)
    calificativ(symbol,integer)
clauses
    apartine(X,[X|_]).
    apartine(X,[_|L]):-apartine(X,L).

    apartineattr(X,[X|_]).
```

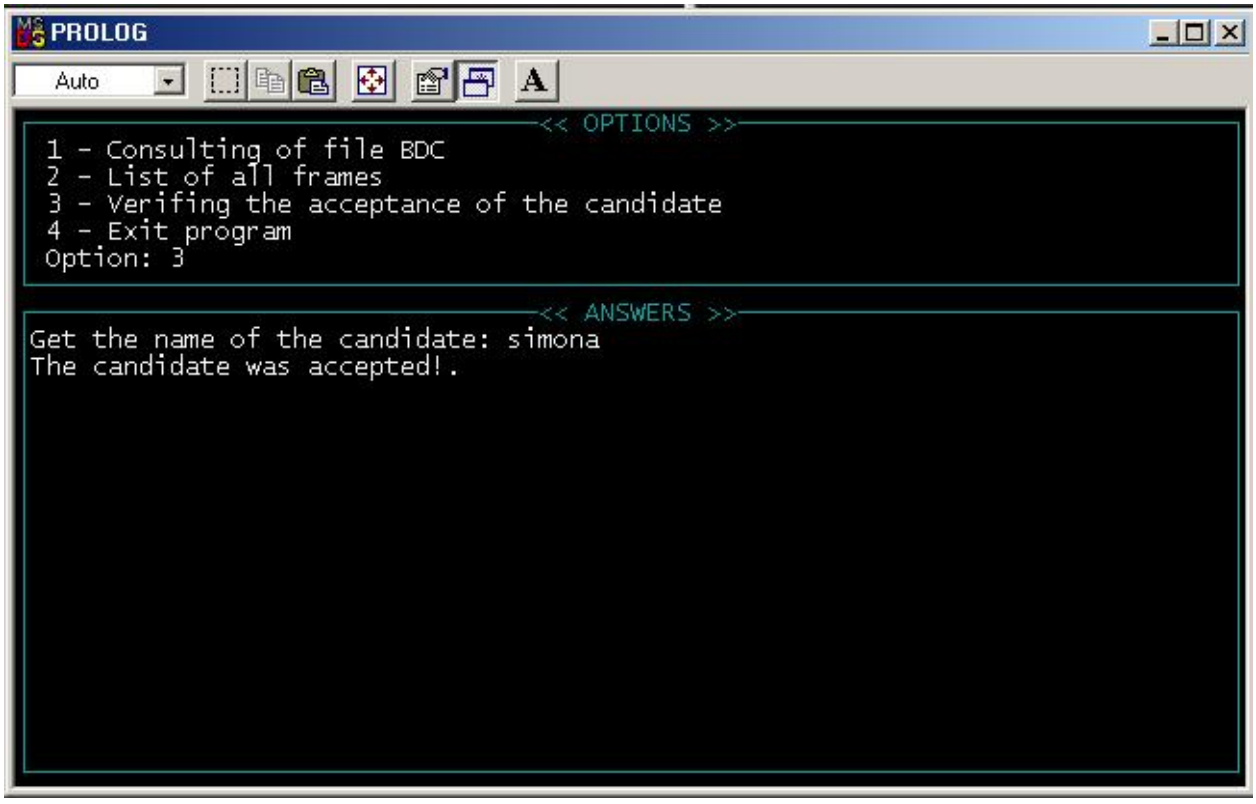
```
apartineattr(X,[_|L]):-apartineattr(X,L).

concatattr([],L,L).
concatattr([X|L1],L2,[X|Lrez]):-concatattr(L1,L2,Lrez).

findneamu([],L,L).
findneamu([Y|L],Ini,Fin):-frame(Y,LP,_),
            findneamu(LP,[Y|Ini],Fin2),
            findneamu(L,Fin2,Fin).
scotdubluriattr([],L,L).
scotdubluriattr([attr(X,Y)|L1],LTemp,LRez):-
           not(apartineattr(attr(X,_),L1)),
           scotdubluriattr(L1,[attr(X,Y)|Ltemp],LRez);
           scotdubluriattr(L1,LTemp,LRez).
findallattr([],L,L).
findallattr([X|LP],Lini,Lrez):-
       frame(X,_,La),
       concatattr(Lini,La,Lini2),
       findallattr(LP,Lini2,Lrez).
calificativ("satisfying",1).
calificativ("good",2).
calificativ("very good",3).
meniu:-shiftwindow(1),
      cursor(0,1),
      write("1 - Consulting of file BDC"),
           cursor(1,1),
      write("2 - List of all frames"),
      cursor(2,1),
      write("3 - Verifing the acceptance of the candidate"),
      cursor(3,1),
      write("4 - Exit program").
citimOptiune:-
      shiftwindow(3),
      write(" Option: "),
      readln(X),
      execut(X).
execut("1"):-
         shiftwindow(2),
         clearwindow,
         consult("comp.bdc"),
         write("Knowledge base was consulted with succes!"),
         citimOptiune.

execut("2"):-shiftwindow(2),clearwindow,
           findall(F,frame(F,_,_),L),
           L=[],
           write("No frames!"),nl,
           citimOptiune;
           shiftwindow(2),
           clearwindow,
           write("Names of frames :"),nl,
           findall(F,frame(F,_,_),L),
           scrieFrame(L,1,6),
           citimOptiune.

execut("3"):-shiftwindow(2),clearwindow,
```

```
        write("Get the name of the candidate: "),
        readln(Nc),
        verifAdmis(Nc),
        citimOptiune.

execut("4").
execut(_):-shiftwindow(2),
       clearwindow,
       write("Inexistent command!"),nl,
       write("Please be careful!"),
       citimOptiune.

calcCoord(X,Y,X1,Y1):-X1=X, Y1=Y+20,Y1<80;
                Y1 = 6,X1 = X+1.

scrieFrame([],_,_):-nl,nl,write("The frames are listed!").
scrieFrame([F|L],X,Y):-cursor(X,Y),
                write(F),
                calcCoord(X,Y,X1,Y1),
                scrieFrame(L,X1,Y1).

verifAdmis(Nc):-not(frame(Nc,_,_)),
           write("The candidate does not exist!").
verifAdmis(Nc):-frame(Nc,LP,LA),
           findneamu(LP,[],NewL),
           findallattr(NewL,[],NewLA),
           concatattr(NewLA,LA,NewLa2),
           scotdubluriattr(NewLa2,[],L),
           scriuRezAdmis(L).

english(L):-apartineattr(attr("english",Calificativ),L),
        giveEnglishAccept(Calificativ).
english(L):-not(apartineattr(attr("english",_),L)),
        write("The candidate not accepted!"),nl,
        write("Because don't have the note at english!").
giveEnglishAccept(Calificativ):-
        calificativ(Calificativ,X),X<2,
        write("The candidate not accepted!"),nl,
        write("Because the note at english is too smal!").
varsta(L):-apartineattr(attr("age",X),L),
       giveVarstaAccept(X,L).
varsta(L):-not(apartineattr(attr("age",_),L)),
       write("The candidate not accepted!"),nl,
       write("Because the age is unknown!").
giveVarstaAccept(X,_):-str_int(X,Y),Y<31,Y>24,
               write("The candidate was accepted!.").
giveVarstaAccept(X,_):-str_int(X,Y),Y<25,
               write("The candidate not accepted!"),nl,
               write("Because the age is not in an admisible limit!").
giveVarstaAccept(X,_):-str_int(X,Y),Y>30,
               write("The candidate not accepted!"),nl,
               write("Because the age is not in an admisible limit!").
giveVarstaAccept(_,L):-
               apartineattr(attr("birthYear",X),L),
               str_int(X,YN),
               date(Y,_,_),
```

```
                Varsta = Y-YN,
                str_int(StrVarsta,Varsta),
                giveVarstaAccept(StrVarsta,L).
    giveVarstaAccept(_,_):-
                write("The candidate not accepted!"),
                nl,write("Because the age cannot be calculated!").

    economie(L):-apartineattr(attr("economy",Nota),L),
            str_int(Nota,NotaInt),
            giveEconomieAccept(NotaInt).
    economie(L):-not(apartineattr(attr("economy",_),L)),
            write("The candidate not accepted!"),nl,
            write("Because the note at economy not exist!").
    giveEconomieAccept(Nota):-
                Nota<8,
                write("The candidate not accepted!"),nl,
                write("Because the note at economy is too smal!").
    scriuRezAdmis(L):-
                english(L);
                economie(L);
                varsta(L).

goal
    makewindow(1,15,3,"<< OPTIONS >>",0,0,7,80),
    makewindow(2,15,3,"<< ANSWERS >>",7,0,18,80),
    makewindow(3,15,0,"",5,1,1,76),
    meniu,
    citimOptiune,
    removewindow,
    removewindow,
    removewindow,
    retractall(frame(_,_,_)).
```


PROLOG
Auto
<< OPTIONS >>
1 - Consulting of file BDC
2 - List of all frames
3 - Verifing the acceptance of the candidate
4 - Exit program
Option: 2
<< ANSWERS >>
Names of frames :
candidate veronica mihai maria
ionel aurel mirela simona
The frames are listed!


PROLOG
Auto
<< OPTIONS >>
1 - Consulting of file BDC
2 - List of all frames
3 - Verifing the acceptance of the candidate
4 - Exit program
Option: 3
<< ANSWERS >>
Get the name of the candidate: maria
The candidate not accepted!
