Anul I 1996

STUDIA
UNIVERSITATIS "BABEŞ-BOLYAI"

INFORMATICA
2

Redacţia: 3400 Cluj-Napoca, str. M. Kogalniceanu nr. 1 • Telefon: 194315

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

A GRAPH MODEL OF DISTRIBUTED DATABASE SYSTEMS

ZOLTÁN KÁSA AND VIORICA VARGA

Abstract. In this paper we deal with a graph model of the distributed database systems, in which the nearest sites with some given properties can be easily found.

1. Introduction

A distributed database system consists of a collection of sites connected in a network by some kind of communication lines, in which each site is a properly database system, but a user at any site can access data at any other sites [2, 3]. By point of view of any user the system must look as a nondistributed one. There are several reasons why, for a distributed system to be successful it must be relational.

A system supports data fragmentation if a given stored relation can be divided up in pieces (so-called fragments) for physical storage purposes. Fragmentation is for performance reasons. In such a case data can be stored at the locations where they are most frequently used, operations are local, network traffic reduced.

There are two kind of fragmentations: horizontal and vertical. Let $\mathcal{R}[A_1, A_2, \cdots, A_n]$ be a relation, where $A_i, i = \overline{1,n}$ are attributes. A horizontal fragment is obtained by a restriction: $\mathcal{R}_i = \sigma_{cond}(\mathcal{R})$, where *cond* is the guard condition. The initial relation can be reconstructed by union of the horizontal fragments: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_m$.

A vertical fragment is obtained by a projection operation:

$$\mathcal{R}_j = \prod_{\{A_{k_1}, A_{k_2}, \cdots, A_{k_p}\}} (\mathcal{R}),$$

Received by the editors: January 27, 1997.

1991 *Mathematics Subject Classification.* 68P15, 68R10, 68M10.

1991 *CR Categories and Descriptors.* C.2.0 [Computer-Communication Networks]: General – data communications; C.2.1 [Computer-Communication Networks]: Network Architecture and Design – distributed networks; G.2.2 [Discrete Mathematics]: Graph Theory – graph algorithms, breadth-first search.

where $A_{k_l}, l = \overline{1,p}$ are attributes. Such a fragmentation must be nonloss, the initial relation would be reconstructed by join of vertical fragments: $\mathcal{R} = \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \cdots \otimes \mathcal{R}_q$.

A system with fragmentation should also support fragmentation independence, i.e. users should be able to work as in the case of nonfragmented data. It is the of the system optimizer to determine which fragments need to be physically accessed in order to satisfy any given user query.

A system supports data replication if a given stored relation or a fragment can be represented by many distinct copies or replicas, stored at many distinct sites. The main advantage of data replication resides in local operations on data. The major disadvantage is that updated replicated data must be updated at all sites where there exist copies. Replications must be transparent to the user, and the system optimizer is responsible for choose the nearest sites to satisfy a user request.

In a distributed system the query optimization is more important than in a centralized one. In a query which involve several sites, there will be many possible ways to access the corresponding data, and the strategy which is used is crucial. Strategies can be easily modeled by graphs.

2. The Graph Model

We consider a graph $\mathcal{G} = (V, E)$ associated with a distributed database system: the sites are represented by vertices (V) and direct connections between sites by corresponding edges (E). The data stored in the system are characterized by some properties, the same data by same properties anywhere they are. The same data can be found at many sites.

Problem 2.1. *For a given vertex i find the nearest sites which satisfy a given property.*

The corresponding graph problem is:

Problem 2.2. *Let given a graph, in which each vertex is characterized by some properties. For a given vertex find the nearest vertex which satisfy a given property.*

Let us denote by $\Gamma(x)$ the set of all adjacent vertices of x (vertices which are connected by an edge to x).

The following breadth-first algorithm [1] determines the nearest vertex from i with the given property P. When a vertex is visited it is marked and inserted in an initially empty queue Q. At the beginning all vertices are unmarked.

The procedure *Visit (x)* visits the unmarked vertex x, marks it, tests if P is true for it, if it is, then exits, if it is not then insert x in the queue Q:

procedure Visit (x):
  if P is true for x **then** exit

```
                else mark x
                     insert x in Q
    end if
end procedure
```

Problem 2.1 is resolved by the following algorithm:

```
procedure Nearest (i):
    Visit (i)
    while Q is nonempty do
        let x be the removed front element of Q
        for all unmarked j ∈ Γ(x) do Visit (j)
    end while
end procedure
```

Let us denote by $\mathcal{P} = (P_1, P_2, \cdots, P_n)$ the set of properties characterizing the data in the system, and by m the number of sites ($m = |V|$). Let us consider the matrix

$$\mathrm{d} = (d_{ij}) \quad \begin{matrix} i = \overline{1,m} \\ j = \overline{1,n} \end{matrix}$$

in which $d_{i,j} - 1$ represents the minimal distance from the site i to the site for which the property P_j is true, if $d_{ij} > 0$. If there is no sites with property P_j, reachable from the site i, then $d_{ij} = 0$.

Problem 2.3. *For a given vertex i find the minimal distances to vertices which satisfy the properties P_j.*

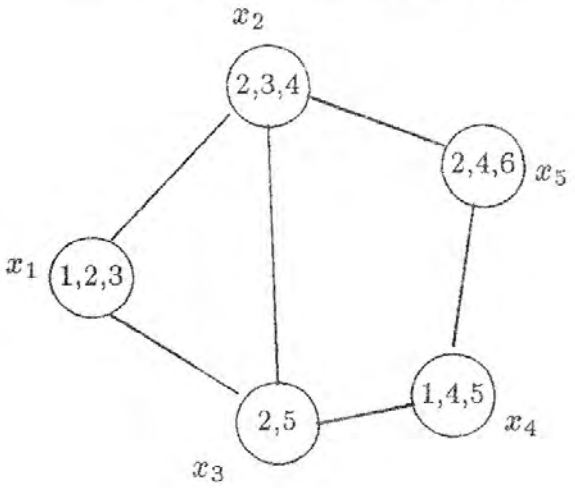
Procedure *Visit2 (i)* visits the unmarked vertex i, marks it, and assign to d_{ij} the correspondent distance k if P_j is true for i. To increase correctly the value of k we will use two queues.

```
procedure Visit2 (i):
    for j = 1, 2, ··· , n do
        if d_{i,j} = 0 and P_j is true for i then d_{i,j} := k end if
    end for
    mark i
    insert i in Q_1
end procedure
```

The algorithm which resolves the problem 2.3 is the following:

procedure Distance (i):
  for $j := 1, 2, \cdots, n$ **do** $d_{i,j} := 0$ **end for**
  Initialize the queue Q_1
  k := 1; Visit2 (i); k := 2
  Move Q_1 to Q_2
  Repeat
    Initialize Q_1
    while Q_2 is nonempty **do**
      let x be the removed front element of Q_2
      for all unmarked $y \in \Gamma(x)$ **do** Visit2 (y) **end for**
    end while
    Move Q_1 to Q_2; k:=k+1
  until Q_1 becomes empty
end procedure

Using the matrix **d** we can characterize the entire system. Let us denote by $\kappa(k)$ the number of copies of the data characterized by P_k, this is the number of 1's in the k^{th} column of **d** (see the example).



	P_1	P_2	P_3	P_4	P_5	P_6
x_1	1	1	1	2	2	3
x_2	2	1	1	1	2	2
x_3	2	1	2	2	1	3
x_4	1	2	3	1	1	2
x_5	2	1	2	1	2	1
$\kappa(k)$	2	4	2	3	2	1

$\kappa = \frac{14}{6} = 2.33\ldots$

We can define the average value of the copies:

$$\kappa = \frac{\sum_{k=1}^{n} \kappa(k)}{n}.$$

This value can vary between 0 and $m = |V|$. The case $1 \leq \kappa \leq m$ is interesting only. If $\kappa < 1$, then there exists at least one k for which $\kappa(k) = 0$, i.e. the corresponding property P_k is false for all sites. If $\kappa = 1$ each data in the system exists only at a single site, but if $\kappa = |V|$ each data exists at all sites.

If $\kappa(k) = 0$ for a given k, then the corresponding column (the k^{th} column) in the matrix **d** must be identically equal to 0.

If in the matrix **d** there exists an element equal to 0, but not the entirely column, the corresponding graph is disconnected.

Problem 2.4. *Having the distance matrix* **d**, *find a path from the site* i *to the site for which* P_j *is true.*

If $d_{i,j} = 0$ then the target site cannot be reached, the graph is disconnected. If $d_{i,j} = 1$ then the target site is identical to the source one. The following algorithm is given for $d_{i,j} > 1$, and finds a the possible path with the given property.

```
procedure Path (i, j):
    mark i
    k := d_{i,j} − 1
    while k > 0 do
        for all unmarked y ∈ Γ(i) do
            mark y
            if d(y, j) = k then i := y
                                memorize y
                                k := k − 1
                                exit the loop for
            end if
        end for
    end while
end procedure
```

If in the distributed system a direct connection between two sites is broken, we must recompute the distance matrix. This can be made easily by the above algorithms.

3. Conclusions

Breadth-first search algorithms are well-known in graph theory. We have presented a breadth-first search algorithm to find data characterized by some conditions in a distributed database system. This algorithm can be used in a dynamic manner i.e. after any modification in the structure of the system (lost connections, new connections) a simple recalculation of the distance matrix ensures the maintenance of the system.

References

[1] S. Baase, *Computer algorithms: Introduction to design and analysis*, Addison-Wesley Publishing Co., 1983.

[2] C. J. Date, *An introduction to database systems*, Addison-Wesley Publishing Co., 1995.

[3] M. T. Özsu, P. Valduriez, *Principles of distributed database systems*, Prentice Hall, 1991.

Babeş-Bolyai University, Faculty of Mathematics and Informatics,
RO 3400 Cluj-Napoca, str. Kogălniceanu 1, Romania
E-mail address: {kasa,ivarga}@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

DATA COLLECTIONS AND MONADS

RADU TRÎMBIŢAŞ

Dedicated to Professors Emil Muntean and Ştefan Niţchi

Abstract. This paper tries to introduce data collection using monads. Three data collection types are defined: weak, zero-element and strong. The main result shows that monads lead to data collection types. Then the algebraic properties of data collections are studied and some identities useful to query optimization are given. Another result shows the equivalence of ringad and extended monad with zero. Finally, we treat the type conversion problem.

1. Collections

Data collections are defined in [1]. We shall use other definitions for data collections, and our approach will be categorical. For notions of Category Theory the reader may consult [3, 4, 11]. Also the paper [8] illustrates how Category Theory notions and constructions can be systematically used in Computer Science.

Definition 1.1. *A* **weak data collection** C *is a parametrized abstract data type (with type parameter* T*)*

$$C(T) = < \tau, [x], ++, \text{aggr} >$$

where

- τ *is the set of data collections over* T*;*
- $[x]$ *is the singleton (single element) collection;*
- $++$*:* $\tau \times \tau \to \tau$ *is the concatenation operator of two collections;*
- aggr *is the aggregation operator defined as follows: if* f*:* $T \to S$ *and* $\oplus : S \times S \to S$ *is a binary operation, then* $\text{aggr}(f, \oplus) : \tau \to S$ *(i.e.* $\text{aggr} : (T \to S) \times \tau \to S$*) which has the following properties:*

(1) $$\text{aggr}(f, \oplus)([x]) = f(x)$$

Received by the editors: October 15, 1996.

1991 *Mathematics Subject Classification.* 18C15, 68P15.

1991 *CR Categories and Descriptors.* H2.1 [Data base management] Logical Design – data models, H2.4 [Data base management] Systems – query processing.

(2) $$\text{aggr}(f,\oplus)(x{+}{+}y) \;=\; \text{aggr}(f,\oplus)(x)\oplus aggr(f,\oplus)(y), \quad x,y\in\tau.$$

Definition 1.2. *A* **zero-element data collection** *C is a parametrized abstract data type (with type parameter T)*

$$C(T) = <\ \tau,\ [\,],\ [x],\ {+}{+},\ \text{aggr}\ >$$

where

- $<\ \tau,\ [x],\ {+}{+},\ \text{aggr}\ >$ *is a weak data collection;*
- *$[\,]$ is the empty collection and*

(3) $$\forall x\in\tau \qquad x{+}{+}[\,]=[\,]{+}{+}x=x.$$

- aggr *is the aggregation operator defined as follows: if $f : T\to S$ and $\oplus : S\times S\to S$ is a binary operation with neutral element u, then $\text{aggr}(f,\oplus):\tau\to S$ (i.e. $\text{aggr}:(T\to S)\times\tau\to S$) which verifies (1), (2) and*

(4) $$\text{aggr}(f,\oplus)([\,]) \;=\; u.$$

Definition 1.3. *A* **strong data collection** *is a zero-element data collection such that the concatenation is associative, that is*

(5) $$\forall x,y,z\in\tau \qquad (x{+}{+}y){+}{+}z=x{+}{+}(y{+}{+}z).$$

The notion of strong collection is similar to monoid collection defined in [5, 6, 7], but there the approach is not categorical.

Remark 1.4. Each strong data collection is also a zero-element data collection; each zero-element data collection is also a weak data collection.

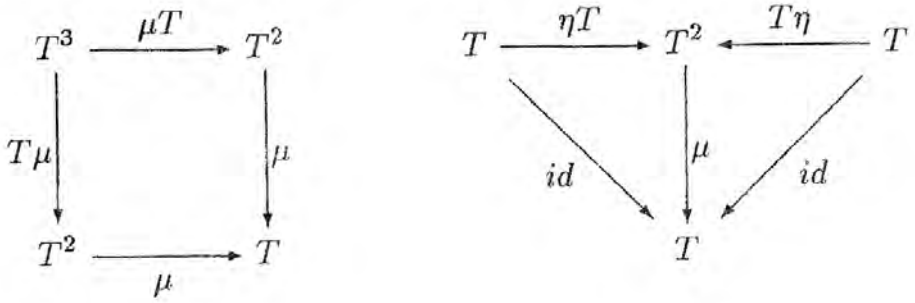
In the sequel $[x_1,\ \ldots,\ x_n]$ will denote the finite collection having the elements $x_1,\ldots x_n$.

2. Monads and data collection

Definition of data types through monads is a usual method in Functional Programming. As each adjunction induces a monad (Huber's theorem see [3, 4, 10]) and each monad leads to adjunctions in several ways, Eilenberg-Moore's and Kleisli's construction being the most known ([3, 4, 10]), the results must resemble those obtained through adjunctions.

Definition 2.1. *A* **monad** *(or* **triple***)* $\mathbf{T}=(T,\eta,\mu)$ *on category $\mathcal{C}$ is an endofunctor $T : \mathcal{C}\to\mathcal{C}$ together with two natural transformations $\eta : id_{\mathcal{C}}\to T$ and*

$\mu : T^2 \to T$ *such that the following diagrams commute:*



(mon1) *(mon2)*

In these diagrams T^n *means* T *composed by himself* n *times.* ■

For examples and properties of monads see [3, 10] and for applications of monads in Computer Science see [12, 13, 14, 15].