Because the note at english is too smal!



Looking at the structure of the implicated frames and at the rules of acceptance, the obtained results proved to be correct.

4. Conclusions

Minsky's frames proved to be a very appropriate way to encode information and fast and accurate means of deciding acceptance of proposed candidates to a competition.

5. Acknowledgements

I would like to thank Professor Nicolae Tandareanu, Professor Ruxandra Gorunescu and Professor Catalin Stoean who helped me through the making of this article. I would like to thank also to my work mate Orlando Gabriel Pirlog.

6. References

[1] N. Tăndăreanu: Sisteme expert. Reprezentarea cunoştinţelor şi inferenţa, Editura Universitaria, 2001

[2] http://ugrad-www.cs.colorado.edu/~cs3202/papers/Josh_Hogrewe.html

[3] http://www.arches.uga.edu/~jhobson/minsky2.htm

Gabriel Voinescu,
student in the 3rd year at the Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova. Recently, I was appointed junior member at the Research Center for Artificial Intelligence (RCAI). My hobbies are: computers (especially Operating Systems), all good music and walks.

A method to compute the last non zero digit of n!

Abstract. The study of determining the last non zero digit of n! has been of wide interest to mathematicians and computer scientists over the time. Multiple methods have been developed. Present paper proposes yet another technique built to provide an algorithm of a very good complexity.

1. Introduction

A simple method for the problem would be to effectively calculate n!, and then find the last non zero digit. The inconvenients of this method would be complexity of O(n) order and also the fact that the number would be very big and difficult to retain in an elementary type of data, needing the implementation of a type of data (array of digits) which should retain very big numbers. This entails the increase of the run time and also of the complexity which becomes O($n(\log_{10} i)^2$).

2. Proposed method

Proposed method provides a very good complexity.
It is built upon certain properties of the function that provides the last non zero digit for a number and on some particular characteristics of n!.

2.1 Properties of the function to provide the last non zero digit of a number

Let *f* be the function to provide the last non zero digit, *e.g.* *f*(650) = 5 - the last non zero digit of 650.
Some of its properties are studied below.

Proposition. f(n) represents the remainder of the division of n to 10 if n does not belong to M_5 (M_5 denotes multiples of 5)

Proof.

n not being multiple of 5, it results that it is not multiple of 10 either and that the last digit of n is a non zero digit.

Proposition. f(f(n)) = f(n);

Proof.

f(n) is a non zero digit, therefore *f* applied to a digit is the digit itself. Therefore if x = f(n) than f(f(n)) = f(x) = x;

Example:

f(f(302)) = f(2) = 2 = f(302);

Proposition. f(a*b) = (f(a)*f(b)) mod 10 if a and b do not belong to M_5 (by *n mod v* it is understood the remainder of the division of *n* to *v*). Since a,b $\notin M_5$ it results that f(a*b) is

indeed the last digit of number n which is obtained by the product between the last digit of numbers a and b.

2.2 Properties of n!

n! is the product of all the numbers from 1 to n or $n!=\prod_{i=1}^{n} i$.

If $n \in M_5$ this product can be rewritten as a product of products of 5 successive numbers

n!=(1*2*3*4*5)*(6*7*8*9*10)*..*((n-4)*(n-3)*(n-2)*(n-1)*n)

or $n!=\prod_{i=1}^{n/5}\prod_{j=1}^{5}(5*i-5+j)$

Example:

50!=(1*2*3*4*5)*(6*7*8*9*10)*..*(46*47*48*49*50)

If $n \notin M_5$, n! = m!*(m+1)*..*(n) where m, the biggest multiple of 5 smaller or equal to n, is rewritten afterwards.

Thus it results that n!=((1*2*3*4*5)*..*((m-4)*(m-3)*(m-2)*(m-1)*m))* (m+1)*..*(n)

This product can also be written as

$n!=\left(\prod_{i=1}^{[n/5]}\prod_{j=1}^{5}(5*i-5+j)\right)*\prod_{k=[n/5]*5+1}^{n}k$ or as a product of all the elements of this array.

	Li
i = 1	1*2*3*4*5
i = 2	6*7*8*9*10
i = [n/5]	([n/5]*5-4)*([n/5]*5-3) *([n/5]*5-2)*([n/5]*5-1)*([n/5]*5)
i = [n/5]+1	([n/5]*5+1) *([n/5]*5+2)*..*(n)

where L_i for i < [n/5] is

$Li=(5*i-4)*(5*i-3)*(5*i-2)*(5*i-1)*(5*i) \Rightarrow$

$L_i=(5*i-4)*(5*i-3)*(5*i-2)*(5*i-1)*5*i$

where [x]=is the biggest integer smaller or equal to x.

$R_n = L_{[n/5]+1}$ in case $n \notin M_5$ or $R_n = 1$ otherwise.

$C_i = L_i / i$

It results that

$$n! = C_1 * 1 * C_2 * 2 * .. * C_{[n/5]} * [n/5] * L_{[n/5]+1}$$

$L_{[n/5]+1}$ is replaced by R_n.

$$n! = C_1 * C_2 * .. * C_{[n/5]} * 1 * 2 .. * [n/5] * R_n$$

1*2*..*[n/5] is replaced by [n/5]!

$$n! = C_1 * C_2 * .. * C_{[n/5]} * [n/5]! * R_n$$

n! depends on [n/5]!

$$[n/5]! = C_1 * C_2 * .. * C_{[n/25]} * [n/25]! * R_{[n/5]}$$

……

$$[n/5^{[n/5]}] = C_1 * [n/5^{[n/5]}]! * R_{[n/5^{[n/5]}]}$$

From proposition 2 it results that:

$$f(C_i) = f(1*2*3*4*5) = 2 \text{ for } i \in \{1..[n/5]\} - M_2$$
$$f(C_i) = f(6*7*8*9*5) = 2 \text{ for } i \in \{1..[n/5]\} \cap M_2$$

It then results that $n! = 2^{[\log_5 n]} * [n/5]! * R_{[n/5]}$.

By introducing in this relation the formula for [n/5]! and then recurrently $[n/5^2]$,$[n/5^3]$ … factorial it results:

$$n! = 2^{\sum_{i=1}^{[n/5]} [\log_{5^i} n)]} * \prod_{I=1}^{[n/5+1]} R_{[n/5^i]} \Rightarrow$$

$$f(n!) = \left(f\left(2^{\sum_{i=1}^{[n/5]} [\log_{5^i} n]} \right) * f\left(\prod_{i=1}^{[n/5]+1} f(R_{[n/5^i]}) \right) \right) \bmod 10$$

It can be observed that $f(2^n)$ is periodical of period 4 $f(2^k) = f(2^{k+4})$ for k>0 because $2^k * 16 = 2^{k+4}$, $f(2^k * 16) = f(f(2^k) * f(16)) = f(f(2^k) * 6)$;

$f(2^k) \in \{2,4,6,8\} \Rightarrow f(f(2^k) * 6) = f(2^k)$ because

f(2*6)=2;

f(4*6)=4;
f(6*6)=6;
f(8*6)=8;

$$f(2^n)=\begin{cases}1, n=0\\ 6, n\in 4i+1, i\in N^+\\ 2, n\in 4i+1, i\in N^+\\ 4, n\in 4i+2, i\in N^+\\ 8, n\in 4i+1, i\in N^+\end{cases}$$

What remains to be calculated is only the index of the power of 2 and the product of $R_{[n/5^i]}$ at each step

Example for n=302

	Li
i=1	1*2*3*4*5
i=2	6*7*8*9*10
I=[302/5]=60	296*297*298*299*300
I=[302/5]+1=61	301*302

$C_i = L_i / i \Rightarrow$

$C_1 = 1*2*3*4*5$

$C_2 = 6*7*8*9*5$

$C_3 = 11*12*13*14*5$

.....

$C_{60} = 296*297*298*299*5$

It can be noticed that

$302! = C_1 *1* C_2 *2* C_3 *3*...* C_{60} *60*301*302 \Rightarrow$

$302! = C_1 * C_2 * C_3 *..* C_{60} *1*2*3.*.*60*301*302 \Rightarrow$

$302! = C_1 * C_2 * C_3 *..* C_{60} *60!*301*302$

The algorithm is applied for n=60

	Li

i = 1	1*2*3*4*5
i=2	6*7*8*9*10
i=[60/5]=12	56*57*58*59*60
I=[60/5]+1=13	

$C_i = L_i / i \Rightarrow$

$C_1 = 1*2*3*4*5$

$C_2 = 6*7*8*9*5$

.....

$C_{12} = 56*57*58*59*5$

It can be noticed that

$60! = C_1 *1* C_2 *2*...* C_{12} *12 \Rightarrow$

$60! = C_1 * C_2 *..* C_{12} *1*2.*.*12 \Rightarrow$

$60! = C_1 * C_2 *..* C_{12} *12!$

The algorithm is applied for n=12.

	Li
i=1	1*2*3*4*5
i=2	6*7*8*9*10
I=[12/5]+1=3	11*12

$C_i = L_i / i \Rightarrow$

$C_1 = 1*2*3*4*5$

$C_2 = 6*7*8*9*5$

The algorithm is applied for n=2

n!=2;

In conclusion:

$$302! = C_1 * C_2 * C_3 * .. * C_{60} * 60! * 301 * 302 \Rightarrow$$
$$302! = C_1 * C_2 * C_3 * .. * C_{60} * C_1 * C_2 * .. * C_{12} * 12! * 301 * 302 \Rightarrow$$
$$302! = C_1 * C_2 * C_3 * .. * C_{60} * C_1 * C_2 * .. * C_{12} * C_1 * C_2 * 2! * 11 * 12 * 301 * 302 \Rightarrow$$
$$302! = C_1^3 * C_2^3 * C_3^2 * .C_{12}^2 * C_{13} * C_{60} * 2 * 11 * 12 * 301 * 302 \Rightarrow$$
$$f(302!) = (f(2^{60+12+2}) * f(2 * 11 * 12 * 301 * 302)) \bmod 10 \Rightarrow$$
$$f(302!) = (4 * f(2 * 1 * 2 * 2)) \bmod 10 \Rightarrow$$
$$f(302!) = 32 \bmod 10 = 2;$$

3.Conclusions

This method has the complexity $O(\log_5 n)$, a lot smaller than other methods. Proposed method resolves many of the inconvenients of the others not only from the memory point of view, not needing the allocation of a considerable memory area, but also of the type and implicitly of the number of operations.

Example:

For n=1220703129 only 15 divisions are made and 29 operations of finding the remainder.

4.References

[1] http://www.acm.inf.ethz.ch/ProblemSetArchive/B_US_SouthCen/1997/Facts.html

5.Acknowledgements

I would like to thank Professor Nicolae Ţăndăreanu for giving me the opportunity to write this article.

I would like to thank Professor Ruxandra Gorunescu and Professor Catalin Stoean for the help and support in the writing of this article. Many thanks go also to my colleagues: Mihai Dobrin, Dumitru Marusi, Orlando Pirlog for our first attempt in solving this problem some years ago.

Sandu Andrei-Constantin was born in Craiova in 1982. In the present, he is a student of the Faculty of Mathematics and Computer Sciences at the University of Craiova and he is a junior member in a group of students from the same university who dedicate their time to finding new ways of improving the knowledge they have acquired from their teachers, by participating to local, national and international contests where he obtained good results. He also studied in Spain in 2003, at the Autonoma University of Madrid.

The Rummy Game

Abstract

Games are often encountered. Because almost everyone uses computes, the games industry evolved into this particular domain. I am trying to present in the following a little about how the computer games are made.

Card games - Introduction

Card games constitute one of the playing domains. This type of games is very interesting and a lot of people spend time and money on it. They are widespread in all the casinos and at any corner.

Hundreds of card games have been devised over the centuries, but relatively few have had lasting appeal. Poker, for instance, is based on several games that no longer exist.

A possible classification of card games into five broad categories was made:

- The first group includes the trick-winning games, in which certain cards or an entire suit are designated trumps (highest ranking cards). Among these are the various forms of whist, bridge, and euchre.
- A second group comprises games in which the object is to own or win certain valuable counting cards and sometimes to show specific scoring combinations known as melds. Among such games are pinochle, bezique, and piquet. The non trump game of casino and the game of hearts, in which the aim is to avoid the capture of counting cards, can also be included in this category.
- The object of another group of games is to obtain a given score by matching, assembling, or discarding cards. Of these, the rummy games are the most widely played.
- Constituting a fourth category are the showdown games, in which players wager that they can show cards, or combinations of cards, outranking those of their opponents. Poker is the best known of the showdown games.
- A final group, based on adding or matching numbers, includes such betting games as black jack (also known as twenty-one), baccarat, chemin de fer, and cribbage.

For more information see [4].

Rummy game particularities

The Rummy is a category of card games. There are many types of rummy. This rummy game is a board game and the payer's cards are little plastic boxes with numbers on them. The PC game doesn't implement all the rules of the real game, but the base is built.

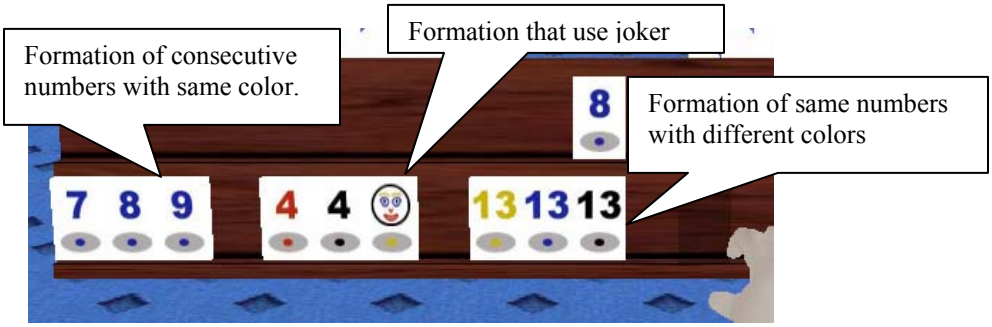
A player is declared winner if all its cards represent valid formation of minim 3 cards. There are two types of formation:

- Same color, consecutive numbers
- Different colors, same numbers.

There are 106 cards: eight formations of 1 to 13 numbers of four colors – two colors for each, and 2 jokers. The cards are randomized in 7 cards columns, resulting 15 columns and one card. The number of that card represents the first playable column. Starting with column, each player receives two columns, resulting 14 cards for each of them. The first player receives one card more.

If a player has 15 cards, he must drop one off its board and if he has 14 and it is his turn, he must take the next card from the cards columns.

The joker can be used in place of any card.



The consecutive formations are calculated modulo 13. It means that the formation (12, 13, 1) is valid. In the same number formations there can't be use cards with same color.

How does the computer think?

The Rummy Game implements an algorithm for computing the pieces list according to the rules of the game. At this level, the algorithm is very simple. The priority is represented by the consecutive numbers formations. This algorithm is presented below:

- The pieces are sorted by colors and by numbers.
- The consecutive formations are isolated from the others pieces.
- With the remaining pieces it is tried to build the other type of formation (same number, different colors).
- The joker will be appended only at the 2-pieces formation of any of the two types (this is a disadvantage because if the player has all formations built and one joker, he can't append the joker to a formation).
- The dropped piece will be the last one. Here, again it is a discomfort. The remaining pieces are ordered by colors.

Why 3-D?

Today, most of the games are made in 3-D. A 3-D game is more real than a 2-D game. In this rummy game you will note that particularity. Having control of the camera makes you believe that you are really playing rummy with some friends. The hand is very important, too. You don't have only a rectangle to select a piece, but that piece 'is in your hand'.

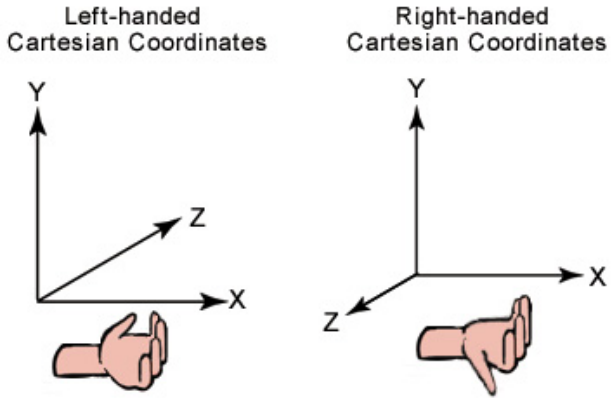
Now let's see how a 3-D game is build. The important thing is to know something about Physics and Three Dimensional Geometry.

Physics helps create games to appear more real. The physical principles are based on true demonstrations and they represent the mathematical interpretation of the things that happen around us. If a leaf falls from a tree, it will never *go up*. Till then it will hit the ground. To this mechanism the gravitational force, the weight of the leaf, the wind, they all contribute. If you push a ball on the ground there are some particular issues that characterize the ball's movement. Here the chafe force intercedes that slows down the ball's speed. Physics demonstrate mathematical formulas for the speed, acceleration, and the direction that the ball will follow. These formulas are used in games and in this way, the player is really enjoying seeing that the real things can be simulated in a virtual world. Here it is more than calculating formulas and drawing a world. The virtual world, and in particular the games try to create what we perceive as being natural. But, can it be created as it is?

In the Rummy Game Physics isn't used, so our discussion won't continue in this domain. For more information see [3].

The next important thing is how to represent the computations made by Physics. Geometry defines a set of formulas used to manipulate objects in a predefined space. The games use 3-D axe who define our work space (scene). There are two types of Cartesian Coordinates:

- Left-handed Cartesian Coordinates.
- Right-handed Cartesian Coordinates.



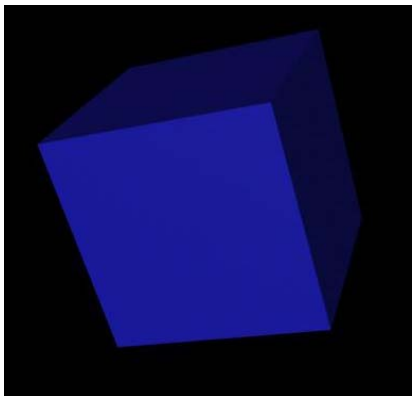
The type of Cartesian coordinates used determines the order in which the vertices of the object will appear in the object. If in left-handed Cartesian coordinates a triangle

has its vertices v0, v1, v2, in right-handed Cartesian coordinates it must has the vertices v0, v2, v1.