Another equivalent definition [9, 14] is:

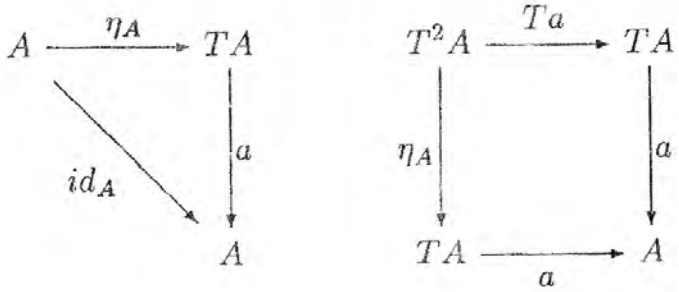
Remark 2.2. A **monad** on $\mathcal{C}$ is the system $(T, \eta, -^*)$ where T and η are as above, and * is the iteration operator (the Kleisli star) defined as follows: if $f : A \to TB$ then $f^* : TA \to TB$ and it satisfies

$$\begin{aligned} (\eta_S)^* &= id_{TA} \\ f^* \circ \eta_A &= f \\ g^* \circ f^* &= (g^* \circ f)^* . \end{aligned}$$ ■

Let's see how data collections, aggregation operators and their properties may be obtained through monads, using the Eilenberg-Moore's construction.

Definition 2.3. *If* $\mathbf{T}=(T, \eta, \mu)$ *is a monad on* $\mathcal{C}$*, a* $\mathbf{T}$*-algebra is a pair* (A, a)*, where* $A \in Ob\mathcal{C}$ *and* $a : TA \to A$ *such that the following diagrams commute:*

(6)



The arrow a *is the structure-mapping of the algebra.* ■

Definition 2.4. *A mapping* $f : (A, a) \to (B, b)$ *is a* **T**-*algebra homomorphism if the diagram*

(7)

$$\begin{array}{ccc} TA & \xrightarrow{Tf} & TB \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle b} \\ A & \xrightarrow[f]{} & B \end{array}$$

commutes. ■

T-algebra and **T**-algebra homomorphisms form a category $\mathcal{C}^T$. We define $U^T : \mathcal{C}^T \to \mathcal{C}$ by $U^T(A, a) = A$ and $U^T(f) = f$ and $F^T : \mathcal{C} \to \mathcal{C}^T$ by $F^T(a) = (TA, \mu_A)$ and $F^T(f) = T(f)$. The functor F^T is the left adjoint of U^T, and the unit and counit of the adjunction are $\eta^T = \eta$ and $\varepsilon^T(A, a) = a : F^T U^T(A, a) \to (A, a)$ respectively (the Eilenberg-Moore's construction). We shall consider only monads over Cartesian closed categories and $\mathcal{C}$ will be $\mathcal{T}$ (the category of types).

Definition 2.5. *The quadruple* **T**$=(T, \eta, \mu, conc)$ *is an* **extended monad** *on* $\mathcal{C}$ *if:*

1. (T, η, μ) *is a monad on* $\mathcal{C}$*;*
2. $conc : T \times T \to T$ *is a natural transformation;*
3. $\forall B \in Ob\mathcal{C}$ $\mu_B \circ conc_B = conc_{TB} \circ (\mu_B \times \mu_B)$*, that is the following diagram*

(8)

$$\begin{array}{ccc} T^2B \times T^2B & \xrightarrow{conc_{TB}} & T^2B \\ {\scriptstyle \mu_B \times \mu_B}\downarrow & & \downarrow{\scriptstyle \mu_B} \\ TB \times TB & \xrightarrow[conc_B]{} & TB \end{array}$$

commutes. ■

Proposition 2.6. For each monad **T**$=(T, \eta, \mu)$ and for each natural transformation $conc : T \times T \to T$ it holds

$$conc_B = \mu_B \circ conc_{TB} \circ (T\eta_B \times T\eta_B).$$

Proof. Using naturality of conc and monad definition we have

$$\mu_B \circ conc_{TB} \circ (T\eta_B \times T\eta_B) = \mu_B \circ T\eta_B \circ conc_B = conc_B. \blacksquare$$

Theorem 2.7. *The extended monad* **T**$=(T, \eta, \mu, conc)$ *determines a weak data collection.*

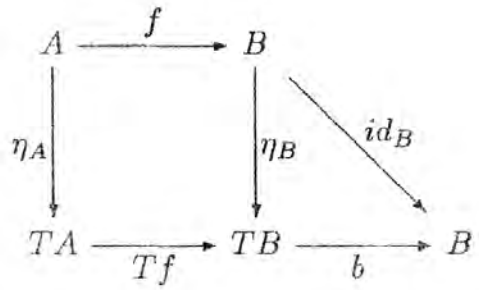
Proof. For a type X we build

$$C(X) = < \tau, [x], ++, \text{aggr} >$$

such that τ is $T(X)$, $[x]$ is $\eta_X(x)$ and $++$ is *conc*. The aggregation operator is defined as follows: let $f : A \to B$ where (B,b) is a **T**-algebra. If $\alpha, \beta \in B$ we define $\alpha \oplus \beta = b[\alpha, \beta]$, where

$$[\alpha, \beta] = conc_B([\alpha], [\beta]) = conc_B\big(\eta_B(\alpha), \eta_B(\beta)\big).$$

Now we define $\text{aggr}(f, \oplus) = b \circ Tf$. Let's prove the above operator satisfies (1) and (2). As $\eta : id \to T$ is a natural transformation and (B, b) is a **T**-algebra we have the following commutative diagrams



that is

$$Tf \circ \eta_A = \eta_B \circ f \tag{9}$$

$$b \circ \eta_B = id_B \tag{10}$$

From (9) and (10) we have

$$b \circ Tf \circ \eta_A = b \circ \eta_B \circ f = id_B \circ f = f \tag{11}$$

that is (1).

We have successively the following equalities:

$b \circ Tf \circ conc_A =$	(naturality of conc)
$= b \circ conc_B \circ (Tf \times Tf)$	(monad)
$= b \circ conc_B \circ (\mu_B \times \mu_B) \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf)$	(extended monad)
$= b \circ \mu_B \circ conc_{TB} \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf)$	((B, b) **T**-algebra)
$= b \circ Tb \circ conc_{TB} \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf)$	(naturality of conc)
$= b \circ conc_B \circ (Tb \times Tb) \circ (\eta_{TB} \times \eta_{TB}) \circ (Tf \times Tf)$	
$= b \circ conc_B \circ [(Tb \circ \eta_{TB} \circ Tf) \times (Tb \circ \eta_{TB} \circ Tf)]$	(naturality of η)
$= b \circ conc_B \circ [(\eta_{TB} \circ (b \circ Tf)) \times (\eta_{TB} \circ (b \circ Tf))]$	

Thus we have

$$b \circ Tf \circ conc_A = b \circ conc_B \circ [(\eta_{TB} \circ (b \circ Tf)) \times (b \circ Tf))].$$

The above relation applied to (x, y) where $x, y \in \tau$ gives us

$$(b \circ Tf \circ conc_A)(x, y) = (b \circ Tf)(x{+}{+}y) = \text{aggr}(f, \oplus)(x{+}{+}y)$$

and

$$\begin{aligned} & b \circ conc_B \circ [(\eta_{TB} \circ (b \circ Tf)) \times (\eta_{TB} \circ (b \circ Tf))(x, y) \\ & = b \circ conc_B[(\eta_{TB} \circ (b \circ Tf))(x), (\eta_{TB} \circ (b \circ Tf))(y)] \\ & = b \circ conc_B(\eta_{TB}(\mathrm{aggr}(f, \oplus)(x)), \eta_{TB}(\mathrm{aggr}(f, \oplus)(y))) \\ & = b \circ conc_B([\mathrm{aggr}(f, \oplus)(x)], [\mathrm{aggr}(f, \oplus)(y)]) \\ & = \mathrm{aggr}(f, \oplus)(x) \oplus \mathrm{aggr}(f, \oplus)(y) \end{aligned}$$

that is (2). ■

Proposition 2.8. It holds

$$\mathrm{aggr}(\eta, ++) = id_T. \tag{12}$$

Proof. Because (TA, μ_A) is a **T**-algebra and from monads definition we have for $x \in TA$

$$\mathrm{aggr}(\eta, ++) = (\mu_a \circ T\eta_A)(x) = id_{TA}(x). \blacksquare$$

Proposition 2.9. If $f : A \to B$ and $h : B \to C$ is a **T**-algebra homomorphism then

$$h \circ b \circ Tf = c \circ T(h \circ f).$$

Proof. We have

$$\begin{array}{lll} h \circ b \circ Tf & = & (h \text{ } \mathbf{T}\text{-algebra homomorphism}) \\ c \circ Th \circ Tf & = & (T \text{ functor} \\ c \circ T(h \circ f). & \blacksquare & \end{array}$$

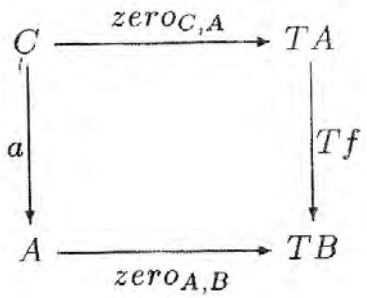
Corollary 2.10.

$$h \circ aggr(f, \oplus) = aggr(h \circ f, \otimes). \blacksquare$$

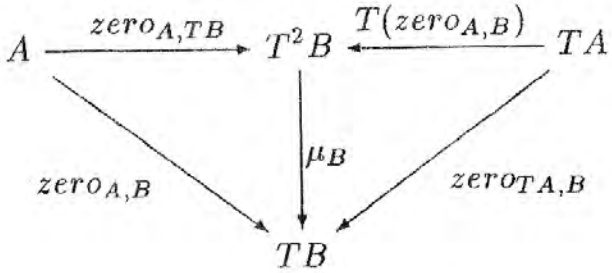
Definition 2.11. *A* **quad** *or* **monad with zero** *is a monad with a family of functions* $zero_{A,B}$ *which satisfies*

$$\begin{aligned} Tf \circ zero_{C,A} &= zero_{A,B} \circ g \\ \mu_B \circ zero_{A,TB} &= zero_{A,B} \\ \mu_B \circ T(zero_{A,B} &= zero_{TA,B} \end{aligned}$$

for each $g : C \to A$ *and* $f : A \to TB$, *that is the following diagrams commute:*



(*quad1*)



(*quad2*) ■

Remark 2.12. We have an equivalent definition stating for the family $zero_{A,B} : A \to TB$ the conditions (see [9]):

$$\begin{aligned} zero_{A,B} \circ g &= zero_{C,B} \\ (zero_{A,B})^* &= zero_{TA,B} \\ f^* \circ zero_{C,A} &= zero_{C,B}. \end{aligned}$$ ■

Definition 2.13. *The quadruple* $(T, \eta, -^*, ++)$ *is a* **semiringad** *if* $(T, \eta, -^*)$ *is a monad, and the concatenation* $++$ *verifies*

$$f^*(x{+}{+}y) = f^*(x){+}{+}f^*(y)$$

for each $f : A \to TA$. ■

Definition 2.14. *A* **ringad** *is a semiringad* $(T, \eta, -^*, ++)$ *which is also a quad, and the zero of quad is the neutral element for concatenation.*

The notion of ringad was introduced by Philip Wadler and colab. They consider it as a natural framework for data collection definition and study.

Semiringad and extended monad are equivalent notions.

Theorem 2.15. *The quadruple* $\mathbf{T} = (T, \eta, -^*, ++)$ *is an extended monad iff it is a semiringad.*

Proof.

(Necessity) If $f : A \to TB$ we have

(13) $$F^* = \mu_B \circ Tf$$

$$\begin{aligned} f^* \circ conc_A = \mu_B \circ Tf \circ conc_A &= && \text{(from (13)} \\ \mu_B \circ conc_{TB} \circ (Tf \times Tf) &= && \text{(extended monad)} \\ conc_B \circ (\mu_B \times \mu_B) \circ (Tf \times Tf) &= \\ conc_B \circ (\mu_B \times Tf) \circ (\mu_B \times Tf) &= && \text{(from (13)} \\ conc_B \circ (f^* \times f^*). \end{aligned}$$

(Sufficiency)

From $f^* \circ conc_A = conc_B \circ (f^* \times f^*)$ (semiringad property) we obtain

$$\begin{aligned} \mu_B \circ Tf \circ conc_A &= && \text{(naturality of conc)} \\ \mu_B \circ conc_{TB} \circ (Tf \times Tf) &= && \text{(naturality of conc)} \\ conc_B \circ (\mu_B \times \mu_B) \circ (Tf \times Tf). \end{aligned}$$

Because the last equality holds for each $f : A \to TB$, choosing f such that $(Tf \times Tf)$ be an epimorphism we have

$$\mu_B \circ conc_{TB} = conc_B \circ (\mu_B \times \mu_B).$$ ■

Remark 2.16. Each ringad is an extended monad which is also a quad and the zero of quad is neutral element for concatenation. ■

Theorem 2.17. *Each ringad determines a zero-element data collection.*

Proof. From remark 2.16, (1) and (2) are true. The proof proceeds as in theorem 2.7 putting $b([\,]) = u$. From ringad definition we have

$$\mathrm{aggr}(f, \oplus)(x) = \mathrm{aggr}(f, \oplus)(x{+}{+}[\,]) = \mathrm{aggr}(f, \oplus)(x) \oplus \mathrm{aggr}(f, \oplus)([\,])$$

which implies $\mathrm{aggr}(f, \oplus)([\,]) = u$, that is (4). ■

Corollary 2.18. Each ringad whose concatenation is associative determines a strong data collection. ■

3. Examples

Example 3.1. If T is the functor which maps a type (set) A to the free algebra with one binary operation (subject to no restriction) generated by A we obtain labeled rooted ordered binary trees. The mapping η maps $x \in A$ to the single-node labeled with x tree, the multiplication μ maps a tree whose labels are trees in TA to the tree obtained attaching to each node the tree having as name the label of that node. The concatenation of trees t_1 and t_2 is the tree having t_1 as left subtree and t_2 as right subtree. The empty tree is the zero of the monad. Alternatively, this collection type may be obtained using an adjunction from the category $\mathcal{B}$ of algebras with one binary operation to the category $\mathcal{T}$ of types. ■

Example 3.2. Let $T : \mathcal{T} \to \mathcal{T}$ be the functor which maps the type(set) A to Kleene's closure A^* of A (the underlying set of free-monoid generated by A) and the function $f : A \to B$ to the unique extension $f^* : A^* \to B^*$. Let $\eta A : A \to A^*$ be the application which maps a to the word $[a]$ (having length equal to 1) and $\mu A : A^{**} \to A$ which take a word of words $[s_1, \ldots, s_k]$ and maps it to the concatenation $s_1 \ldots s_k$ in A^* (obtained removing inner parentheses). The concatenation $concA : A^* \times A^* \to A^*$ is as usual and the null word is his neutral (unit) element. The mappings $\eta : id \to T$, $\mu : T^2 \to T$ and $conc : T \times T \to T$ are natural transformations and $(T, \eta, \mu, conc, zero)$ is a ringad ($zero_{A,B}$ is the null word). Thus we have the list collection type.

This ringad can also be obtained from the monad determined by forgetful functor $U : \mathbf{Mon} \to \mathcal{T}$ and his left adjoint (the free functor). ■

Example 3.3. Let $T : \mathcal{T} \to \mathcal{T}$ be the functor which maps the type(set) X to the underlying set of commutative free-monoid generated by X. TX is the set of equivalence classes of words made from symbols $x \in X$; a word which contain a segment xy is equivalent to the word obtained replacing all occurrences of xy by yx. Let $[w]$ be the equivalence class of w. The application ηX maps x to $[x]$ and μX maps a word of words in TX to a word in X removing inner parentheses. Concatenation of two words is the equivalence class of the word obtained through usual concatenation. Thus we obtain a ringad which determines the multiset(bag) collection type.