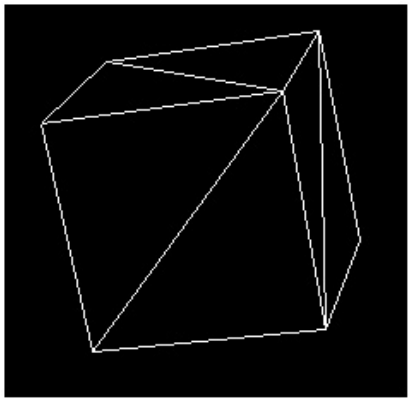
A 3-D primitive is defined as a collection of vertices that forms a single 3-D entity.

Complex primitives are polygons. The simplest polygon is the triangle. 3-D objects, generally, are formed only from triangles, because is very simple to create them. In conclusion, a 3-D object is a collection of vertices, linked between them and the result is a closed surface. Its components are: vertices, edges and faces.

This box is a 3-D object. It has 8 vertices, 18 edges and 12 faces. The faces are triangles.



Smooth box



Wire box

For more information see [1] and [2].

From this point of view, a game can be defined as a lot of 3-D objects that are moving in the scene. Every game has its own rules that will be obeyed by these objects. An object can be a human body, a car, etc. For example, in the rummy game, the objects are: the table and the chairs, the user's board on which the pieces are placed, the hand and the pieces. The only moveable objects are the pieces. Their coordinates are changing using the 3-D transformation.

It is important to notice that, at the beginning of the game, all the objects have their coordinates around the scene's origin (0, 0, 0). In this way, the transformations can be easily applied to the object. And anyway, any object must be built with its local coordinates. When the object is inserted into the scene the reference to the object is made through its local coordinates.

3-D Transformations

In applications which work with 3-D graphics the most commonly used are transformations. You can use geometrical transformations to do the following:

- Change objects position.
- Resize objects dimension.
- Rotate objects around one given point.

- Change viewing positions, directions, and perspectives.

A point can be transformed into another point by using a 4×4 matrix. In the following example, a matrix reinterprets the point (x, y, z), producing the new point (x', y', z').

$$[x'y'z'1] = [x\ y\ z\ 1]\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

Transformations are of three types:

- Translation.

  The vector's components (T_x, T_y, T_z) are added to the point's components (x, y, z) and the result is the point (x', y', z').

$$[x'y'z'1] = [x\ y\ z\ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

- Scaling.

  The vector's components (S_x, S_y, S_z) are multiplied to the point's components (x, y, z).

$$[x'y'z'1] = [x\ y\ z\ 1]\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation.

  There are three types of rotations, each one of them on one axis, and the result is the rotated point around the axis with the angle θ:

  - Around the x-axis:

$$[x'y'z'1] = [x\ y\ z\ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Around the y-axis:

$$[x'y'z'1] = [x\ y\ z\ 1]\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Around the z-axis:

$$[x'y'z'1] = [x\ y\ z\ 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These are the things needed to move objects in a 3-D space.

In the rummy game there are used only translation and rotation. A little example will explain more about this. To put a piece to the 2nd user's board there must be performed the next computations for the piece's coordinates:

R_X (PI + 0.1) * T (0 + x, 102 + y, -51 + z) * R_Y (PI / 2).

For the translation vector, x, y and z values are computed with help from the index of the piece's position on the user's board and (0, 102, -51) is the position of the user's board. For the 3rd and 4th user, the R_Y's angle must be PI and 3 * PI / 2.

Conclusions & Future Work

- There must be made several improvements in using the joker. At this level, the joker can be inserted only at the end of the formation, not inside it.
- For the dropped piece, there is no sorting algorithm to determine the distance between a piece and a corresponding formation. One of my goals concerning improvement of the application is to build an algorithm in this scope. For any of the remaining pieces, the distance between it and any of the corresponding formation will be computed. Afterwards, the pieces will be sorted using this flag. The longest distanced piece will be dropped.
- When a player closes his board, there must be made the computation of the pieces for all the players. But this in a career game version.
- Other important improvement that can be made is to compute all the possibilities to have formations and to choose the optimal one.

References

[1] – DirectX Documentation for C++.
[2] – Computational Geometry course (MIT Craiova – Mihaela Sterpu).
[3] – High school Physics manual (Mechanics).
[4] – History of Card Games (http://www.play-online-black-jack.com).

About me

I am 21 years old and I am from Piatra-Olt, Olt. When I was a kid I often played with my friends. The first thing that I did with a computer was playing (The Lion King).

Maybe it does not sound to well, but now, I do not like to play computer games as I used to do before. It is a problem of etics. Almost all the computer games encourage to violence.

I like classical music, geometry and all the scients that uses it (I just like them, this does not mean that I know them well). I also like everything that is natural (trees, lakes, sky and walking through them). And I like all the nice things.

I hope you will enjoy with the new 3D Rummy Game.

Udrescu Bogdan George, MIT Craiova, December 2003.
E-mail: m3d@email.ro

An application to Knowledge Representation and Processing Systems

1. Abstract

This paper refers to the category of Knowledge Representation and Processing System. A system of this type represents a method to store, manipulate and consult information. An application to illustrate such a system is presented.

2. Introduction

A Knowledge Representation and Processing System (KRPS) is a compact collection of components, well linked one to another, a collection which makes the system "think". The system is able to give an answer to a question, and for that it uses a knowledge base and a specific knowledge representation method. The knowledge base used by the KRPS is a collection of entities having a syntactical structure in concordance to the knowledge representation method, which gives the opportunity, after a process of reasoning, to get information about the objects represent in the base. The roles of systems of this kind are: firstly to give answers about the objects represented in the knowledge base and the secondly to update the knowledge base([3]).

3. Theoretical fundamental concepts of a KRPS

Let V be an alphabet, $V \neq \emptyset$ and V^+ the smallest set with the next properties:

- $V \subseteq V^+$
- if $w_1 \in V^+$ and $w_2 \in V^+$ then $w_1 w_2 \in V^+$.

The elements of V^+ are called "words" over the alphabet V.

For each element $w \in V^+$ there is a $k \geq 1$ and there are $a_1, a_2, ..., a_k \in V$ so that $w = a_1 a_2 ... a_k$. The k number is called the length of the w word. If $w_1 \in V^+$ and $w_2 \in V^+$ then the word $w_1 w_2$ is obtained by the concatenation of w_1 and w_2. Any subset of V^+ is called a language over V. In the theory of knowledge bases the case when k is equal with 0 is not considered, although in the general language theory the null word is used.

A Knowledge Representation Processing System is a set of components which mean in their entirety:

- the syntactical structure of the entities;
- the functions that the system realizes;
- the way of reasoning.

Let V be an alphabet and let L_{Repr} be a subset of V^+ ([1]).

Definition 1.1

A knowledge base K over V is a finite subset of L_{Repr}.

In consequence, to define a knowledge base it is necessary to specify:

- the representation language L_{Repr}, the V alphabet and the constructions rules for the elements of L_{Repr}.
- the rules for the $K \subseteq L_{Repr}$ base elements selection ([1]).

There is a knowledge base $K \subseteq L_{Repr}$ and a restrictions set, R, over its elements. If all K's elements satisfy the restrictions then the base is called an *admissible knowledge base.* The restrictions set, R, is the same for all bases attached to the KRPS. Let L_{KB} be the set of all admissible knowledge bases with respect to the set R.

Let $K \in L_{KB}$ be an admissible knowledge base. A K interrogation is the process where-through we ask a question to the KRPS. A question is an element of L_Q - the query language. The answer to the question is an element of L_{Ans} - the answering language. Let $w \in L_Q$ be a question addressed to the relative system over an admissible knowledge base, then $Ans(K, w)$ is the answer given by the system. So, *the answer function* is $Ans : L_{KB} \times L_Q \to L_{Ans}$. The answer to an interrogation is obtained after a *deduction*. The deduction is a binary relation, it is characteristic to each system and it is symbolized by $\vdash\!\!\!- \subseteq L_{KB} \times L_Q$. The deduction is the same for any admissible knowledge base of the system. The deduction and the answer function are well linked. If $K \vdash\!\!\!- w$ then $Ans(K, w)$ contains all the w interrogation solutions.

Let L_I be an *introduction language*. Its elements are used to update a knowledge base. The update process is defined by the function $Upd : L_{KB} \times L_I \to L_{KB}$, which is called *the update function.* The update function is applied to an admissible knowledge base (its function is to modify or create new elements for the knowledge base) and another admissible knowledge base is obtained in the process.

Definition 1.2

A system $S = (L_{KB}, L_Q, L_{Ans}, L_I, \vdash\!\!\!-, Ans, Upd)$ is called a Knowledge Representation and Processing System.

The component of S have the following significance.

- L_{KB} represents the all admissible knowledge bases set. An admissible knowledge base satisfies the restrictions set, R.
- L_Q represents the query language.
- L_{Ans} represents the answering language.
- L_I represents the introduction language.
- $\vdash\!\!\!-$ represents the deduction relation realized by S.

- $Ans : L_{KB} \times L_Q \rightarrow L_{Ans}$ represent the answer function.
- $Upd : L_{KB} \times L_{KB}$ represents the update function ([1]).