Equivalently, this monad is induced by the adjunction $< U, F, \eta, \mu >$ where U : **CMon** $\to \mathcal{T}$ is the forgetful functor and $F : \mathcal{T} \to$ **CMon** is his left adjoint (extended with concatenation and null word). ■

Example 3.4. Let be $P : \mathcal{T} \to \mathcal{T}$(or more generally P : **Set** $\to$ *Set*) given by $P(X) = \mathcal{P}(x)$(the powerset of X) and for $f : X \to Y$, $\mathcal{P}f : \mathcal{P}(X) \to \mathcal{P}(Y)$, $(\mathcal{P}f)(X) = f(X)$.
We define $\eta X : X \to \mathcal{P}(X)$, $\eta X(x) = x$ and $\mu X : \mathcal{P}\mathcal{P}(X) \to \mathcal{P}(X)$ which maps a set of sets to their union. The concatenation is the usual union and the zero is $\emptyset$. We have a ringad which generates the set collection type. The equivalent adjunction is determined by the underlying functor U : **UCSL** $\to \mathcal{T}$ from the category of upper complete semilattices to $\mathcal{T}$ (or **Set**) and his left adjoint (with corresponding extension). ■

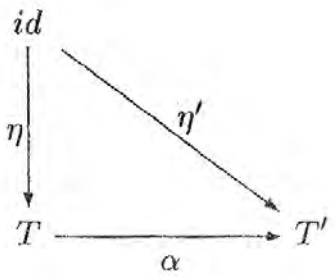
Remark 3.5. In fact, every equationally defined category of one sorted algebraic structures is equivalent to the category of Eilenberg-Moore algebras for some monad in **Set** (Linton's theorem [3]). ■

Example 3.6. (Arrays) The example is inspired from [2] which treats it in the context of computing the Fast Fourier Transform of a vector stored in a database. An array is like a list having in front its size. The unit of monad is a one element array, μ is the flatten(linearization) of a two dimensional array, the zero is the empty array, and the concatenation is the juxtaposition. ■

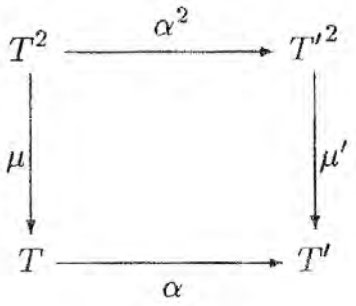
4. Type conversion

Type conversion would be an answer to the problem of managing simultaneous several collection type.

Definition 4.1. *Let* **T**=(T, η, μ) *and* **T'**=(T', η', μ') *two monads on the same category* $\mathcal{C}$. *A* **monad homomorphism** α : **T** $\to$ **T'** *is a natural transformation* $\alpha : T \to T'$ *such that the following diagrams commute:*



(*monadhom*1)



(*monadhom*2)

where $\alpha^2 = T'\alpha \circ \alpha T = \alpha T' \circ T\alpha$. ■

Definition 4.2. *Let* $\mathbf{T}=(T,\eta,\mu,conc)$ *and* $\mathbf{T'}=(T',\eta',\mu',conc')$ *extended monads on the same category* $\mathcal{C}$. *An* **extended monad homomorphism** *from* $\mathbf{T}$ *to* $\mathbf{T'}$ *is a monad homomorphism* $\alpha : \mathbf{T} \to \mathbf{T}'$ *such that the following diagram commutes:*

$$\begin{array}{ccc} T \times T & \xrightarrow{conc} & T \\ {\scriptstyle \alpha \times \alpha}\downarrow & & \downarrow{\scriptstyle \alpha} \\ T' \times T' & \xrightarrow[conc']{} & T' \end{array}$$

(mexthom) ■

Definition 4.3. *Let* $\mathbf{T}=(T,\eta,\mu,zero)$ *and* $\mathbf{T'}=(T',\eta',\mu',zero')$ *two quads on the same category* $\mathcal{C}$. *A* **quad homomorphism** *or a* **monad with zero homomorphism** *from* $\mathbf{T}$ *to* $\mathbf{T'}$ *is a monad homomorphism* $\alpha : \mathbf{T} \to \mathbf{T}'$ *such that the following diagram commutes:*

$$\begin{array}{ccc} A & & \\ {\scriptstyle zero_{A,B}}\downarrow & \searrow^{zero'_{A,B}} & \\ TB & \xrightarrow[\alpha_B]{} & T'B \end{array}$$

(quadhom) ■.

Definition 4.4. *Let* $\mathbf{T}=(T,\eta,\mu,conc,zero)$ *and* $\mathbf{T'}=(T',\eta',\mu',conc',zero')$ *two ringads on the same category* $\mathcal{C}$. *A* **ringad homomorphism** *from* $\mathbf{T}$ *to* $\mathbf{T'}$ *is an extended monad homomorphism* $\alpha : \mathbf{T} \to \mathbf{T}'$ *which is also a quad homomorphism.* ■

Let $C(T) =< \tau, [x], ++, \text{aggr} >$ and $C'(T) =< \tau', [x]', ++', \text{aggr}' >$ be two data collection types over the same type T, defined by the extended monads $\mathbf{T}=(T,\eta,\mu,conc)$ and $\mathbf{T'}=(T',\eta',\mu',conc')$.

Let's solve the problem of conversion from $C(T)$ to $C'(T)$.

Let $[x_1, \ldots, x_n] \in C(T)$. We have from proposition 2.8

$$[x_1, \ldots, x_n] = [x_1] ++ \cdots ++ [x_n].$$

Usually, such a conversion is done as follows

$$\alpha([x]) = [x]' \tag{14}$$

$$\alpha([x_1, \ldots, x_n]) = [x_1]' ++' \cdots ++' [x_n]' = [x_1, \ldots, x_n]' \tag{15}$$

Proposition 4.5. The mapping α defined as above is an extended monad homomorphism.

Proof.

(a) Let's prove the naturality of α. Let $[x_1, \ldots, x_n] \in TA$. We have $Tf([x_1, \ldots, x_n]) = [f(x_1), \ldots, f(x_n)]$. Also,

$$\begin{aligned}(\alpha B \circ Tf)([x_1, \ldots, x_n]) &= \alpha B([f(x_1), \ldots, f(x_n)] = \\ &= [f(x_1), \ldots, f(x_n)]'\end{aligned}$$

and

$$\begin{aligned}(T'f \circ \alpha A)([x_1, \ldots, x_n]) &= (T'f)[x_1, \ldots, x_n]' = \\ &= [f(x_1), \ldots, f(x_n)]',\end{aligned}$$

that $\alpha B \circ Tf = T'f \circ \alpha A$.

(b) The commutativity of (*monadhom*1) follows immediately from the definition of α:

$$(\alpha \circ \eta)(x) = \alpha([x]) = [x]' = \eta'(x).$$

(c) Let's prove the commutativity of (monadhom2).

Let $[[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]] \in T^2A$.

$$\begin{aligned}(\alpha \circ \mu)\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]\right) &= \\ \alpha\left([x_{1,1}, \ldots, x_{1,k_1}, \ldots, x_{l,1}, \ldots, x_{l,k_l}]\right) &= \\ [x_{1,1}]' +\!\!+' \cdots +\!\!+' [x_{1,k_1}]' +\!\!+' \cdots +\!\!+' [x_{l,1}] +\!\!+' \cdots +\!\!+' [x_{l,k_l}]' &= \\ [x_{1,1}, \ldots, x_{1,k_1}, \ldots, x_{l,k_l}]' &\end{aligned} \tag{16}$$

On the other hand:

$$\begin{aligned}\alpha^2\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]\right) &= \\ (\alpha T' \circ T\alpha)\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]\right) &= \\ \alpha T'\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]'\right) &= \\ \left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots, x_{l,k_l}]]'\right) &\end{aligned}$$

and

$$\begin{aligned}(\mu' \circ alpha^2)\left([[x_{1,1}, \ldots, x_{1,k_1}], \ldots [x_{l,1}, \ldots x_{l,k_l}]]\right) &= \\ \mu'\left([[x_{1,1}, \ldots, x_{1,k_1}]', \ldots [x_{l,1}, \ldots x_{l,k_l}]']'\right) &= \\ [x_{1,1}, \ldots, x_{1,k_1}, \ldots, x_{l,k_l}]' &\end{aligned} \tag{17}$$

Relations (16) and (17) imply the desired commutativity.

(d) Let's prove now the commutativity of (mexthom).

Let $(x, y) = ([x_1, \ldots, x_k], [y_1, \ldots, y_l]) \in TA \times TA$. We have

$$(\alpha \circ conc)(x, y) = \alpha([x_1, \ldots, x_k, y_1, \ldots y_l]) = [x_1, \ldots, x_k, y_1, \ldots y_l]'$$

and

$$(conc' \circ (\alpha \times \alpha))(x, y) = conc'([x_1, \ldots, x_k]', [y_1, \ldots, y_l]') = [x_1, \ldots, x_k, y_1, \ldots y_l]'$$

that is (mexthom). ■

Let's extend now the conversion to zero-element collection:

$$\begin{aligned} \alpha([]) &= []' \\ \alpha([x]) &= [x]' \\ \alpha([x_1, \ldots, x_n]) &= [x_1]' +\!\!+' \cdots +\!\!+' [x_n]' = [x_1, \ldots, x_n]' \end{aligned} \tag{18}$$

Proposition 4.6. α defined above is a ringad homomorphism.

Proof. The previous proposition implies α is an extended monad homomorphism and (18) implies the commutativity of (quadhom). ■

References

[1] Catriel Beeri, Yoram Kornatzky – Algebraic Optimization of Object-Oriented Query Languages, *Report* 91-6, Hebrew University of Jerusalem, Israel, 1991.

[2] P. Buneman – The Fast Fourier Transform as a Database Query, *Technical Report* MS-CIS-93-27/L&C 60, University of Pennsylvania, March, 1993.

[3] Michael Barr, Charles Wells – *Toposes, Triples and Theories*, Springer Verlag, Berlin, Heildelberg, Tokyo, 1985.

[4] Michael Barr, Charles Wells – *Category Theory for Computing Science*, Prentice-Hall, 1990.

[5] L. Fegaras – A Uniform Calculus for Collection Types, *Technical Report* No. CS/E 94-030, OGI, December, 1994.

[6] L. Fegaras, D. Maier – Towards an Effective Calculus for Object-Oriented Query Languages, *ACM SIGMOD International Conference on Management of Data*, San-Jose, May, 1995.

[7] L. Fegaras, D. Maier – An Algebraic Framework for Physical OODB Design, *5th International Workshop on Database Programming Languages*, Gubbio, Italy, 1995.

[8] Joseph A. Goguen – A categorical manifesto, *Math. Struct. in Comp. Science*, vol. 1, pp. 49-67, 1991.

[9] E. G .Manes – *Algebraic Theories*, Springer-Verlag, Berlin, 1976.

[10] Saunders MacLane – *Categories for the Working Mathematicians*, Springer-Verlag, 1971.

[11] Benjamin C. Pierce – A Taste of Category Theory for Computer Scientist, *Research Report* CMU-CS-88-203, Carnegie-Mellon University, Pittsburgh, 1988.

[12] P. Wadler – The essence of functional programming, *19th Annual Symposium on Principles of Programming Languages*, Santa Fe, New Mexico, January, 1992.

[13] P. Wadler – Comprehending monads, *Math. Struct. in Comp. Science*, vol. 2 (1992), pp. 461-493.

[14] P. Wadler – Monads and composable continuation, *LISP and Symbolic Computation*, 7, pp. 39-56, 1994.

[15] P. Wadler – How to Declare an Imperative, *ILPS 95*, J. Lloyd (ed), 1995.

"BABEŞ-BOLYAI" UNIVERSITY, FACULTY OF MATHEMATICS AND COMPUTER SCIENCE, RO-3400 CLUJ-NAPOCA, ROMANIA

E-mail address: tradu@math.ubbcluj.ro

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

SIMPLIFICATION OF MAGIC-SET RULES BY LOGIC GRAMMARS

D. TĂTAR AND V. VARGA

Abstract. The *magic sets* method is a bottom-up query evaluation technique that solves a query with particular *adornments* to the goal predicate. The program is transformed to an equivalent program using its adorned rules and *sideways information passing*. The transformed program has a bigger set of rules .The logic grammar as introduced in [6] can be used to simplify these rules.

1. Logic grammars

In [6] we defined a new concept of "Logic grammar" (LG) and we showed the soundness and completeness of them. In this section we shortly recall the basic concepts that are used in definition of "Logic grammar". Section 2 will present the magic-set method of optimization in Datalog programs [5] and in section 3 we will use LG's for simplification of magic-set rules. The language considered here is essentially that of the first-order predicate logic without function symbols. Let

- Pr be a set of predicates,
- C be a set of constants,
- V be a set of variables.

An atom over $C \cup V$ is of the form

$$p(u_1, \cdots, u_n), n \geq 0,$$

where $p \in Pr$ with arrity n, and each u_j is an element of $C \cup V$. If the arguments u_j are not interesting in a particular context, then we will denote an atom simply by p. Let A be the set of atoms over $C \cup V$, and, if $P' \subseteq Pr$, let $A_{P'}$ be the set of atoms with predicate symbols from P'. In some recent papers [1], [2], the set of predicates is considered as divided into two disjoint sets: the set EDB of extensional predicates (or extensional database predicates) which represent basic

Received by the editors: January 27, 1997.

1991 *Mathematics Subject Classification.* 68N17.

1991 *CR Categories and Descriptors.* D.1.6 [Programming Techniques]: Logic Programming; H.2.3 [Database Management]: Languages – data description languages, query languages.

"facts", and the set IDB of intensional databases predicates, representing facts deduced from the basic facts via the logic program [11]. Particularly, the set EDB can be the empty set (as, for simplicity, in most of the following demonstrations).

Definition 1.1. *A logic program P is a sequence of Horn clauses (definite clauses), that is, clauses of the form:*

$$p \leftarrow q_1, ..., q_n,$$

where p and $q_1, ..., q_n$ are atomic formulas in first-order logic, the comma is the logic operation "and", and the sign $\leftarrow$ is "if" or reverse of the logical implication.

We refer to the left (p) and right-hand side ($q_1, ..., q_n$) of a clause as its head and body. A clause is logically interpreted as the universal closure of the implication $q_1 \wedge \cdots \wedge q_n \rightarrow p$. If a clause has no right-hand side we will call it a fact or a unit clause. Let us observe that this definition considers only the class of positive logic programs (all atoms in all clauses are positive). From the properties of IDB and EDB predicates it follows that a predicate from the set EDB cannot occur in the head of clauses, but a predicate from the set IDB can occur in the set of facts.

Definition 1.2. *A goal G consists of a conjunction of atoms, and is denoted by:*

$$\leftarrow r_1, \cdots, r_t.$$

The following definition considers the fact that a goal G is treated by the resolution mechanism as a word rewritten by some well-defined rules. The last form of this rewriting tells us about some properties of the goal G.

Definition 1.3. *The logical grammar GL associated with a logic program P and the goal G is the system:*

$$GL = (I_N, I_T, X_0, F)$$

where:

- $I_N = A_{IDB} \cup \{X_0\}$ *is the set of nonterminals,*
- $I_T = A_{EDB} \cup \{\lambda\} \cup \{False, True\}$ *is the set of terminals,*
- X_0 *is the goal G,*
- F *is a finite set of production rules, of the form:*

$$p \rightarrow q_1 \cdots q_m, m \geq 1,$$

where $p \in IDB$ and "$p \leftarrow q_1, ..., q_m$" is a clause in the program P.

or

$$p \rightarrow \lambda$$

where "p." is a unit clause in the program P.

We assume, in the following, that substitutions, composition of substitutions, and the most general unifier $\sigma = \mathbf{mgu}(g, h)$ of atoms g and h are defined as in logic programming, [1,2].

For a logic grammar GL we define the rewriting relation "$\Rightarrow$" as follows:

Definition 1.4. *If $R \in A^+$ and $Q \in A^*$, then*

$$R \Rightarrow^{\sigma}_{GL} Q$$

if there exists an atom $h \in I_N$, and a production rule in F:

$$g \rightarrow h_1 ... h_m$$

such that:

$$R = R_1 h R_2, \sigma = \mathbf{mgu}(h, g)$$

and

$$Q = \sigma(R_1)\sigma(h_1)\cdots\sigma(h_m)\sigma(R_2)$$

(Here the variables of the production rule are renamed to new variables, so that all the variables in the rule do not appear in R).

Let $\Rightarrow *$ denote the reflexive and transitive closure of the relation $\Rightarrow$. As in formal language, modulo the peculiarity of the above relation $\Rightarrow$, an "derivation" is the sequence:

$$G_1 \Rightarrow^{\theta_1} G_2 \cdots \Rightarrow^{\theta_{n-1}} G_n.$$

We denote by $G_1 \Rightarrow^{\theta} * G_n$ the fact that θ is the composition of all substitutions in every direct derivation.

Definition 1.5. *The language generated by a logical grammar $GL = (I_N, I_T, X_0, F)$ is $L(GL) = \{(R, \theta) | X_0 \Rightarrow^{\theta} * R, R \in A^*_{I_T}, \theta = \theta_1 \cdots \theta_k$, k is the length of derivation for R, and θ_i is the substitution in the step $i\} \cup \{\Omega\}$*

We have some possibilities for the pair (R, θ) :

- if X_0 (or the goal G) is a ground formula, (not containing the variables) , then the substitution θ is the empty substitution, and R is $True$ or $False$, depending on the fact that G is a formula deducible or not from the set of clauses of P (by refutation);
- if X_0 contains variables, and the computation terminates, in the pairs (R, θ) we have $R \in I^*_T$, and the number of pairs is the number of solutions. If $EDB = \phi$, then $R = \lambda$. Let us denote R in the last situation by [ ], the empty clause, like usually in logic of resolution, and let θ be the answer substitution.
- if the program P is not terminating for the goal G, then $L(GL) = \{\Omega\}$, where $\Omega \notin IDB \cup EAB$.

2. Magic-set transformation

A *general deductive database* is defined as a pair $(\mathbf{D},\mathcal{L})$, where $\mathbf{D}$ is a finite set of clauses and $\mathcal{L}$ is a first order language. It is assumed that $\mathcal{L}$ has at least two symbols, one representing a constant symbol and another one representing a predicate symbol. A *definite* (resp. *normal*) database is a deductive database $(\mathbf{D},\mathcal{L})$, where $\mathbf{D}$ contains only definite (resp. normal) clauses. A *relational database* is a deductive database $(\mathbf{D},\mathcal{L})$, where $\mathbf{D}$ contains only definite facts.

Datalog is a logic programming language designed for use as a database language [2]. It is nonprocedural, set-oriented with no order sensivity, no special predicates, and no function symbols.

Consider a *Datalog* program P. As we mentioned in section 1, an *IDB predicate* is a predicate that appears in the head of some rule; if a predicate does not appear in any head, then it is an *EDB predicate*. The *EDB* (*extensional database*) is a set of relations for the *EDB* predicates; each relation is a set of tuples (or ground facts). The *IDB* (*intensional database*) is a set of relations for the *IDB* predicates. The *EDB* is the input for program P, and the *IDB* is the output and it can be computed by applying the rules of P to the *EDB*.

Because *Datalog* programs operate on potentially large databases, efficient computation of them is very important. The optimization methods are classified according to various criteria like: *formalism*, the *search strategy*, the *objective of the optimization*. There are two altenative formalisms: *algebric* and *logic*. In the case of logic formalism the evaluation of a *Datalog* goal requires building a proof tree. This tree can be constructed in two different ways: *bottom-up*, starting from the existing facts and inferring new facts, thus going towards the conclusions, or *top-down* trying to verify the premises which are needed in order for the conclusion to hold. From the objective of optimization method point of view some methods perform *program transformation*, namely, they transform a program into another program which is written in the same formalism, but which yields a more efficient computation when one applies an evaluation method to it; we refer to these as *rewriting methods*. These methods contrast with the *pure evaluation methods*, which propose effective evaluation strategies, where the optimization is performed during the evaluation itself.

The *magic sets* method is a bottom-up query evaluation technique which solves a query with particular *adornments* to the goal predicate. The program is transformed to an equivalent program using its adorned rules and *sideways information passing*. The transformed program models the constant propagation strategy of top-down methods through its *magic subgoals* added to the body of rules in the original program.

Let P be a *Datalog* program, and consider a goal on P. We can view the goal itself as defined by a rule and we add the goal rule to the program.

Definition 2.1. *An adornment of an atom* A *is an assignment either bound or free (abbreviated to* b *or* f *respectively) to each argument of* A. *An adornment of an atom with* n *arguments is denoted as a* n*-tuple. An atom* $p(t_1, ..., t_n)$ *with adornments* $< a_1, ..., a_n >$ *is denoted as* $p^{a_1,...,a_n}(t_1, ..., t_n)$ *where each* a_i *is assigned to* t_i *and is either* b *or* f.

Definition 2.2. *Given an adornment of the head of rule* r, *an argument of a subgoal of* r *is said to be* distinguished *if either:*

1. *It is a constant, or*
2. *It is a variable occuring in the head of* r *and the corresponding adornment is* b, *or*
3. *It appers in a* EDB *subgoal of* r *which has a distinguished argument.*

From this definition, variables in a *EDB* predicate occurrence are either all distinguished or all not distinguished. A *EDB* predicate occurrence with all variables distinguished is a *distinguished predicate occurrence.*

For each rule r of P, and for each adornment of the head predicate of r, we generate an *adorned rule* as follows. We consider all the distinguished arguments to be bound; this will generate an adornment for all the *IDB* predicates that are in the rule. The rule obtained by replacing all these predicates with their adorned version is an *adorned rule.*

We give distinct number to different occurences of the same predicate p in the right-hand side of a rule. For predicate p, we denote its i-th occurrence by "p_i". If there is only one occurrence of a certain predicate in the right-hand side of a rule, we may omit the occurrence number in that rule.

Definition 2.3. *We say that an adorned rule is* reachable *for the goal iff either it is the adorned rule corresponding to the goal rule, with all the* LHS *predicate arguments free, or its head predicate appears, with the same adornment, in the* RHS *of a reachable rule.*

ALGORITHM MAGIC SETS

Input:
A set of adorned rules P^A, including the goal rule, all reachable from the goal.
Output:
A new set of rules P^{magic}, equivalent to P^A with respect to the goal.
Method:
$P^{magic} := P^A$;
For each adorned rule r, and **For** each occurence of an intensional predicate p in the *RHS* of r **Do**
Begin
Generate one *magic rule* in the following way:
a) Delete all other occurrences of *IDB* predicates in the *RHS*;
b) Replace the name of p in this occurrence with $magic\_r\_p^a\_i$

where a is the adornment of p in that occurrence and i is the occurrence number;

c) Delete all nondistinguished variables of this occurrence of p, thus possibly obtaining a predicate with fewer arguments;

d) Delete all nondistinguished *EDB* predicates in r;

e) Replace the name of the head predicate p' with $magic\_p'^{a'}$ where a' is the adornment of p';

f) Delete all nondistinguished variables of p';

e) Exchange the places of the head magic predicate and the body magic predicate;

Add this rule to P^{magic}.

End

For each adorned rule r in the original program **Do**
Begin
Generate a *modified* rule in the following way:
For each occurence of an intensional predicate p in the *RHS* of r **Do**
Begin
add to the *RHS* the predicate $magic\_r\_p^a\_i(X)$ where a is the adornment of the occurence of p, i is the occurence number, X is the list of distinguished arguments in this occurrence.
If p is not the head predicate
Then the magic predicate must be inserted just before that occurrence;
Else the magic predicate is to be inserted at the beginning of the rule body, before all other literals.
End
Replace r with its modified version in P^{magic};
End

For each *IDB* predicate p and **For** each adornment **Do**
Begin
Generate a *complementary* rule as follows:
For each adorned rule r, and **For** each occurrence of p in the *RHS* of r **Do**
Begin
add the rule:
$magic\_p^a(X) :- magic\_r\_p^a\_i(X)$
where i is the considered occurrence of p, a is its adornment and X is the list of its distinguished arguments.
End

Add this rule to P^{magic}

End

Endmethod

3. Application of logic grammars formalisme

Let us illustrate the simplification of the magic-set algorithm's output by an example from [2].

Example 3.1. *The program P is:*

: r_1: $anc(X,Y) : -par(X,Y).$

: r_2: $anc(X,Y) : -anc(X,Z), par(Z,Y).$

Let us consider the goal ?-anc(X,a) which is added to the program P as the rule r_0.

The reachable adorned system P^A is:

: R_0: $q^f(X) : -anc^{fb}(X,a).$

: R_1: $anc^{fb}(X,Y) : -par(X,Y).$

: R_2: $anc^{fb}(X,Y) : -anc^{fb}(X,Z), par(Z,Y).$

As result of first DO loop, the following rules are gencrated:

: from R_0 $:magic\_R_0\_anc^{fb}(a).$

: from R_2 $:magic\_R_2\_anc^{fb}(Z) : -magic\_anc^{fb}(Z), par(Z,Y).$

As result of second DO loop, the following rules are generated:

: from R_0 : $q^f : -magic\_R_0\_anc^{fb}(a), anc^{fb}(X,a)$

: from R_2 : $anc^{fb}(X,Y) : -magic\_R_2\_anc^{fb}(Z), anc^{fb}(X,Z), par(Z,Y)$

As result of third DO loop, the following rules are generated:

: from R_0 $:magic\_anc^{fb}(X) : -magic\_R_0\_anc^{fb}(X).$

: from R_2 $:magic\_anc^{fb}(X) : -magic\_R_2\_anc^{fb}(X).$

Finally, the following equivalent program is obtained:

1. $:magic\_R_0\_anc^{bf}(a).$
2. $:magic\_R_2\_anc^{fb}(Z) : -magic\_anc^{fb}(Z), par(Z,Y).$
3. : $q^f : -magic\_R_0\_anc^{fb}(a), anc^{fb}(X,a)$
4. : $anc^{fb}(X,Y) : -magic\_R_2\_anc^{fb}(Z), anc^{fb}(X,Z), par(Z,Y)$
5. $:magic\_anc^{fb}(X) : -magic\_R_0\_anc^{fb}(X).$
6. $:magic\_anc^{fb}(X) : -magic\_R_2\_anc^{fb}(X).$

The logical grammar GL associated with this logic program P and the goal G is the system:

$$GL = (I_N, I_T, X_0, F)$$

where I_N is the set of predicates, I_T is as in section 1, X_0 is the goal G, F is the finite set of production rules, of the form:

1. $magic\_R_0\_anc^{fb} \rightarrow \lambda.$

2. $magic\_R_2\_anc^{fb} \rightarrow magic\_anc^{fb}, par$
3. $q^f \rightarrow magic\_R_0\_anc^{fb}, anc^{fb}$
4. $anc^{fb} \rightarrow magic\_R_2\_anc^{fb}, anc^{fb}, par$
5. $magic\_anc^{fb} \rightarrow magic\_R_0\_anc^{fb}$
6. $magic\_anc^{fb} \rightarrow magic\_R_2\_anc^{fb}$

The last two rules (5 and 6) are of type "renaming" rules. Moreover, in the clauses 5 and 6, the two pairs of predicates $magic\_anc^{fb}, magic\_R_0\_anc^{fb}$ and $magic\_anc^{fb}, magic\_R_2\_anc^{fb}$ have the same arguments: X. That means that the "rewriting" relation defined in section 1 has the property: $\forall\theta$ and $\forall G_n$, $magic\_anc^{fb} \Rightarrow^{\theta} *G_n$ iff $magic\_R_0\_anc^{fb} \Rightarrow^{\theta} *G_n$. Analogously, $\forall\theta$ and $\forall G_n$, $magic\_anc^{fb} \Rightarrow^{\theta} *G_n$ iff $magic\_R_2\_anc^{fb} \Rightarrow^{\theta} *G_n$.

As in formal grammars [7], [4], this grammar can be transformed by elimination of useless predicates $magic\_R_0\_anc^{fb}$ and $magic\_R_2\_anc^{fb}$. The obtained grammar GL' has the same generative power : $L(GL) = L(GL')$. Thus, the semantics of logic programs is the same.

The logic grammar after the elimination of "renaming" rules is the following:

: $1.magic\_anc^{fb} \rightarrow \lambda.$
: $2.magic\_anc^{fb} \rightarrow magic\_anc^{fb}, par$
: $3.q^f \rightarrow anc^{fb}$
: $4.: anc^{fb} \rightarrow par$
: $5.: anc^{fb} \rightarrow magic\_anc^{fb}, anc^{fb}, par$

The corresponding logic program is identical with the program in [2].

References

[1] K.R. Apt, M.H. van Emden *Contribution to the theory of logic programming* Journal of ACM, vol.29, 1982, pp. 841–862.

[2] S. Ceri, G. Gottlob, L. Tanca, *Logic Programming and Databases*, Spriger-Verlag, 1990.

[3] P. Deransart, J. Maluszynski *A grammatical view of logic programming* MIT Press, 1993.

[4] C.J. Hogger, *Derivation of Logic Programs*, Journal of ACM,

[5] J. Minker, *Perspective in deductive databases*, J. of Logic Programming, vol.5, 1988, pp. 33–61.

[6] D. Tatar, *Logic grammars as formal languages*, Studia Universitas "Babes-Bolyai", 1994, nr.3.

[7] J.E. Hopcroft, J.D. Ullman, *Introduction to automata theory, languages and computation*, Springer Verlag, 1979.

BABEŞ-BOLYAI UNIVERSITY, FACULTY OF MATHEMATICS AND INFORMATICS,
RO 3400 CLUJ-NAPOCA, STR. KOGĂLNICEANU 1, ROMANIA
E-mail address: {dtatar,ivarga}@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume 1, Number 2, October 1996

HOW TO MODEL MESSAGE PASSING BY LAMBDA-CALCULUS

S. MOTOGNA

Abstract. The paper contains some aspects concerning message passing in object oriented languages which are specified through lambda- calculus. This aspect is most often not taken in view while other object oriented features are well presented. The semantics used is based on lambda calculus with types. In this paper we prove that the two models for message passing are, in essence, equivalent.

1. Introduction

Object oriented programming has established itself as the most modern and natural way of software development. That's why recent studies try to formalise the OOP characteristics as better as possible. One of the way in which the research is carried on is lambda calculus, most of the studies following Cardelli and Wegner point of view [3].