Concerning the deduction relation of a KRPS, a lot of properties have been studied over the time.

Let S be a KRPS, $S = (L_{KB}, L_Q, L_{Ans}, L_I, \vdash\!\!\!-, Ans, Upd)$, and for a given knowledge base $K \in L_{KB}$ let $C(K) = \{w \in L_Q \mid K \vdash\!\!\!- w\}$.

The reasoning defined by the deduction relation α of the system is:

- reflexive, if $K \subseteq C(K)$;
- idempotent, if $C(K) = C(C(K))$;
- monotone, if $K_1 \subseteq K_2 \Rightarrow C(K_1) \subseteq C(K_2)$ ([1])

From all of these properties, the monotone property is the most important. A reasoning is monotone if all things that are deduced from any admissible knowledge base, are still deduced from any other admissible knowledge base that includes the first one.

4. Representing and Processing data using KRPS.

The S_F, S_C, S_P and S_V sets are introduced.

- S_F is a set of symbols of functions. To each symbol of this type, a natural number called the arity of the respective symbol is attached. If a symbol $f \in S_F$ is a symbol with the arity equal to *n*, the symbol will be represented by $f^{(n)}$.
- S_C is a set of symbols called constants.
- S_P is a set of symbols of predicates. To each symbol of this type, as in the case of symbols of functions, a natural number called the arity of the respective symbol is attached.

The triplet $B = (S_F, S_C, S_P)$ is called a *base*.

Let S_V be a set of symbols of *variables*. A term is a constant, a variable or of the form $f(t_1,...,t_k)$ where $f^{(k)} \in S_F$ and $t_1,...,t_k$ are terms over the base B.

An *atom* over the base B is an element that has the following form: $p(t_1,...,t_n)$ where $p^{(n)} \in S_P$ and $t_1,...,t_n$ are terms over the base B ([1]).

A *formula* is an atom, the negation of a formula, the disjunction of two formulas, the conjunction of two formulas. The negation of the F formula is symbolized by $-F$. The disjunction of the F and G formulas is symbolized by $F \vee G$. The conjunction of the F and G formulas is symbolized by $F \wedge G$.

Example 1

Let $B = (\{f^{(1)}\}, \{a\}, \{p^{(1)}, q^{(2)}\})$ be a base. Let L_{KB} be the collection of all sets over the base B that do not contain variables. Let L_Q be the set of all formulas over the base B using as variable the set $S_V = \{x\}$. For each $K \in L_{KB}$ the deduction relation is defined as:

- If X is an atom that does not contain variables then:
  - (D1) $K \vdash\!\!\!- X$ if $X \in K$.
  - (D2) $K \nvdash\!\!\!- X$ if $X \notin K$.
- If F and G are formulas that do not contain variables over the base B then:
  - (D3) $K \vdash\!\!\!- F \vee G$ if $F \in K$ or $G \in K$.
  - (D4) $K \vdash\!\!\!- -(F \vee G)$ if $-F \in K$ and $-G \in K$.
  - (D5) $K \vdash\!\!\!- F \wedge G$ if $F \in K$ and $G \in K$.
  - (D6) $K \vdash\!\!\!- -(F \wedge G)$ if $-F \in K$ or $-G \in K$.
  - (D7) $K \vdash\!\!\!- --F$ if $F \in K$.
- If F is a formula that does contain variables then $K \vdash\!\!\!- F$ if there is a substitution σ so $K \vdash\!\!\!- F_\sigma$.

Let $L_{Ans} = \{yes, no\} \cup 2^{Sub}$, where Sub is the set of all substitutions.

Let $Ans : L_{KB} \times L_Q \to L_{Ans}$ be defined by the relation:

- If w is a formula that does not contain variables then:

$$Ans(K, w) = \begin{cases} yes, \ if \ K \vdash w \\ no, \ if \ K \vdash -w \end{cases}$$

- If w is a formula that does contain variables then:
  $Ans(K, w) = \{\sigma \in Sub \mid K \vdash\!\!\!- w\sigma\}$.

Let L_I be the set of atoms over the base B that does not contain variables. The update function $Upd : L_{KB} \times L_I \to L_{KB}$ is defined by the relation $Upd(K, w) = K \cup \{w\}$, where "$\cup$" is the reunion operation defined in the theory of sets.

Let $S = (L_{KB}, L_Q, L_{Ans}, L_I, \vdash\!\!\!- Ans, Upd)$ be a KRPS. Defining the components of a KRPS means to create the general setting of knowledge representation and the way that these can be processed.

Let $K = \{p(a), q(a,a), p(f(a)), q(f(a), a)\}$ be a knowledge base for the system S. K is an admissible base because it contains only atoms without variables.

The function Ans can be evaluated.

- $Ans(K, p(a) \wedge q(a, f(a))) = no.$
  The answer is "no" because $K \vdash\!\!\!- p(a)$ but $K \nvdash\!\!\!- q(a, f(a))$.
  The rule used to demonstrating this is D5.
- $Ans(K, p(f(a)) \vee q(f(a), f(a))) = yes$.
  The answer is "yes" because $K \vdash\!\!\!- p(a)$. The rule used to demonstrating this is D1.

The update of the K knowledge base with $p(f(f(a)))$ is realized by the Upd function which must be applied to the knowledge base K.

$$Upd(K, p(f(f(a)))) = K \cup \{p(f(f(a)))\} = K_1.$$

$K_1 = \{p(a), q(a,a), p(f(a)), q(f(a), a), p(f(f(a)))\}$ ([2]).

5. A KRPS to play the role of a Data Base Management System

5.1 Description of the Model

The following definitions describe the KRPS to store data about the products in a shop.

$L_{\mathrm{Re}\,pr} = (code, name, cantity, price, pieces)$.

$L_{KB} = \{K \mid (x, y_1, z_1, t_1, u_1) \in K, (x, y_2, z_2, t_2, u_2) \in K \Rightarrow y_1 = y_2, z_1 = z_2, t_1 = t_2, u_1 = u_2\}$

$L_Q = \{(x, q) \mid q \in listall, listprice, listpieces, totalvalue\}$.

$Ans : L_{KB} \times L_Q \to L_{Ans}$

$$Ans(K,(x,q)) = \begin{cases} q = listall \Rightarrow (x,y,z,t,u) \mid (x,y,z,t,u) \in K, K \vdash (x,q), \\ q = listprice \Rightarrow t \mid (x,\_,\_,t,\_) \in K, K \vdash (x,q), \\ q = listpieces \Rightarrow \sum_{(x,\_,\_,\_,u) \in K} u, K \vdash (x,q), \\ q = totalvalue \Rightarrow \sum_{(x,\_,\_,t,u) \in K} tu, K \vdash (x,q), \\ no, K \vdash -(x,q). \end{cases}$$

$L_{Ans} = \{N \cup R \cup Str \cup \{no\}\}$

$L_I = \{((x, y, z, t, u), op) \mid op \in \{add, ch, del\}\}$

$Upd : L_{KB} \times L_I \to L_{KB}$

$$Upd(K,(x,y,z,t,u),op) = \begin{cases} op = add \quad and \\ \quad 1)\ \exists (x,\_,\_,p,\_) \in K, t \neq p \Rightarrow K = K \cup \{(x,y,,z,t,u)\} \\ \quad 2)\ \exists (x,\_,\_,\_,\_) \notin K \Rightarrow K = K \cup \{(x,y,z,t,u)\} \\ \quad 3)\ \exists (x,y_1,z_1,p,u_1) \in K,\ t = p, \Rightarrow K = (K \setminus \{(x,y_1,z_1,p,u_1)\}) \cup \{(x,y,z,p,u+u_1)\} \\ op = ch \ \ and \ \ \exists (x,y,z,t_1,u) \in K \Rightarrow K = (K \setminus \{(x,y,z,t_1,u)\}) \cup \{(x,y,z,t,u)\} \\ op = del \ and \ \ \exists (x,y,z,t,u) \in K \Rightarrow K = K \setminus \{(x,y,z,t,u)\} \\ op = chL \ \ and \ \ \exists (x,y,z,t,\alpha L) \in K \Rightarrow \\ \qquad \Rightarrow K = (K \setminus \{(x,y,z,t,\alpha L)\}) \cup \{(x,y,z+20000,t,\alpha L)\}. \end{cases}$$

This reasoning is monotone, reflexive and idempotent([2]).

5.2 Experimental results

This application was developed using Visual C++. It needs an additional file to store data. The data are represented in the following table.

Code	Pieces	Price	Name	Cantity
1	50	6000	Paine PanGroup	380g
2	12	105000	Alexandrion *******	0.5L
3	200	6000	Eugenia	20g
4	20	12000	Mustar Dulce Galati	400g
5	20	39000	Fanta De Padure	2L
6	3	20000	Carti de Joc	pac
7	60	28500	Viceroy Albastru	pac

First picture shows the system answer at the question $Ans\,(K,(3,listprice\ ))$.

Code	Pieces	Price	Name	Cantity
3	200	6000	Eugenia	20g

Next picture shows the system answer at the question $Ans(K,(5,listpieces))$.

Code	Pieces	Price	Name	Cantity
5	20	39000	Fanta De Padure	2L

The following pictures show the new K base after applying the update $Upd(K,(1,\_,\_,\_,\_),del)$,

Code	Pieces	Price	Name	Cantity
2	12	105000	Alexandrion *******	0.5L
3	200	6000	Eugenia	20g
4	20	12000	Mustar Dulce Galati	400g
5	20	39000	Fanta De Padure	2L
6	3	20000	Carti de Joc	pac
7	60	28500	Viceroy Albastru	pac

and the update $Upd(K,(\_,\_,\_,\_,wL),chL)$ which increases by 20.000 the price of all products which are liquids.

Code	Pieces	Price	Name	Cantity
2	12	125000	Alexandrion *******	0.5L
3	200	6000	Eugenia	20g
4	20	12000	Mustar Dulce Galati	400g
5	20	59000	Fanta De Padure	2L
6	3	20000	Carti de Joc	pac
7	60	28500	Viceroy Albastru	pac

6. Conclusions

KRPS systems are very appropriate to play the role of a Data Base Management System. Moreover, they can encode additional information about the entities, information that cannot be represented by the means of a DBMS but necessary to encode more realistic characteristics of the entities.

7. Acknowledgements

I thank Professor Nicolae Ţăndăreanu because he has taught us the basic concepts in Knowledge Bases and because he has shown so much trust in me and my colleagues and has given me the opportunity to write this article.

I want to say „Thank, you!" to Ruxandra Gorunescu for her seriousness and assiduity in teaching me and my colleagues the notions of Knowledge Bases.

I also thank Cătălin Stoean, for his support and because he showed me the "undergrounds" of the *Prolog* language.

I especially thank my team mate and my best friend, Gabriel Voinescu, because he has supported me when we have worked together on the majority of our projects.

8. References

[1] N. Tăndăreanu: Sisteme expert. Reprezentarea cunoştinţelor şi inferenţa, Editura Universitaria, 2001

[2] Ruxandra Gorunescu, Knowledge Base Seminars, Faculty of Mathematics and Computer Science of Craiova.

[3] Csima Judit Knowledge Processing & Applied Artificial Intelligence
http://www.scism.sbu.ac.uk/inmandw/review/knowacq/review/rev16576.html

9. About me

Pîrlog Orlando-Gabriel, student in the 3rd year at the Faculty of Mathematics and Computer Science of Craiova. I am junior member at the Research Center of Artificial Intelligence (RCAI). My hobbies are: motoring, football, programming and listening to the 80's music. Presently, I am working with my team mates at an application for generating test papers.

Convex Hulls in Three Dimensions

ABSTRACT

The convex hull of a set is the smallest convex set containing this set. The focus of this article is to build an algorithm for constructing the convex hull of a set of three dimensions points. In this article the following concepts are presented: properties of polyhedra, proof and consequence of Euler's formula. The results are applied to the study of convex hull using an incremental algorithm.

Introduction

A subset P of the plane is convex if for every $p, q \in P$ the line segment pq is completely contained in P. The convex hull of a set P, denoted CH(P), is the smallest convex set containing P.

Analogy: if the points are nails on a board, a rubber band (convex hull) encloses all of the nails on the board.

Alternative definition: *CH(P)* is the unique convex polygon whose vertices are points of P that contains all points in P.[2]

A polyhedron is the natural generalization of a two-dimensional polygon to three-dimensions: it is a region of the space whose boundary is composed of a finite number of flat polygonal faces, every two of which are either disjoint or meet at edges and vertices. This description is vague, and it is a surprisingly delicate task to mate it capture just the right class objects. Since the primary concern of this description is convex polyhedra, which are simpler than general ones, a precise definition of them could be avoided. But facing the difficulties helps develop three-dimensional geometric intuition, an invaluable skill for understanding computational geometry.

The boundary or surface of a polyhedron is composed of three types of geometric objects: zero-dimensional vertices (points), one-dimensional edges (segments), and two-dimensional faces (polygons). It is a useful simplification to the demand that the faces are convex polygons. This is no loss of generality since any nonconvex face is coplanar.

What constitutes a valid polyhedral surface can be specified by conditions on how the components relate to one another. We impose three types of conditions: the components intersect "*properly*", the local topology is "*proper*", and the global topology is "*proper*".

2. Fundamental theoretical concepts of a polyhedra

1. Components intersect "*properly*".

   For each pair of faces, we require that either

   (a) they are disjoint, or

   (b) they have a single vertex in common, or

   (c) they have two vertices and the edge joining them in common.

   This is where the assumption that faces are convex simplifies the conditions. Improper intersections include not only penetrating faces, but also faces touching in the "*wrong*" way (see Figure 1). There is no need to specify conditions on the intersection of edges and vertices, as the conditions on faces cover them also. Thus an improper intersection of a pair of edges implies an improper intersection of faces. [1]

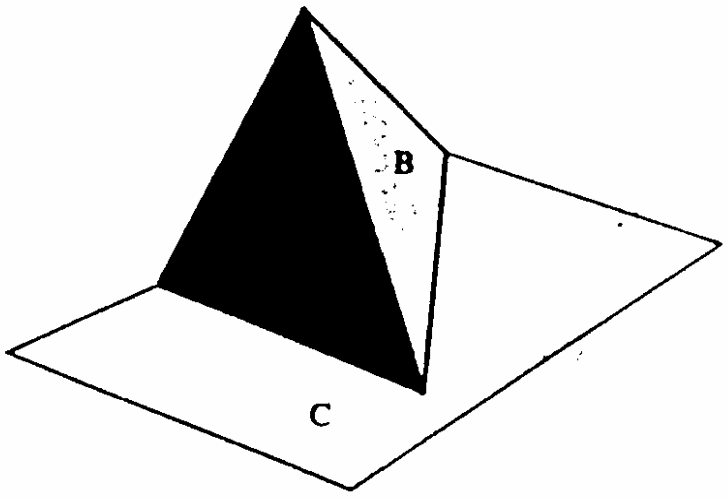


Figure 1

2. Local topology is "*proper*".

   The local topology is what the surface looks like in the vicinity of a point.

   This notion has been made precise via the notation of "*neighborhoods*", i.d. arbitrarily small portions (open regions) of the surface surrounding a point. We seek to

exclude the three objects shown in Figure 2. In all three examples in that figure, there are points that have neighborhoods that are not topologically two-dimensional disks. The technical way to capture the constraint is to require the neighborhoods of every point on the surface to be "*homeomorphic*" to a disk. A homeomorphism between two regions permits stretching and bending, but no tearing. A fly on the surface would find the neighborhood of every point to be topologically like a disk. A surface for which this is true for every point is called 2-monifold, a class more general than the boundaries of polyhedra.

We have expressed the condition geometrically, but it is useful to view it combinatorially also. Suppose we triangulate the polygonal faces. Then every vertex is the apex of a number of triangles. Define the link of a vertex *v* to be a collection of edges opposite to *v* in all the triangles incident to *v*. For a legal triangulated polyhedron, we require that the link of every vertex is a simple, closed polygonal path. The link for the cycled vertex in Figure 2 (b), for example, is not such a path. One consequence of this condition is that every edge is shared by exactly two faces. [4]

3. Global topology is "*proper*".

The surface is intended to be connected, closed, and bounded. Thus, it is required that the surface is connected in the sense that from any point one may walk to any other on the surface. This can be stated combinatorially by requiring that the 1-skeleton, the graph of edges and vertices are connected. Note that this excludes, for instance, a cube with a "*floating*" internal cubical cavity. Together with stipulating a finite number of faces, our previous conditions already imply closeness and boundness of these faces, although this is perhaps not self-evident.

One might be inclined to rule out "*holes*" in the definition of the polyhedron, holes in the sense of "*channels*" from one side of the surface to the other that do not disconnect the exterior (unlike cavities). The usual terminology is adopted and permit polyhedra are permitted to have an arbitrary number of such holes. The number of holes is called the *genus* of the surface. Normally only polyhedra with genus zero: will bet considered i.d. hose topologically equivalent to the surface et a sphere.[3]

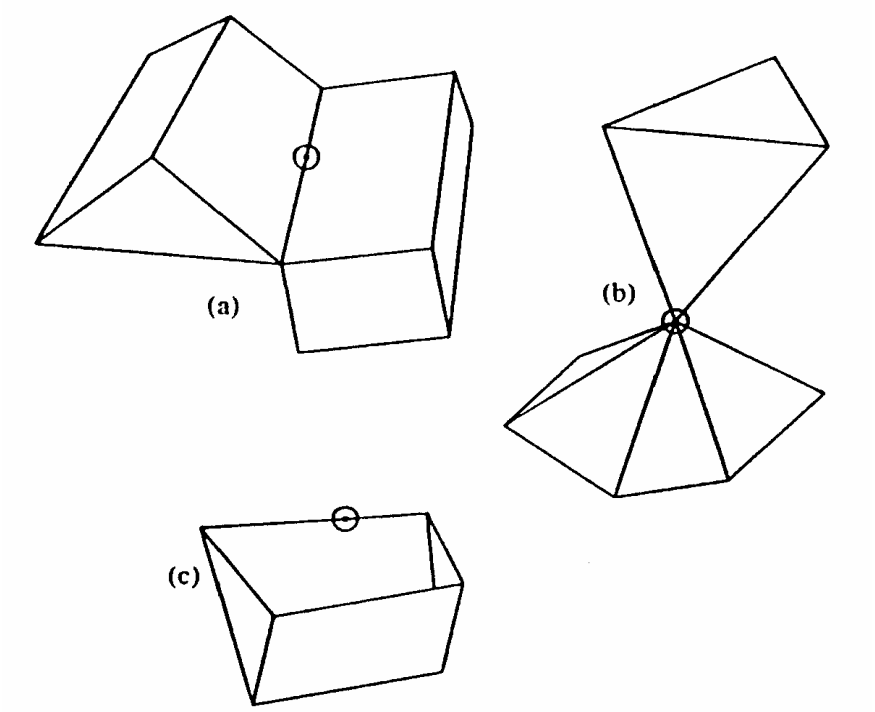


Figure 2

In summary, the boundary of a polyhedron is a finite collection of planar, bounded convex polygonal faces such that.