The notions with which the OOP deals are objects, classes and methods (or messages, as method calls). The main features of OOP are: inheritance (or subtyping), polymorphism and information hiding (followed up from data abstraction). Various extensions of typed lambda calculus model these features.

In my point of view message itself is not "a big deal". Message passing seems to me a more important aspect, a more suitable notion, because is the only way in which the object can communicate.

A message syntax can be supposed of the following form:

message.receiver

without restraining its generality, because all object oriented languages has a similar form (if the receiver is not specified it is implicit).

The transmission of a message implies the existence of a receiver, which is an object (or more generally an expression returning an object); when it receives a message, the system, at run-time, selects among the methods defined for that

---

1991 *Mathematics Subject Classification.* 68NO5, 68Q55.

1991 *CR Categories and Descriptors.* D.1.5 [Programming Techniques] Object oriented programming, F.4.1 [Mathematical logic and formal languages] Mathematical logic, Lambda-calculus and related systems.

object the one whose name corresponds to the message name, if this exists. The existence of such a method should be checked at compile-time.

2. Message passing

There are two ways to understand message passing:

a): the first one is based on Cardelli's point of view [2] and consists of: the object is considered a record in which methods are its fields and whose labels are the messages of the corresponding methods. This point of view is know as "objects as records" analogy and can be found in [3], [5], [7].

b): the second way is used in the language CLOS and consists of: in the context of typed functional languages, message passing is viewed as a functional application in which the message is the function (identifier of the function) and the receiver is its argument [4].

In some sense these two interpretations are similar because records can be represented as functions from labels (messages) to values.

2.1. "Objects as records".

Let's review some aspects in favour of this choice:

- in Simula (considered an ancestor language in object oriented paradigm) objects are records with possibly functional components
- a record selection usually requires the selection label to be known at compile-time (allowing compile-time checking of the called method).

In [2] is presented how records can model the basic features of objects, including: inheritance, multiple inheritance, message passing, private instance variables and the special variable "self". The method chosen is based on typed lambda calculus with records and variants.

The subtype relation is defined as follows [2]: a record type γ is a subtype (written $\leq$) of a record type γ' if γ has all the fields of γ', and possibly more, and the common fields of γ and γ' are in the $\leq$ relation. The typing rules are:

[Rule 1a:] if $e_1 : \gamma_1$ and ... and $e_n : \gamma_n$ then $(a_1 = e_1, \ldots, a_n = e_n) : (a_1 : \gamma_1, \ldots, a_n : \gamma_n)$

[Rule 2a:] $\iota \leq \iota$ (ι a basic type, like *int* and *bool*)

$$\gamma_1 \leq \gamma_1', \ldots, \gamma_n \gamma_n' \Rightarrow (a_1 : \gamma_1, \ldots, a_{n+m} : \gamma_{n+m}) \leq (a_1 : \gamma_1', \ldots, a_n : \gamma_n')$$

[Rule 3a:]if $a : \gamma$ and $\gamma \leq \gamma'$ then $a : \gamma'$

[Rule 4a:]if $f : \sigma \to \gamma$ and $a : \sigma$ then $f(a)$ is meaningful, and $f(a) : \gamma$

[Rule 5a:] if $f : \sigma \to \gamma$ and $a : \gamma'$, where $\sigma' \leq \sigma$ then $f(a)$ is meaningful, and $f(a) : \gamma$

[Rule 6a:] if $\sigma' \leq \sigma$ and $\gamma \leq \gamma'$ then $\sigma \to \gamma \leq \sigma' \to \gamma'$

2.2. **"Message passing as functional application".** We note first that we must distinguish methods from functions, at least two main aspects being considered:

- *overloading* (the answer of two objects to the same message can differ, depending on the type of the object). Thus, messages are identifiers of the overloaded functions and in message passing the first argument of the overloaded function is represented by the receiver and the one on whose type the selection of the code to be executed is based. Each method constitutes a branch of the overloaded function referred by the message it is associated to;
- *late binding*: The difference consists in the fact that a function is bind to its meaning at compile-time, while the meaning of a method can be decided only at run-time when the actual type of the receiver is known. This feature is called late binding and it's shown up in the combination between overloading and subtyping.

Combining overloading with late binding implies a new distinction between message passing and ordinary functions. Overloading and late binding requires a restriction in the evaluation technique of arguments: while ordinary function application can be dealt with by either call-by-value or call-by-name, overloaded application with late binding can be evaluated only when the run-time type of the argument is known, i.e. when the argument is fully evaluated (closed and in normal form). In view of our analogy "messages as overloaded functions" this corresponds to say that message passing acts by call-by-value or, more generally, only closed and normal terms responds to messages.

An overloaded function is formed by a set of ordinary functions (lambda-abstractions), each one forming a different branch, and the notation is:

$$(M\&N)$$

for an overloaded function with two branches M and N that will be selected according to the type of the argument.

The subtyping relation is defined as follows:

$$\text{if } U_2 \leq U_1 \text{ and } V_1 \leq V_2 \text{ then } U_1 \to V_1 \leq U_2 \to V_2$$

and

$$\begin{array}{l}\text{if } \forall i \in I, \exists j \in J \text{ such that } U'_j -> V'_j \leq U''_i \to V''_i \\ \textit{then} \{U'_j \to V'_j\}_{j\in J} \leq \{U''_i \to V''_i\}_{i\in I}.\end{array}$$

The type-checking rules are [4]:

[Rule 1b:] $x^V : V$ x^V denotes $x : V$
[Rule 2b:] if $\lambda x^U.M : U \to V$ then $M : V$
[Rule 3b:] if $M : U \to V$ and $N : W \leq U$ then $MN : V$
[Rule 4b:] $\in$: {}

[Rule 5b:] if $M : W_1 \leq \{U_i \to V_i\}_{i\leq(n-1)}$ and $N : W_2 \leq U_n \to V_n$ then $(M\&^{\{Ui->Vi\}i\leq n}N) : \{U_i \to V_i\}_{i\leq n}$

[Rule 6b:] if $M : \{U_i \to V_i\}_{i\in I}$ and $N : U$, $U_j = \min_{i\in I}\{U_i | U \leq U_i\}$ then $M * N : V_j M * N$ denotes the overloaded application

3. Comparative study

First of all we must note that the same notation has been used for typing and subtyping relations.

If we study the two sets of typing rules we must find an "equivalence" between them. We will follow the equivalence from the first set of rules (from 2.a) to the second set of rules (from 2.b), since records can be represented as functions from labels (messages) to values.

[Rule 1a]: e_i are values of the corresponding type γ_i (for $1 \leq i \leq n$). Then if we proceed with labelling in a record we obtain Rule 1a. In other words the type of a record is obtained from the types of the labels. Since it is a rule specialized for records we can't find a match in the second set of rules.

[Rule 2a] We can decompose this rule as follows:

$$\begin{aligned}&\text{if } \gamma_1 \leq \gamma_1' \& \ldots \& \gamma_n \leq \gamma_n' \text{ then}\\ &(a_1 : \gamma_1, \ldots, a_n : \gamma_n) \leq (a_1 : \gamma_1', \ldots, a_n : \gamma_n')\\ &(a_1 : \gamma_1, \ldots, a_{n+m} : \gamma_{n+m}) \leq (a_1 : \gamma_1, \ldots, a_n : \gamma_n)\end{aligned}$$

Comparing this rule with [Rule 5b] which states that overloading a function (with n+1 members) with a new member N which type is a subtype of $U_n \to V_n$ the result has the type $\{U_i \to V_i\}_{i\leq n}$.

We can generalize [Rule 5b] as follows:

$$\begin{aligned}&\text{if } M : W_1 \leq \{U_i \to V_i\}_{i\leq(n-1)} \text{ and } N_1 : W_{21}, \ldots, N_m : W_{2m}\\ &\text{then } (M\&^{\{Ui->Vi\}i\leq n}N_1\&\ldots\&N_m) : \{U_i \to V_i\}_{i\leq n+m}\end{aligned}$$

and the correspondence with [Rule 2a] is observable.

[Rule 3a] is in fact the principle of subtyping.

The following 3 rules have as basic studying object the typing rules when functions are involved.

[Rule 4a] states that the type of the function value can be deducted from the definiton of the function, if it is meaningful. If we describe the function definition in the lambda-calculus notation we obtain the operation called application. And [Rule 2b] it is the exact transcription of the [Rule 4a].

[Rule 5a] is concerned with the case when we consider a subtype of the domain type of the function. [Rule 3b] states a similar situation when abstraction is involved.

[Rule 6a] is similar to the definition of the subtyping relation from 2b.

4. Conclusions

When necessary, we develope some theory from the most suitable point of view. We must never forget that this theory has to be consistent. Both ways of modelling message passing meet this criterion.

On the other hand, it might be possible that these two theories have to be unified. Then the problem of equivalence is asked.

We have argued that in principal the typing rules can be deduced one from another. The study takes into account only one direction of this deduction (from "objects as records" study to "message passing as functional application" study). The reverse sense of the equivalence can be deducted in a similar way.

Finally, we will note once again that the main idea of this deduction is the fact that records can be represented as functions from labels (messages) to values.

References

[1] K.B. Bruce, G. Longo, *A modest model of records, inheritance and bounded quantification*, Information and Computation, 87(1/2), 1990, pg. 196-240.

[2] L. Cardelli, *A semantics of multiple inheritance*, Semantics of Data Types, Lectures Notes in Computer Science, 173, Springer Verlag, 1984, pg. 51-67.

[3] L. Cardelli, P. Wegner, *On understanding types, data abstraction and polymorphism*, Computing Surveys, 17(4), dec. 1995, pg. 471-522.

[4] G. Castagna, G. Ghelli, G. Longo, *A semantics for λ-early: a calculus with overloading and early binding*, ACM Conference on Lisp and functional languages, 1994, pg. 107- 123.

[5] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall International Series, 1988.

[6] S. Motogna, *Formal Specification for Smalltalk through Lambda-Calculus. A Comparative Study*, Studia 3/1993.

[7] S. Motogna, V. Prejmerean, *Various kinds of inheritance*, Studia 3/1992, pg. 75-80

Babeş-Bolyai University, Faculty of Mathematics and Informatics,
RO 3400 Cluj-Napoca, str. Kogălniceanu 1, Romania
E-mail address: motogna@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

A PROOF METHOD FOR CLOSED NORMAL DEFAULT THEORIES

MIHAIELA LUPEA

Abstract. In his paper [4], Reiter defined a proof method for closed normal theories, based on the linear resolution for classical logic. A refinement of the Reiter method, using linear resolution with ordered clauses is proposed in this paper. This refinement is exemplified by some examples.

1. Introduction

Reiter introduced in [4] a very attractive formalization of the non-monotonic reasoning, adding to the first-order logic a new kind of inference rules called defaults. A default is an expression of the form $\frac{\alpha(x):\beta_1(x),\ldots,\beta_m(x)}{\gamma(x)}$, and is interpreted as follows : "if one believes $\alpha(x)$ and it is consistent to believe $\beta_1(x),\ldots,\beta_m(x)$, then one can also believe $\gamma(x)$ ".
The default logic presented in this paper is taken from [3] and [4].

Definition 1.1. *Let L be a first-order language. A* ***default theory*** $\Delta = (D, W)$ *consists of a set W of closed formulas in first-order logic and a set D of defaults $d = \frac{\alpha(x):\beta_1(x),\ldots,\beta_m(x)}{\gamma(x)}$, where*

-$\alpha(x), \beta_1(x), \ldots, \beta_m(x), \gamma(x)$ are well formed formulas in first-order logic;
-$\alpha(x)$ is called the ***prerequisite*** *of the default d;*
-$\beta_1(x), \ldots, \beta_m(x)$ are called ***justifications*** *of the default d;*
-$\gamma(x)$ is called the ***consequent*** *of the default d.*

The set W of formulas is the based knowledge of the theory, are treated as axioms, and we may reason about them using the inference rules of classical logic and the defaults, obtaining **beliefs**.
The defaults are inference rules which complete the incompletely specified world W. These rules model the non-monotonic reasoning, thus, we can infer conclusions,

Received by the editors: October 15, 1996.
1991 *Mathematics Subject Classification.* 03B35,68Q40,68T27.
1991 *CR Categories and Descriptors.* F4.1 [Mathematical Logic and Formal Languages] Mathematical Logic – mathematical theorem proving, proof theory, I2. [Artificial Intelligence] Deduction and Theorem Proving – nonmonotonic reasoning and belief revision, resolution.

later on invalidated by new informations which denied the justifications of some defaults already applied.

A default theory $\Delta = (D, W)$ is **consistent** if W is a consistent set of formulas.

A **default** is **closed** if none of $\alpha(x), \beta_1(x), \ldots, \beta_m(x), \gamma(x)$ contains a free variable. If the **default** is not closed, it is **open**. **A default theory is closed** if all of its defaults are closed.

Every open default theory may be transformed into a closed theory. Therefore in all what follows we will only consider the closed default theories. A closed default has the form $\frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma}$.

We denote

$PRE(D) = \bigcup\{\alpha | \frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma} \in D\}$, and

$CONS(D) = \bigcup\{\gamma | \frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma} \in D\}$.

Having a default theory we are interested to extend it (using the inference rules from classical logic and the defaults), in a consistent way, obtaining all the acceptable set of beliefs that one may hold about the incompletely specified world W.

Definition 1.2. *Let $\Delta = (D, W)$ be a closed default theory. For any set of closed formulas $S \subseteq L$, let $\Gamma(S)$ be the smallest set satisfying the following properties: (i) $W \subseteq \Gamma(S)$;*
(ii) $Th(\Gamma(S)) = \Gamma(S)$, where $Th(S) = \{P | S \vdash P\}$;
(iii) if $\frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma} \in D$ and $\alpha \in \Gamma(S)$ and $\neg\beta_1, \ldots, \neg\beta_m \notin S$ then $\gamma \in \Gamma(S)$.
*A set of closed formulas $E \subseteq L$ is an **extension** for Δ if and only if $\Gamma(E) = E$, i.e. E is a fixed point of the operator Γ.*

The default theories can have zero, one or more extensions.

Definition 1.3. *The set of **generating defaults** of an extension E of a closed default theory is*
$GD(E, \Delta) = \{d \in D | d = \frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma}$ *and* $\alpha \in E$ *and* $\neg\beta_1, \ldots, \neg\beta_m \notin E\}$

Theorem 1.4. *Let E be an extension of the closed default theory $\Delta = (D, W)$. Then $E = Th(W \cup CONS(GD(E, \Delta)))$.*

2. Normal default theories

Between default theories, there is a class of theories which have always an extension. These theories have all their defaults of the form $\frac{\alpha:\beta}{\beta}$, and are called **normal default theories**. Their defaults are called **normal defaults** and they model the most common non-monotonic rules used in practice :"if α, then believe β until the contrary of β is known". The normal default theories have many interesting properties according to the following theorems.

Theorem 2.1. *Every closed normal default theory has an extension.*

Theorem 2.2. ***(Semi-monotonicity)*** *Let $\Delta = (D, W)$ and $\Delta' = (D', W)$ be two normal default theories, $D' \subseteq D$, and E' an extension of Δ'. Then the theory Δ has an extension E such that:*
(i) $E' \subseteq E$;
(ii) $GD(E', \Delta') \subseteq GD(E, \Delta)$.