1. the faces intersect properly;
2. the neighborhood of every point is topologically an open disk, or (equivalently) the link of every vertex is a simple polygonal chain; and
3. the surface is connected, or (equivalently) the 1-skeleton is connected.

The boundary is closed and encloses a bounded region of space. Every edge is shared by exactly two faces; these faces are called *adjacent*.

Convex polyhedra are called *polytopes*, or sometimes 3-polytopes to emphasize their three-dimensionality. A polytope is a polyhedron that is convex on that the segment connecting any two of its points inside. Convex polygons can be characterized by the local requirement that each vertex be convex, polytopes can be specified locally by requiring that all *dihedral* angles be convex($\leq \pi$). Dihedral angles are the internal angles in space at an edge between the planes containing its two incident faces. For any polytope, the sum of the face angles around each vertex is at most 2π, but this condition does not alone imply convexity.

3. Euler's Formula

In 1758 Leonard Euler noticed a remarkable regularity in the numbers of vertices, edges, and faces of a polyhedron of genus zero: the number of vertices and faces together is always by two more than the number of edges; and this is true for all polyhedra. So a cube has 8 vertices and 6 faces, and 8+6=14 is two more than its 12 edges. And the remaining regular polytopes can be seen to satisfy the same relationship. If we let V, E *and* F *be the number of vertices, edges and faces respectively of a polyhedron, then, what is now known as Euler's formula is:* V-E+F=2. *[1]*

Proof of Euler's Formula is comprised of three parts:

1. Converting the polyhedron surface to a plane graph.

2. The theorem for trees.

3. Proof by induction.

Firstly, the polyhedron is "*flattened*" on surface *P* to be a plane, perhaps with considerable distortion by the following procedure. Imagine the surface is made of a pliable material. Choose an arbitrary face *f* of *P* and remove it, leaving a hole in the surface. Now stretch the hole wider and wider until it becomes much larger than the original size of *P*. It should be intuitively plausible that one can then flatten the surface onto the plane, resulting in a plane graph *G* (the 1-skeleton of the polytope): a graph embedded in the plane without edge crossings, whose nodes derive from vertices of *P*, and whose arcs derive from edges of P. The edges of f become the outer boundary of *G*. Each face of *P* except for *f* becomes a bounded face of *G*; *f* becomes the exterior, unbounded face of *G*.

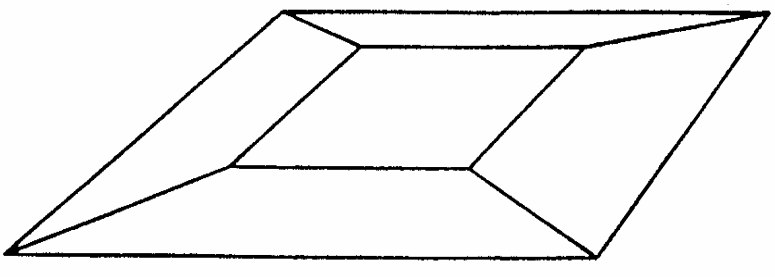


Figure 3 The 1-skeleton of a cube, obtained by flattening to a plane

Figure 3 illustrates the graph that results from flattening a cube. Thus if we count this exterior face of G as a true face (which is the usual convention), then the vertices, edges, and faces of P are in one-to-one correspondence with those of G. This permits us to concentrate on proving Euler's formula for plane graphs.

The second step is to prove the formula in the highly restricted case where G is a tree. Of course a tree could never result from stretching a polyhedron, but this is a useful tool for the final step of the proof. So suppose G is a tree of V vertices and E edges. It is a property of trees that $V=E+1$, a fact that is assumed for the proof. A tree bounds or delimits only one face, the exterior face, so $F=1$. Now Euler's formula is immediate: $V-E+F=(E+1)-E+1=2$.

The third and final step of the proof is by induction on the number of edges. Suppose Euler's formula is true for all connected graphs with no more than $E-1$ edges, and let G be a graph of V, E, and F vertices, edges, and faces respectively. So suppose G has a cycle, and let e be an edge of G in some cycle. The graph $G'=G\backslash e$ is connected, with V vertices, $E-1$ edges, and (here is the crux) $F-1$ faces: removal of e must join two faces into one. By the introduction hypothesis, $V-(E-1)+(F-1)=2=V-E+F$. [1]

Consequence: Linearity

Euler's formula implies that the number of vertices, edges, and faces of a polytope are linearly related: if $V=n$, then $E=O(n)$ and $F=O(n)$. This will permit to use "n" rather loosely in complexity analyses involving polyedra.

Because we seek to establish an upper bound on E and F as a function of $V=n$, it is safe to triangulate every face of the polytope, for this will only increase E and F without affecting V. So for the remainder of this argument let it be assumed that all the faces of the polytope are triangles. If the edges face by face are counted, then $3F$ is obtained because each face has three edges. But since each edge is shared by two faces, this double-counts the edges. So $3F=2E$. Now substitution into Euler's formula establishes the linear bounds:

$$V-E+F=2$$

$$V-E+2E/3=2$$

V-2=E/3

E=3V-6<3V=3n=O(n)

F=2E/3=2V-4<2V=2n=O(n)

Theorem: For a polyhedron with *V=n*, *E*, and *F* vertices, edges, and faces respectively, *V-E+F=2*, and both *E* and *F* are *O(n)*. [3]

4. Incremental algorithm in two-dimensions

Input: $\{P_1,...,P_n\} \subseteq R^2$ such that no three are collinear; let *P* be a random permutation of the input points.

Step 0: Randomly extract 3 points from *P* and do the following:

a) Form a triangle *T*.

b) Pick a point *c* interior to the triangle *T*.

Step 1: Draw an arc from *c* to each point in *P*.

Step 2: Partition the points in *P* by the edge of *T* (the triangle) which their arc crosses (a point is actually marked as dead if it is in the interior of this polygon; otherwise the point is considered alive).

Step 3: While there is a point above some edge in the current *CH* do

a) Pick a random point *p* above, say, edge *e*.

b) Build-Tent *(p, e, Poly)*

Output: The convex hull of *P*.

5. Incremental algorithm in three-dimensions

The overall structure of the three-dimensional incremental algorithm is identical to that of the two-dimensional version. At the i^{th} iteration, compute $H_i \leftarrow \text{conv}(H_{i-1} \text{ Y } p_i)$. And again the problem of computing the new hull naturally divides into two cases. Let $p = p_i$ and $Q = H_{i-1}$. Decide if $p \in Q$. If so, discard *p*; if not, compute the cone tangent to *Q* whose apex is p, and construct the new hull.

The test $p \in Q$ can be made in the same fashion as two dimensions: p is inside Q if p is to the positive side of every plane determined by a face of Q. The left-of-triangle test is based on the volume of the determined tetrahedron, just as the left-of-segment test is based on the area of triangle. If all faces are oriented consistently, the volumes must all have the same sign (positive under the conventions considered in this article). This test can clearly be accomplished in time proportional to the number of faces of Q that is O(n).

When p is outside Q, the problem becomes more difficult, as the hull will be altered. Recall that in the two-dimensional incremental algorithm, the alteration required finding two tangents from p to Q. In three dimensions, there are tangent planes rather than tangent lines. These planes bound a *cone* of triangle faces, each of whose apex is p, and whose base is an edge e of Q. An example is shown in Figures 4 and 5. Figure 4 shows H_{i-1} and H_i from one point of view, and Figure 5 shows the same example from a different viewpoint.