(i) says that the normal default theories are monotonic with respect to defaults. An important practical consequence of (ii) is that closed normal default theories admit a proof procedure which is local with respect to the defaults, so that proofs may be constructed which ignore some of the defaults.

Theorem 2.3. *(Orthogonality of extensions) If E and F are two extensions of the same normal default theory, then $E \cup F$ is an inconsistent set of formulas.*

Theorem 2.4. *Suppose $\Delta = (D, W)$ is a closed normal default theory such that $W \cup CONS(D)$ is consistent. Then Δ has a unique extension.*

In classical logics and monotonic logics all the formulas derived are valid, they are called theorems. For non-monotonic logics, a derived formula (a **belief**) is not necessary valid, it is only consistent with all the formulas of the extension to which belongs.
A proof theory for the default theories means a method for answering the question "given β, can β be believed?". This question may be formalized in :"given a closed normal default theory Δ and a closed formula $\beta \in L$, determine whether Δ has an extension E such that $\beta \in E$."

Definition 2.5. *(Reiter [4]) Let $\Delta = (D, W)$ be a closed normal default theory, and $\beta \in L$ a closed formula. A finite sequence $D_0, \ldots, D_k$ of finite subsets of D is a **default proof of** β with respect to Δ if and only if :*
(p1) $W \cup CONS(D_0) \vdash \beta$;
(p2) for $1 \leq i \leq k$
$W \cup CONS(D_i) \vdash PRE(D_{i-1})$;
(p3) $D_k = \emptyset$;
(p4) $W \cup \bigcup_{i=0}^{k} CONS(D_i)$ *is consistent.*

This definition does not provide a real proof procedure, because it gives no method for determining the D_i sets, nor does it specify a method to verify the consistency of a set of formulas. We may say that if the conditions *(p1)*, *(p2)*, *(p3)*, *(p4)* are satisfied, then β can be believed.

The completeness of this method results from the following two theorems.

Theorem 2.6. *Let $\Delta = (D, W)$ be a consistent closed normal theory, and $\beta \in L$ a closed formula. If β has a default proof $D_0, \ldots, D_k$ with respect to Δ, then Δ has an extension E such that $\beta \in E$.*

Theorem 2.7. *Suppose that E is an extension for a consistent closed normal default theory $\Delta = (D, W)$, and that $\beta \in E$. Then β has a default proof with respect to Δ.*

The condition *(p4)* is a test of consistency for a set of formula in first-order logic, but we know that this problem is not semi- decidable. Therefore we have the following theorem:

Theorem 2.8. *The extension membership problem for closed normal default theories is not semi-decidable.*

A proof method for normal default theories, inspired from the Reiter method is defined and studied in [5].

3. Linear resolution and OL-resolution

3.1. Linear resolution. Linear resolution, introduced by Loveland, provides a top down theorem prover. This refinement of resolution is complete and was used by Reiter ([4]) in his default proof method to obtain the D_i sets.

Definition 3.1. *A **linear resolution proof** of β from some set of clauses S has the form of figure1, where:*
*(i) the **top clause** R_0 is a clause of $\neg\beta$*
*(ii) for $1 \leq i \leq n$, R_i (called a **center clause**) is a resolvent of R_{i-1} and C_{i-1} (called a **side clause**)*
(iii) for $0 \leq i \leq n-1$, $C_i \in S$ or C_i is a clause of $\neg\beta$ or C_i is R_j for some $j < i$.
(iv) $R_n = \square$, the empty clause.

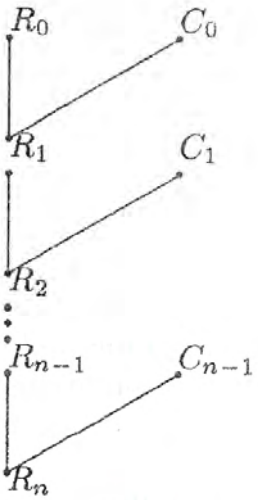


FIGURE 1. Linear resolution

3.2. **Ordered linear resolution.** We can strengthen the linear resolution by ordering the clauses and by using the information of the literals resolved upon, and we obtain the **ordered linear resolution**(OL-resolution). These two concepts increase the efficiency of linear resolution and do not destroy its completeness.
An ordered clause means that its literals are increasing ordered according to their position in the clause, from left to right. In the process of linear deduction we can require that the literal resolved upon from the center clause be the last in the clause.
The informations of the literals resolved upon allow to know if a side clause is a clause of the initial set or a center clause previously generated. We can keep the literal resolved upon from the center clause in the resolvent and it becomes **framed literal**.

Example 3.2. *The resolvent of the ordered clauses* $C_1 = P \vee Q$ *and* $C_2 = P \vee \neg Q \vee R$ *is the clause* $C = P \vee [Q] \vee P \vee R$.
We can apply ***merging left*** *operation for identical unframed literals and the clause* C *becomes* $P \vee [Q] \vee R$.

In this way we can lead the process of deduction, eliminating the expansive search of the side clauses.
These concepts are formalised in the following definitions and theorems taken from [1] and [2].

Definition 3.3. *An ordered clause* R *is a* ***reducible ordered clause*** *if and only if the last literal of* R *is unifiable with the negation of a framed literal of* R.

Definition 3.4. *Let* R_1 *and* R_2 *be two ordered clauses with no variables in common and* L_1, L_2 *be two unframed literals in* R_1 *and* R_2 *respectively. Let* L_1 *and* $\neg L_2$ *have a most general unifier* δ. *Let* R^* *be the ordered clause obtained by concatenating the sequence* $R_1\delta$ *and* $R_2\delta$ *, framing* $L_1\delta$, *removing* $L_2\delta$, *and merging left for any identical unframed literals in the remaining sequence. Let* R *be obtained from* R^* *by removing every framed literal not followed by any unframed literal in* R^*. R *is called an* ***ordered resolvent*** *of* R_1 *against* R_2. *The literals* L_1 *and* L_2 *are called the* ***literals resolved upon.***

Definition 3.5. *Let* R *be a reducible ordered clause. Let the last literal* L *be unifiable with the negation of some framed literal with a most general unifier* δ. *The* ***reduced ordered clause of*** R *is the ordered clause obtained from* $C\delta$ *by deleting* $L\delta$ *and every subsequent framed literal not followed by any unframed literal.*

Definition 3.6. *Given a set* S *of ordered clauses and an ordered clause* R_0 *in* S, *an* ***OL-deduction of*** R_n ***with top ordered clause*** R_0 *is a deduction which satisfies the conditions:*

(i) For $i=0,1,\ldots,n\text{-}1$, R_{i+1} *is an ordered resolvent of* R_i *(called a* ***center ordered clause****) against* C_i *(called* ***side ordered clause****). The literal resolved upon*

in R_i is the last literal.

(ii) Each C_i is either an ordered clause in S or an instance of some R_j, $j < i$. C_i is an instance of some C_j, $j < i$, if and only if R_i is a reducible ordered clause. In this case, R_{i+1} is the reduced ordered clause of R_i.

(iii) No tautology is in the deduction.

Theorem 3.7. ***(Completeness of OL-deduction)*** *If R is an ordered clause in an unsatisfiable set S of ordered clauses, then there is an OL-deduction of the empty clause from S with top ordered clause R.*

Example 3.8. *Let $S = \{P \vee Q, P \vee \neg Q \vee R, P \vee \neg Q \vee \neg R, \neg P \vee R, \neg P \vee \neg R\}$ be a set of clauses. We use OL-deduction to proved that the set S is unsatisfiable. The figure2 represents the OL-deduction of $\square$ from S with $P \vee Q$ as a top clause.*

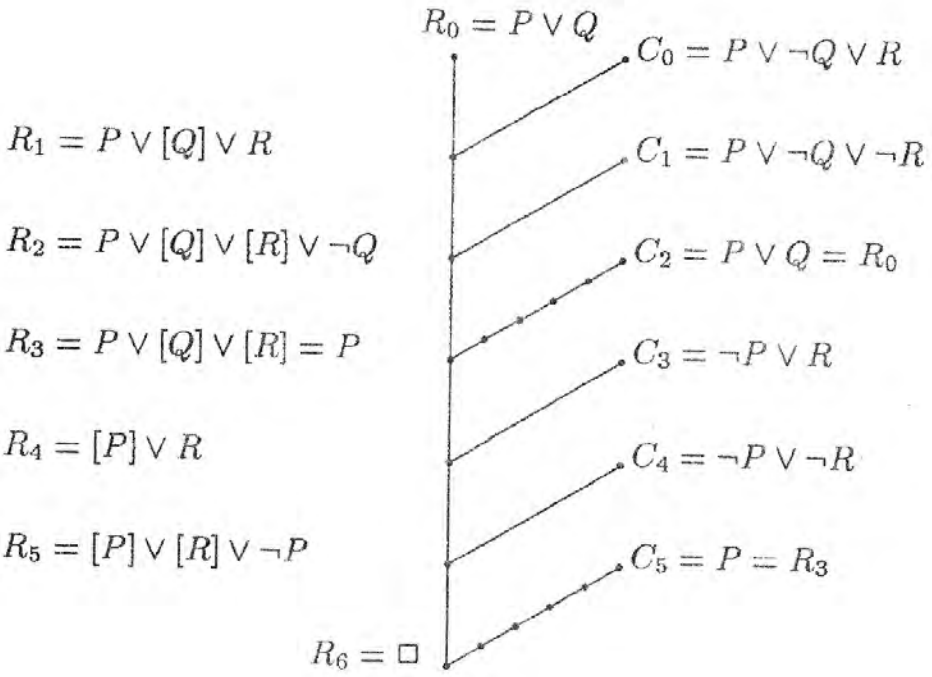


FIGURE 2. OL-deduction

In this example, R_2 is a reducible ordered clause, therefore C_2 is a center clause previously generated, namely R_0, and R_3 is the reduced ordered clause of R_2. Similar R_6 is the reduced ordered clause of R_5.
The steps 3 and 6 in the process of deduction can be eliminated, constructing directly the reduced ordered clauses R_3 and R_6. We can observe that the framed literals are useful only for the unframed literals which follow them.

This refinement of linear resolution is very easy to implement and is very efficient, because we need only the last resolvent (last center clause) in the process of deduction.

4. A default proof method for normal default theories using OL-resolution

A refinement of the Reiter proof method [4] for closed normal theories is proposed in this section. The OL-resolution seems to be appropriate to be used in the default proof method of Reiter for determine the sets D_i. The idea is to use OL-resolution to allow the goal β to help select a suitable subset D_0 of D, and so on. We must have an appropriate representation of a closed normal default theory for using the OL-resolution. Assume that W is a set of ordered clauses. For a normal default $d = \frac{\alpha:\gamma}{\gamma}$ suppose $C_1, ..., C_r$ are all the clauses of γ. A pair $(C_i, \{d\})$ is called an **ordered consequent clause** of the default d.
Let $\Delta = (D, W)$ be a closed normal default theory, where W is a set of ordered clauses. We define $CLAUSES(\Delta) = \{(C, \emptyset) | C \in W\} \cup$
$\cup\{(C, \{d\}) | d \in D$ and $(C, \{d\})$ is an ordered consequent clause of d$\}$.
A pair *(C,D)*, where C is an ordered clause and D is a set of defaults is called **indexed ordered clause**; C is said to be **indexed by** D.
The **resolvent** of the two indexed ordered clauses (C_1, D_1) and (C_2, D_2) is the indexed ordered clause $(R, D_1 \cup D_2)$, where R is the ordered resolvent of C_1 against C_2.
An OL-resolution of β from some set S of indexed ordered clauses has the properties:
- the top clause R_0 is an ordered clause of $\neg\beta$;
- for $1 \leq i \leq n$, R_{i-1} and C_{i-1} are indexed ordered clauses and R_i is their resolvent;
- for $0 \leq i \leq n-1$, $C_i \in S$ or C_i is a an ordered clause of $\neg\beta$ or C_i is R_j for some $j < i$;
- $R_n = (\square, D)$ for some set of defaults.

We say that such OL-resolution proof of β returns D.

Definition 4.1. *A top down default proof of β with respect to a closed normal default theory $\Delta = (D, W)$ is a sequence of OL- resolution proofs $L_0, ..., L_k$ such that:*
(i) L_0 is an OL-resolution proof of β from $CLAUSES(\Delta)$;
(ii) for $0 \leq i \leq k$, L_i returns D_i;
(iii) for $1 \leq i \leq k$, L_i is an OL-resolution proof of $PRE(D_{i-1})$ from $CLAUSES(\Delta)$;
(iv) $D_k = 0$;
(v) $W \cup \bigcup_{i=0}^{k} CONS(D_i)$ is consistent.

Theorem 4.2. *((Completeness of top down default proofs) Let $\Delta = (D, W)$ be a closed normal default theory, and β a closed formula. Then the theory has an extension E such that $\beta \in E$ if and only if β has a top down default proof with respect to the theory.*

The demonstration of this theorem results from the completeness of default proofs (Theorem 2.6 and Theorem 2.7) and the completeness of OL-resolution (Theorem 3.7). **Remarks:**

1. The defaults of $D_0, D_1, ..., D_{k-1}$ belong to the generating defaults set of the extension E.
2. If there are more extensions which contain the same formula β, then exists a top down default proof for β corresponding to each extension.
3. If none of the defaults of the theory has prerequisite, then the top down default proof consists in L_0 only.

Example 4.3. *Let $\Delta = (D, W)$ be a normal default theory, where $W = \{\neg P \vee R, P \vee \neg Q \vee \neg R\}$, and $D = \{d_1 = \frac{:P \vee \neg Q \vee R}{P \vee \neg Q \vee R}, d_2 = \frac{:\neg P \vee \neg R}{\neg P \vee \neg R}\}$.*
A top down default proof of $\beta = \neg P \wedge \neg Q$ is L_0 which returns $D_0 = \{d_1, d_2\}$ according to figure3.

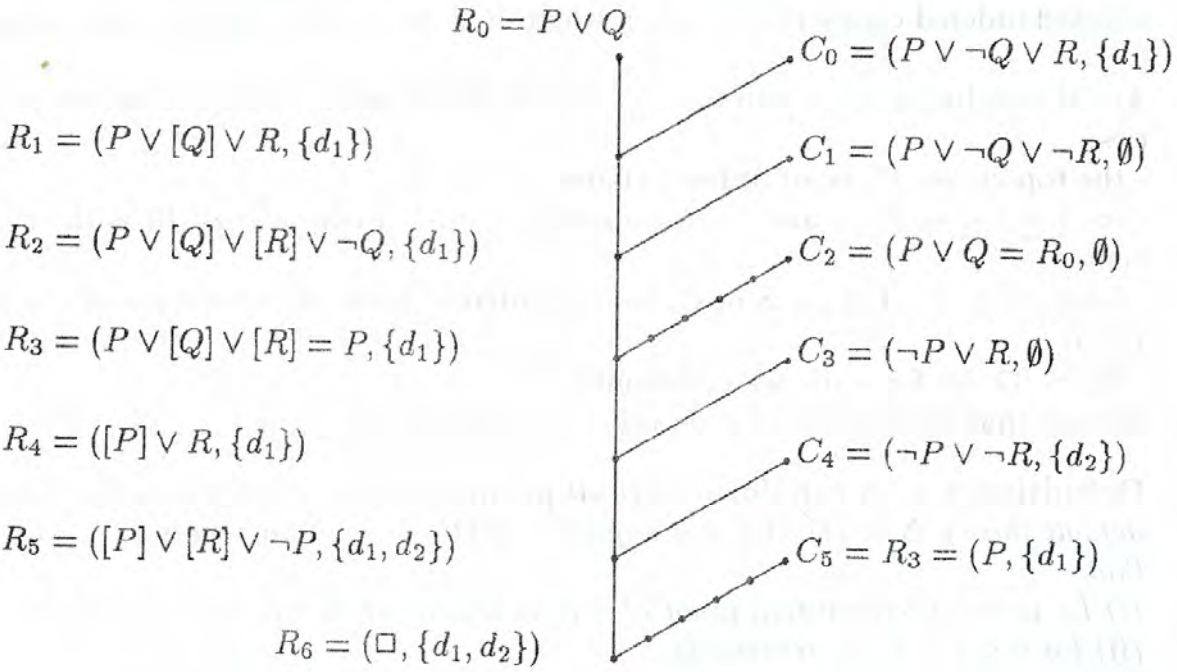
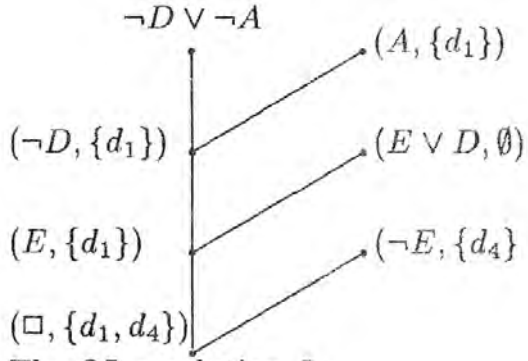


FIGURE 3. A top down default proof of $\neg P \wedge \neg Q$

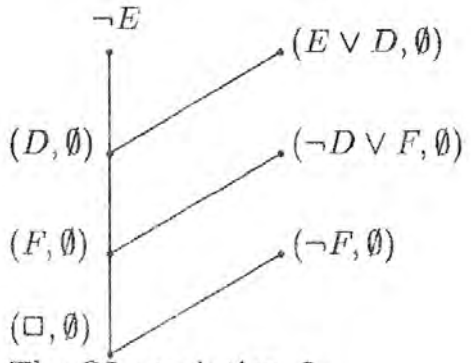
In this example R_0 is $\neg\beta$, $CLAUSES(\Delta) = \{(\neg P \vee R, \emptyset), (P \vee \neg Q \vee \neg R, \emptyset), (P \vee \neg Q \vee R, \{d_1\}), (\neg P \vee \neg R, \{d_2\})\}$. The set $W \cup CONS(\{d_1, d_2\})$ is

consistent. The theory of this example has an unique extension $E = Th(W \cup CONS(\{d_1, d_2\}))$, and $\beta \in E$.

Example 4.4. *Let $\Delta = (D, W)$ be a closed normal default theory, where $W = \{C \to D, A \wedge B \to E, E \vee D, D \to F\}$, and $D = \{d_1 = \frac{E \vee F : A \wedge F}{A \wedge F}, d_2 = \frac{A : B}{B}, d_3 = \frac{A \wedge E : C}{C}, d_4 = \frac{: \neg E}{\neg E}\}$.*
This theory has two extension $E_1 = Th(W \cup \{A \wedge F, B, C\})$ and $E_2 = \{Th(W \cup \{A \wedge F, \neg E\})$. We want to demonstrate that formula $\beta = D \wedge A$ belongs to both extensions. We will construct the top down default proof of β corresponding to extension E_2 (figure 4).



The OL-resolution L_0



The OL-resolution L_1

FIGURE 4

$CLAUSES(\Delta) = \{(A, \{d_1\}), (F, \{d_1\}), (B, \{d_2\}), (C, \{d_3\}), (\neg E, \{d_4\}), (\neg C \vee D, \emptyset), (\neg A \vee \neg B \vee E, \emptyset), (E \vee D, \emptyset), (\neg D \vee F, \emptyset)\}$
The top down default proof of $A \wedge D$ as an element of E_2 is the sequence L_0, L_1, where:
L_0 returns $D_0 = \{d_1, d_4\}$, $PRE(D_0) = \{\neg E, \neg F\}$
L_1 returns $D_1 = \emptyset$.
The set $W \cup CONS(\{d_1, d_4\})$ is consistent.
The top down default proof of $A \wedge D$ as an element of E_1 is the sequence L_0, L_1, L_2, L_3 where:
L_0 returns $D_0 = \{d_1, d_3\}$
L_1 returns $D_1 = \{d_2, d_1\}$
L_2 returns $D_2 = \{d_1\}$
L_3 returns $D_3 = \emptyset$
and can be constructed in a similar way. The set $W \cup CONS(\{d_1, d_2, d_3\})$ is consistent.

References

[1] C.L. Chang, R.C. Lee *Symbolic Logic and Mechanical Theorem Proving*, Academic Press Inc., 1973.

[2] A. Florea, A. Boangiu *Inteligenta artificiala*, Bucuresti, 1994.
[3] W. Lukasiewicz *Non-monotonic reasoning*, Ellis Horwood Limited, 1990.
[4] R. Reiter *A Logic for Default Reasoning*, Artificial Intelligence 13 (1980), pp. 81-132.
[5] D. Tatar, M. Lupea *A Note on Non-Monotonic Logics*, Studia Univ.Babes-Bolyai, Mathematica, XXXVIII, 3, 1993, pp. 109-115.

BABEŞ-BOLYAI UNIVERSITY, FACULTY OF MATHEMATICS AND INFORMATICS,
RO 3400 CLUJ-NAPOCA, STR. KOGĂLNICEANU 1, ROMANIA
E-mail address: lupea@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

THE NORMAL FORM OF A PERTURBED KEPLERIAN HAMILTONIAN

A. BLAGA

Abstract. In normalizing a perturbed Kepler Hamiltonian the Kepler Hamiltonian vector field is not complete, because solutions with angular momentum 0 reach the origin in *finite* time. We use first the *Moser regularization* and then obtain a *perturbed geodesic Hamiltonian*. In practice, computing the Poisson brackets will lead to discuss the constrained Hamiltonian systems and this paper deals with a genuine algorithmic method for computing a constrained normal form of a perturbed Keplerian Hamiltonian, involving the powerfull tool of Gröbner Bases Theory.

1. Introduction

Let f be a regularized perturbed Keplerian Hamiltonian. This paper describes an algorithm which brings f into normal form up to a certain order. A perturbed Keplerian Hamiltonian

$$f = H_0 + \varepsilon H_1 + \frac{\varepsilon^2}{2!} H_2 + \dots$$

is in normal form to order n iff $\{H_0, H_i\} = 0$, $i \in \{1, 2, \dots, n\}$, where H_0 is Kepler Hamiltonian which describes the motion of two bodies in $\mathbf{R}^3$ under the influence of the gravity.

For a general idea about regularization process see the appendix.

In the next section we will give a brief presentation of what a perturbed Kepler Hamiltonian is. At the end of section 3 we include the definition of a normal form.

The normal form algorithm starts in section 4 with some facts we need about Lie series and ends up with an important lemma about the space of Lie series. The mechanism of normal form algorithm can be found in section 5 but

Received by the editors: December 22, 1996.

1991 *Mathematics Subject Classification.* 13P10, 58F36, 68Q40.

1991 *CR Categories and Descriptors.* I.1.2 [Algebraic Manipulation]: Algorithms – algebraic algorithms.

the implementation techniques appears in the next section with the constrained normal form algorithm and the powerful tool of Gröbner bases ([1], [2]).

In the appendix one can follow the detailed presentation of the `MapleV` constrained normal form algorithm.

1.1. **The environment.** To explain our theory we need some background about Hamiltonian systems and perturbation theory. At the end of this section we will give the meaning of the perturbed Kepler Hamiltonian.

First of all let's consider $\mathbf{R}^3 - \{0\} \subseteq \mathbf{R}^3$, the space of positions $\xi = (\xi_1, \xi_2, \xi_3)$ in $\mathbf{R}^3$, without the origin. On $\mathbf{R}^3$ define the euclidean inner product $< \xi, \eta >= \xi_1\eta_1 + \xi_2\eta_2 + \xi_3\eta_3$ and its induced norm $|\xi|^2 = \xi_1^2 + \xi_2^2 + \xi_3^2$.

On $T\mathbf{R}^3$ with coordinates (ξ, η) we have the standard symplectic form $\omega = \sum_{i=1}^{3} d\xi_i \wedge d\eta_i$.

Definition 1.1. A Hamiltonian *on* $(T\mathbf{R}^3, \omega)$ *is a* smooth *function*

$$H : T\mathbf{R}^3 \to \mathbf{R}$$

where the dynamics *of the system* $(T\mathbf{R}^3, \omega, H)$ *are the solutions of the differential equation*

$$\begin{cases} \dot{\xi} &= \dfrac{\partial H}{\partial \eta} \\ \dot{\eta} &= -\dfrac{\partial H}{\partial \xi}, \end{cases} \tag{1}$$

which are called Hamilton's equations.

Remark 1.2. *The solutions of the Hamilton's equations are the* integral curves *of the Hamiltonian* vectorfield X_H *on* $T\mathbf{R}^3$.

Definition 1.3. *The* Kepler Hamiltonian *describes the motion of two bodies in* $\mathbf{R}^3 - \{0\}$ *under the influence of gravity and is given by:*

$$H_0 : T_0\mathbf{R}^3 \to \mathbf{R}, \quad H_0(\xi, \eta) = \frac{1}{2}|\eta|^2 - \frac{\mu}{|\xi|}. \tag{2}$$

Here, $T_0\mathbf{R}^3 = (\mathbf{R}^3 - \{0\}) \times \mathbf{R}^3 \subseteq T\mathbf{R}^3$.

Definition 1.4. A perturbation H, of a Kepler Hamiltonian *of* H_0 *is a formal power series*

$$H = H_0 + \varepsilon H_1 + \frac{\varepsilon^2}{2!} H_2 + \ldots, \tag{3}$$

where H_i, $i > 0$ *are smooth functions on* $T_0\mathbf{R}^3$.

Before defining the normal form of a perturbed Keplerian Hamiltonian we will give a short description of what a Poisson algebra is.

2. Poisson algebra

Definition 2.1. *Let* $(\mathcal{M}, \omega)$ *be a* smooth symplectic manifold. *Let* $\mathcal{F}(M)$ *be the space of* smooth formal power series *in* ε *with coefficients in* $C^\infty(\mathcal{M})$, *the space of smooth functions on* $\mathcal{M}$.

Remark 2.2. *For* $f, g \in C^\infty(\mathcal{M})$ *define* $f \cdot g \in C^\infty(\mathcal{M})$ *by* $(f \cdot g)(m) = f(m)g(m)$. *Then* $(\mathcal{F}(M), \cdot)$ *is a commutative algebra.*

For $f, g \in C^\infty(\mathcal{M})$ define a *Poisson bracket* $\{\cdot, \cdot\}$ by

$$\{f, g\}(m) = \omega(m)(X_f(m), X_g(m)),$$

where X_f, X_g are Hamiltonian vector fields corresponding to f and g.

If $f = \sum_{i=0}^{\infty} f_i \varepsilon^i$, $g = \sum_{i=0}^{\infty} g_i \varepsilon^i$ in $\mathcal{F}(M)$ then define a Poisson bracket on $\mathcal{F}(M)$ by

$$\{f, g\} = \sum_{k=0}^{\infty} \Big(\sum_{i=0}^{k} f_i g_{k-i} \Big) \varepsilon^k.$$

Since $(C^\infty(\mathcal{M}), \cdot, \{\cdot, \cdot\})$ is a *Lie algebra*, $(\mathcal{F}(M), \cdot, \{\cdot, \cdot\})$ is also. Moreover, since

$$\{f, g \cdot h\} = \{f, g\} \cdot h + g \cdot \{f, h\}$$

for $f, g, h, \in C^\infty(\mathcal{M})$ it also holds for $f, g, h \in \mathcal{F}(M)$. Therefore $(\mathcal{F}(M), \cdot, \{\cdot, \cdot\})$ is a Poisson algebra.

Definition 2.3. *We define the* adjoint map*:*

$$ad_f : \mathcal{F}(M) \to \mathcal{F}(M),$$

by $ad_f g = \{f, g\}, \quad f, g \in \mathcal{F}(M)$

Remark 2.4. *ad acts as a derivation on* $(\mathcal{F}(M), \cdot)$.

Definition 2.5. *Let* H_0 *be the Kepler Hamiltonian* (2). *The perturbed Kepler Hamiltonian* $H = H_0 + \varepsilon H_1 + \frac{\varepsilon^2}{2!} H_2 + \ldots$. H *is in* normal form *iff* $\{H_0, H_i\} = 0$ *for all* $i > 0$.

In the finite case, meaning that $\{H_0, H_i\} = 0$ for $0 < i \leq n$, we say that H is in *normal form to order* n.

The main goal of this paper is to compute the normal form of a given degree n for a perturbed Kepler Hamiltonian, using the very nice idea from [5]. Unfortunately this idea has never been implemented in a computer algebra system. We start with the background theory of the algorithm.

3. Lie series and the splitting lemma

Definition 3.1. *If $f \in \mathcal{F}(M)$ then*

$$\varphi_\varepsilon^f = exp(\varepsilon ad_f) = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} ad_f^n$$

is called Lie series.

Proposition 3.2. 1. *The formal flow of X_f, $f \in \mathcal{F}(M)$ is given by $\varepsilon \mapsto \varphi_\varepsilon^f$.*

2. *φ_ε^f is an automorphism of the formal Poisson algebra $(\mathcal{F}(M), \cdot, \{\cdot,\cdot\})$:*

$$\begin{aligned} \varphi_\varepsilon^f(gh) &= \varphi_\varepsilon^f(g) \cdot \varphi_\varepsilon^f(h), \\ \varphi_\varepsilon^f\{g,h\} &= \{\varphi_\varepsilon^f(g), \varphi_\varepsilon^f(h)\}. \end{aligned}$$

3. *$\varepsilon \mapsto \varphi_\varepsilon^f$ is one parameter group of automorphisms of formal Poisson algebra.*

Definition 3.3. *X_f, $f \in C^\infty(\mathcal{M})$, has periodic flow if there is $T > 0$ on $\mathcal{M}$ such that for every $m \in \mathcal{M}$ and $g \in C^\infty(\mathcal{M})$*

$$\varphi_{T(m)}^f g = g.$$

Remark 3.4. *T is not necessarily the minimal period of an integral curve of X_f.*

In the sequel we will give an important lemma, which is the basic tool in computing the normal form.

Lemma 3.5. *1. If $H, F \in \mathcal{F}(M)$ and φ_ε^f is the formal flow of X_f, $f \in \mathcal{F}(M)$ then*

$$(\varphi_\varepsilon^F)^* H = exp(\varepsilon ad_F) H.$$

2. **Splitting lemma.** *If X_{H_0} has periodic flow on $C^\infty(\mathcal{M})$ then*

$$C^\infty(\mathcal{M}) = \ker ad_{H_0} + \operatorname{im} ad_{H_0}. \tag{4}$$

Proof. The proof of the splitting lemma is based on solving the equation

$$L_{H_0} F = G, \quad F, G \in C^\infty(\mathcal{M}).$$

Given $F \in C^\infty(\mathcal{M})$ one can decompose it as $F = \overline{F} + (F - \overline{F})$, where $\overline{F}$ is the average:

$$\overline{F} = \frac{1}{T} \int_0^T (\varphi_t{}^{H_0})^* F \, dt. \tag{5}$$

Moreover, the equation $ad_{H_0} G = F - \overline{F}$ is solved by

$$G = \frac{1}{T} \int_0^T (\varphi_t{}^{H_0})^* (F - \overline{F}) \, dt. \tag{6}$$

The next section will deal with finding the normal form of a perturbed Kepler Hamiltonian. Section 6 gives the algorithms and several ideas in implementing the normal form.