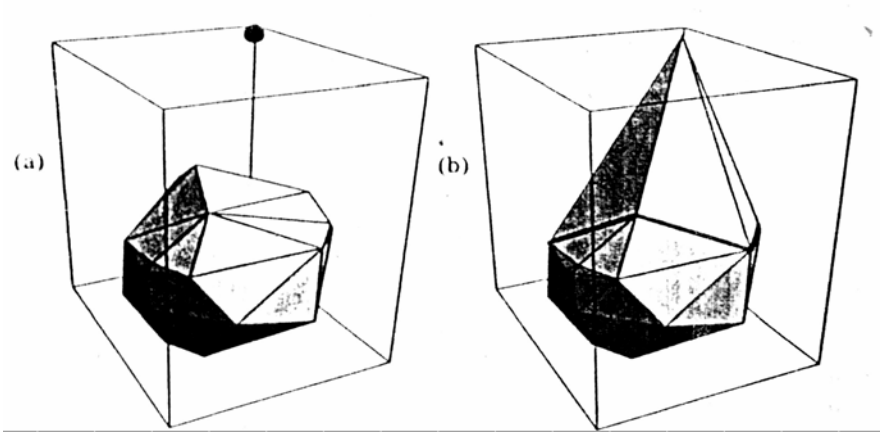


Figure 4 Viewpoint one: (a) H_{i-1} *before adding a point in corner*

(b) after: H_i

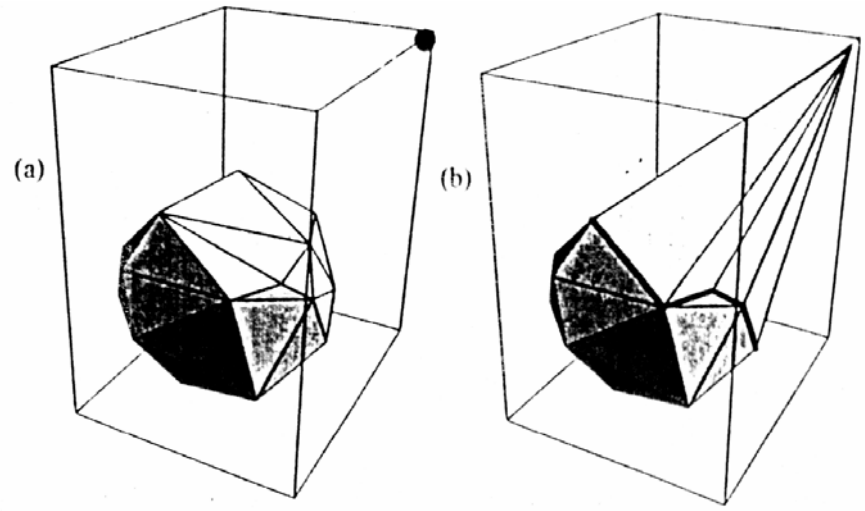


Figure 5 Viewpoint two : (a) H_{i-1} *before adding a point in corner*

(b) after: H_i

Imagine staying in point *p* and looking toward *Q*. Assuming for the moment than no faces are viewed edge-on, the interior of each face of *Q* is either visible or not visible from *p*. It should be clear that the visible faces are precisely those that are to be discarded in moving from $Q = H_{i-1}$ to H_i. Moreover, the edges on the border of the visible region are precisely those that become the bases of cone faces apexed at *p*. Suppose *e* is an edge of *Q* such that the plane determined by *e* and *p* is tangent to *Q*.

6. Implementation remarks:

The application is written in C. To memorize the information about faces, edges and vertex three structures are used. At the beginning it has two options are given. Example: Cube and Random points. If is the first option chosen is created the convex hull of the following points:

0 0 0

0 50 0

50 50 0

50 0 0

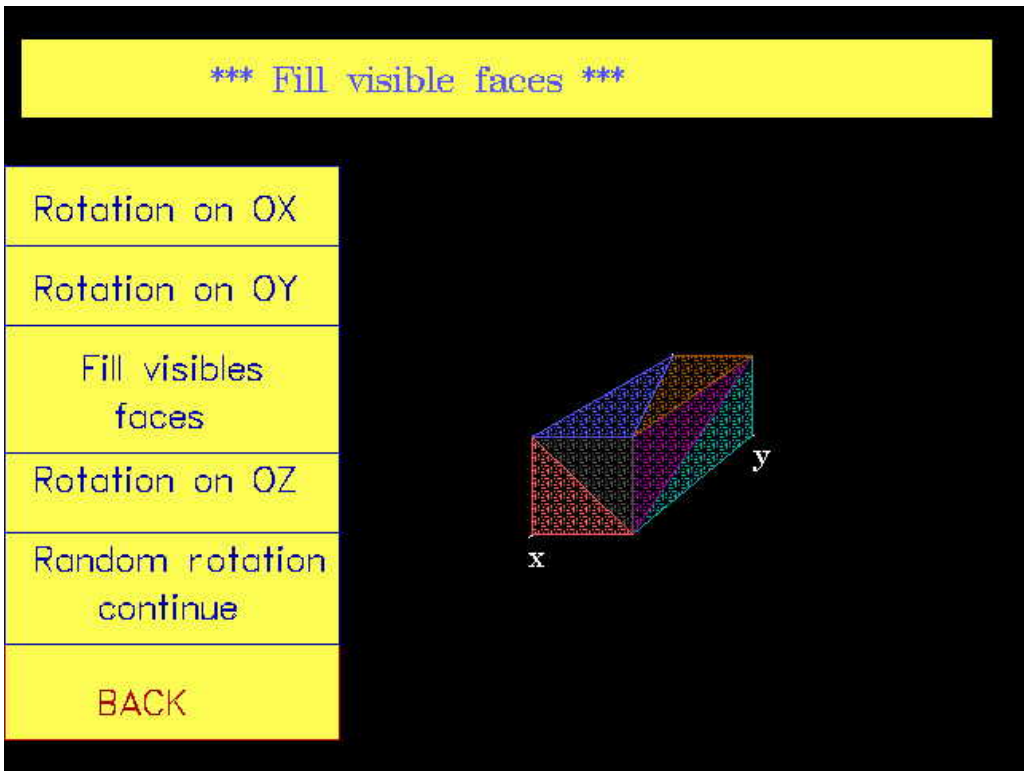
0	0	50
0	50	50
50	50	50
50	0	50

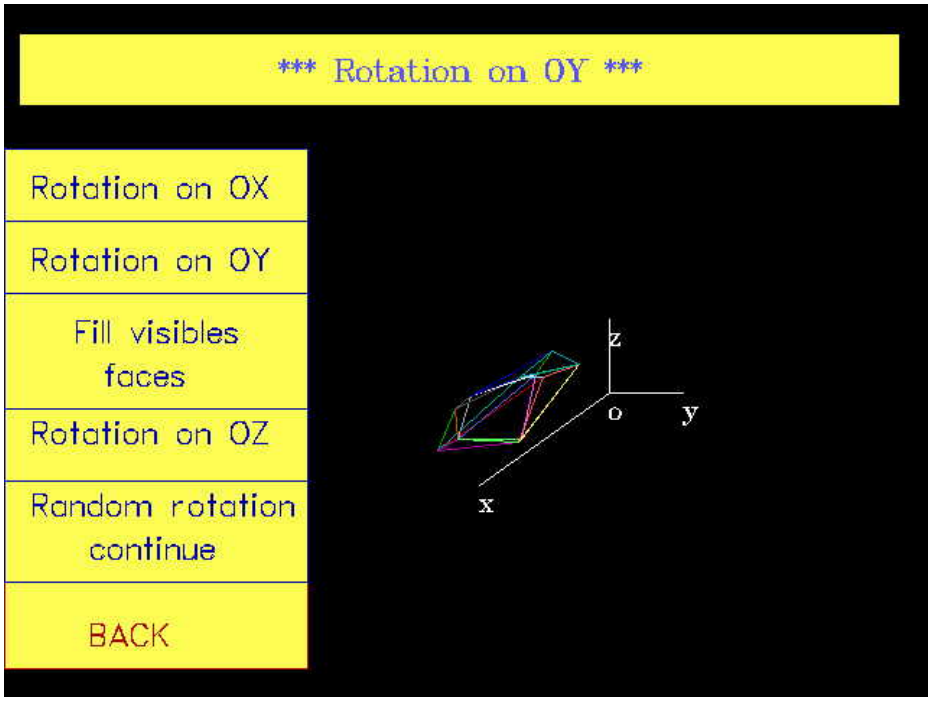
These points represent indeed the coordinates of the vertex of a cube.

If the second option is chosen the application asks for a number of points. For each point it will generate aleatory the three coordinates (between 0 and 50) and then it will pass to form the convex hull for them.

The resulted drawing can be seen using different projection formulas and the three axes: OX, OY, OZ.

7. Experimental results:





8. Conclusions and future work

The algorithm presented is one of the most quickly algorithms $O(n^2)$. An algorithm which is better has O(n logn) and is based on the divide-and-conquer method (Preparata and Hong in 1977). A challenge is to make a convex hull in four dimensions where the fourth dimension can be time, or in more dimensions. E.g.: The key sartorial characteristic of a person could be represented by height, sleeve length, inseam length, neck and waist circumferences. Then each person could be viewed as a point in a five-dimensional space: height, arm, leg, neck, waist.

9. Acknowledgements

I want to express my thanks to all the teachers that guided me along the time (from general school to faculty) but especially to my teacher of informatics from high school, Mrs. Dobre Carmen and to my Professor of computational geometry, Mrs. Mihaela Sterpu, which guided and helped me in making this article.

10. References

[1] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Communications of the ACM*, vol. 20, pp. 87-93, 1977.

[2] Nancy Amato (Texas A&M Department of Computer Science) http://parasol.tamu.edu/~amato/Courses/620/ Scribe/Intro/ConvexHulls.ppt

[3] O. Devillers and M. Golin. Incremental Algorithms for Finding the Convex Hulls of Circles and the Lower Envelopes of Parabolas. Inform. Process. Lett., 56(3):157--164, 1995. http://www-sop.inria.fr/prisme/publis/dg-iafch-95.ps.gz

[4] Scot Drysdale: Notes for lecture 7: Convexity
http://cm.bell-labs.com/who/clarkson/cis677/lecture/7/

Dobrin Mihai, student in the 3rd year at the Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova. Recently, I was appointed junior member at the Research Center for Artificial Intelligence (RCAI). My hobbies are: computer (especially programming), beekeeping, classical music. At this moment I am working on a program for topography at a firm in Craiova. I spend my spare time as a a scout at the Rabon-Cfr group or… between the bees…beekeeping them!…