4. Computing normal form

Before going into details we note that normal form theory finds a sequence of transformations which brings the perturbed Hamiltonian to normal form of a certain order.

A given $H = H_0 + \varepsilon H_1 + \frac{\varepsilon^2}{2!}H_2 + \ldots$, $H \in \mathcal{F}(M)$ will be brought into a normal form up to some order as follows.

For $G_1 \in C^\infty(\mathcal{M})$ we have the Lie series, $\varphi_\varepsilon{}^{G_1} = \sum_{n=0}^{\infty} \frac{\varepsilon^n}{n!} ad_{G_1}^n$. Then we change cordinates using $\varphi_\varepsilon^{G_1}$. The transformed Hamiltonian is:

$$\begin{aligned} F_\varepsilon^{(1)} &:= (\varphi_\varepsilon^{G_1})^* H = \\ &= H_0 + \varepsilon(H_1 + ad_{G_1} H_0) + \frac{\varepsilon^2}{2!}(H_2 + 2ad_{G_1} H_1 + ad_{G_1}^2 H_0) + \mathcal{O}(\varepsilon^3). \end{aligned}$$

In the above equation we have to find $G_1 \in C^\infty(\mathcal{M})$.

By the splitting lemma H_1 can be decomposed as $H_1 = \overline{H_1} + H_1'$, where $\overline{H_1} \in \mathbf{ker}\ ad_{H_0}$ and $H_1' \in \mathbf{im}\ ad_{H_0}$. Thus

$$F_\varepsilon^{(1)} = H_0 + \varepsilon(\overline{H_1} + H_1' + ad_{G_1} H_0) + \frac{\varepsilon^2}{2!}(H_2 + 2ad_{G_1} H_1 + ad_{G_1}^2 H_0) + \mathcal{O}(\varepsilon^3),$$

If we determine G_1 so that $H_1' = ad_{H_0} G_1$ then $F_\varepsilon^{(1)}$ is in normal form to the first order, since $\{H_0, \overline{H_1}\} = 0$. We now use (6) to determine G_1.

In order to compute the second order normal form it is important to preserve the already computed first order term from $F_\varepsilon^{(1)}$. Thus we have to use another change of coordinates, $\varphi_\varepsilon^{\varepsilon G_2}$. We obtain:

$$\begin{aligned} F_\varepsilon^{(2)} &:= (\varphi_\varepsilon^{\varepsilon G_2})^* F_\varepsilon^{(1)} = \\ &= H_0 + \varepsilon\overline{H_1} + \frac{\varepsilon^2}{2!}(H_2 + 2ad_{G_1} H_1 + ad_{G_1}^2 H_0 + 2ad_{G_2} H_0) + \mathcal{O}(\varepsilon^3). \end{aligned}$$

Here $G_2 \in C^\infty(\mathcal{M})$ should be determined by solving:

$$ad_{H_0} G = \tilde{H}_2{}',$$

where $\tilde{H}_2 = H_2 + 2ad_{G_1} H_1 + ad_{G_1}^2 H_0$ and $\tilde{H}_2 = \overline{\tilde{H}_2} + \tilde{H}_2{}'$, with $\overline{\tilde{H}_2} \in \mathbf{ker}\ ad_{H_0}$ and $\tilde{H}_2{}' \in \mathbf{im}\ ad_{H_0}$.

In general having computed the $(n-1)^{th}$ order normal form, one could find the n^{th} order normal form by

$$F_\varepsilon^{(n)} := (\varphi_\varepsilon^{\varepsilon^{n-1} G_n})^* F_\varepsilon^{(n-1)},$$

where $G_1, G_2, \ldots, G_n - 1$ are known and $G_n(G_1, G_2, \ldots, G_n - 1)$ has to be solved.

One might use the nonrecursive formula:

$$F_\varepsilon^{(n)} := (\varphi_\varepsilon^{\sum_{i=1}^n \varepsilon^{i+1} G_i})^* H,$$

where G_i, $i = 1, 2, \ldots, n$ is a *triangular* system of equations.

5. The constrained normal form algorithm

We introduce some notations.

Let

$$H_0(q,p) := \sqrt{|q|^2|p|^2 - < q,p >^2}, \quad (q,p) \in T\mathbf{R}^4 \text{ and} \tag{7}$$

$$T^+S^3 = \{(q,p) \in T\mathbf{R}^4 | \ |q|^2 = 1, < q,p >= 0, p \neq 0\},$$

$$F_1 = |q|^2 - 1 = 0, \quad F_2 =< q,p >= 0. \tag{8}$$

Define the field $K := \mathbf{R}(|p|, k)$ of rational functions in $|p|$ and k, with real coefficients and its extension $K \subseteq L := \mathbf{R}(|p|, k, H_0)$.

If $(q,p) \in T\mathbf{R}^4$ we define

$$\mathcal{F} := \{f = \sum_{n \geq 0} \frac{\varepsilon^n}{n!} f_n \mid f_n \in L[q,p]\}$$

and

$$\mathcal{G} := \{f \in \mathcal{F} \mid f_0 = H_0\}$$

Note that

$$\mathcal{F}|_{T^+S^3} = \{f = \sum_{n \geq 0} \frac{\varepsilon^n}{n!} f_n \mid f_n \in K[q,p]\}$$

and

$$\mathcal{G}|_{T^+S^3} = \{f \in \mathcal{F}|_{T^+S^3} \mid f_0 = |p|\}.$$

Remark. For $f \in \mathcal{G}$, $f|_{T^+S^3} \in \mathcal{G}|_{T^+S^3}$ is a *perturbed geodesic Hamiltonian* ([5]).

The constrained normal form algorithm takes as input $f \in \mathcal{G}$ and the order of normalization and outputs $\tilde{f} \in \mathcal{G}$, the normal form of f with the required order.

From section 4 the first order normal form can be easily computed by averaging $f_1 \in K[q,p]$. For H_0 given by (7) we restrict the flow $\varphi_t^{H_0}$ to T^+S^3. It is given by:

$$\varphi_t^{H_0}|_{T^+S^3}(q,p) = \begin{pmatrix} q\cos 2t + \frac{p}{|p|}\sin 2t \\ -|p|q\sin 2t + p\cos 2t \end{pmatrix} \tag{9}$$

This reduces the amount of computation needed in averaging. Thus we calculate

$$\overline{f_1} = \frac{1}{\pi}\int_0^\pi f_1\left(q\cos 2t + \frac{p}{|p|}\sin 2t, -|p|q\sin 2t + p\cos 2t\right)\, dt. \tag{10}$$

After few steps of normalization the integrand in (8) will have so many terms that it will not be manageable. For this we introduce the equivalence relation $\approx$. For $f, g \in \mathcal{G}$, we say that $f \approx g$ iff $(f-g)|_{T^+S^3} = 0$. In other words $f \approx g$ iff f and g have the same coset representative N in $L[q,p]/I$, where I is the ideal generated by T^+S^3. Later we will see that N is called normal form of f or g. Practicaly speaking, to find a coset representative we have to introduce *Gröbner bases* for our problem:

Definition 5.1. *Let be $I \neq \{0\}$ an ideal in $L[q,p]$ and $G = \{g_1, g_2, \ldots, g_n\} \subseteq I$ a set of nonzero polynomials in $L[q,p]$. We define a* term ordering *on $L[q,p]$ by $q_1 > q_2 > q_3 > q_4 > p_1 > p_2 > p_3 > p_4$. G is a Gröbner basis for I iff for all $f \in I$, $f \neq 0$ there exists an $i \in \{1, 2, \ldots, n\}$ such that the leading monomial of g_i divides the leading monomial for f.*

Proposition 5.2. *If $G = \{g_1, g_2, \ldots, g_n\}$ is a Gröbner basis for I then I is generated by $g_1, g_2, \ldots, g_n$, that is*

$$I =< g_1, g_2, \ldots, g_n > .$$

If $f \in \mathcal{G}$ and $\mathcal{I} =< F_1, F_2 >$, then $r \in \mathcal{G}$, obtained by a division algorithm on multivariate polynomials with respect to the unique *reduced* Gröbner basis of I, it is also unique ([1]). $r \in \mathcal{G}$ is called the *normal form* of f and it is denoted by $N(f)$.

Proposition 5.3. *If $f, g \in L[q,p]$ then*

$$f \equiv g \pmod{I} \quad \text{iff} \quad N(f) = N(g).$$

Thus our definition for equivalence relation, $\approx$, can be restated as:

$$f \approx g \quad \text{iff} N(f) = N(g),$$

where the normal form is taken with respect to the Gröbner basis of the ideal $\mathcal{I}$ generated by T^+S^3.

Because T^+S^3 is an invariant manifold of X_{H_0} we see that $f \approx g$ implies $\overline{f} \approx \overline{g}$. This means that in order to compute the average of f we might compute $\overline{f}|_{T^+S^3}$ by using the Gröbner basis for the constraints. Thus, in our case we calculate $\overline{f_1} \in \mathcal{G}|_{T^+S^3}$ by just normalizing $\overline{f_1} \in \mathcal{G}$ with respect to our Gröbner basis.

The next step is preparing for the second order normal form. First we solve the equation $\overline{f_1} = ad_{H_0} g_1$ as it was pointed in section 4. This can be done by:

$$g_1 = \frac{1}{\pi} \int_0^{\pi} t \left(f_1 - \overline{f_1}\right) \left(q \cos 2t + \frac{p}{|p|} \sin 2t, -|p| q \sin 2t + p \cos 2t\right) \, dt.$$

In this case g_1 must *not* be normalized with respect to the Gröbner basis. In general, X_{g_1} does not leave T^+S^3 invariant. Thus g_1 has to be modified to g_1^* (see

appendix):

$$g_1^* := g_1 - \frac{1}{2}\{g_1, < q,p >\}(|q|^2 - 1) + \frac{1}{2}\{g_1, |q|^2 - 1\} < q,p > .$$

Now $X_{g_1^*}$ leaves T^+S^3 invariant, meaning

$$\{g_1^*, < q,p >\}|_{T^+S^3} = 0$$

and

$$\{g_1^*, |q|^2 - 1\}|_{T^+S^3} = 0.$$

The last step in computing second order normal form is to use (6):

$$\tilde{f} = f_0 + \varepsilon\overline{f_1} + \frac{\varepsilon^2}{2!}\overline{(f_2 + 2ad_{g_1^*}f_1 + ad^2_{g_1^*}f_0)}.$$

Applying the same method one can compute higher order normal forms.

6. Appendix

6.1. **Regularization process.** In normalizing a perturbed Kepler Hamiltonian, some problems arise even at the very beginning: the Kepler Hamiltonian vector field is not complete. The reason for its incompleteness is that solutions with angular momentum 0 reach the origin in *finite* time.

To remove this incompletness we use *Moser regularization* ([3]), which changes the phase space from $T_0\mathbf{R}^3$ to $T(S^3 - (0,0,0,1))$, the tangent bundle of the sphere $S^3 \subseteq \mathbf{R}^4$ without the north pole.

First of all we introduce the time scale by $\frac{ds}{dt} = \frac{k}{|\xi|}$ and define

$$M(\xi,\eta) = \frac{|\xi|}{k}\left(H(\xi,\eta) + \frac{k^2}{2}\right) + \frac{\mu}{k},$$

where μ comes from (2). The Hamiltonian vector field of (2) is clearly:

$$\begin{cases} \dfrac{d\xi}{dt} &= \eta \\ \dfrac{d\eta}{dt} &= -\mu\dfrac{\xi}{|\xi|^3}. \end{cases} \tag{11}$$

We would like to have rescaled Hamiltonian equations for H, so we restrict (ξ,η) to lie in the level set $H^{-1}(-k^2/2)$ or equivalently $M^{-1}(\mu/k)$, $\mu/k > 0$.

Remark 6.1. *$H < 0 \Rightarrow H_0 < 0$, so we have bounded Keplerian orbits.*

Thus, after time rescaling, the equations satisfied by M on the level set $M^{-1}(\mu/k)$, are:

$$\begin{cases} \dfrac{d\xi}{ds} &= \dfrac{|\xi|}{k}\dfrac{\partial H}{\partial \eta} \\ \dfrac{d\eta}{dt} &= -\dfrac{|\xi|}{k}\dfrac{\partial H}{\partial \xi}. \end{cases} \tag{12}$$

As a last step of regularization, compose M with *Moser's map*:

$$\mathcal{M} : T(S^3 - (0,0,0,1)) \to T_0\mathbf{R}^3, \quad \mathcal{M}(q,p) = (\xi,\eta),$$

$$\xi_i = -\frac{1}{k}(p_i + q_i p_4 - p_i q_4),$$

$$\eta_i = \frac{kq_i}{1-q_4}, \quad i = 1,2,3.$$

The new Hamiltonian is:

$$G(q,p) = G_0(p) + \varepsilon G_1(q,p) + \frac{\varepsilon^2}{2!}G_2(q,p) + \cdots,$$

where $G_0(q,p) = |p|$. Thus a perturbed Kepler Hamiltonian is transformed into a *perturbed geodesic Hamiltonian* with its geodesic vector field X_{G_0} on

$$T^+S^3 = \{(q,p) \in T\mathbf{R}^4 |\ |q|^2 = 1, < q,p >= 0, p \neq 0\}.$$

$|q|^2 = 1$ is the condition for the body to move on the unit sphere. For it to have phase space TS^3 we need the extra condition $< q,p >= 0$.

Another problem that arises in practice is computing Poisson brackets on T^+S^3. This leads to the discussion of *constrained Hamiltonian systems.*

6.2. **Constraints.** The manifold $TS^3 \subseteq T\mathbf{R}^4$ defined by:

$$\begin{cases} F_1(q,p) &= |q|^2 - 1 = 0 \\ F_2(q,p) &= < q,p >= 0 \end{cases}$$

is called *constraint manifold.* Since the matrix

$$\begin{pmatrix} \{F_1,F_1\} & \{F_1,F_2\} \\ \{F_2,F_1\} & \{F_2,F_2\} \end{pmatrix} = \begin{pmatrix} 0 & 2 \\ -2 & 0 \end{pmatrix}$$

on TS^3 is non-singular, TS^3 is a symplectic manifold with symplectic form $\omega = \omega|_{TS^3}$. Since $T^+S^3 \subseteq TS^3$ is an open set, it is also a symplectic manifold.

In the sequel we will give a method to compute the Poisson brackets $\{,\}_{TS^3}$ on $C^\infty(TS^3)$ by $\{f,g\}_{TS^3} = \omega(X_f, X_g)$. The idea is to construct smooth functions F^* and G^*, which are extensions of f and g on $T\mathbf{R}^4$ so that:

$$\{f,g\}_{TS^3} = \{F^*,G^*\}|_{TS^3}, \tag{13}$$

and

$$F^*|_{TS^3} = f, \quad G^*|_{TS^3} = g.$$

Let F be an arbitrary smooth extension of f to $\mathbf{R}^4$. This extension exists by *Whitney extension theorem* ([4]), because TS^3 is closed subset of $T\mathbf{R}^4$.

Because TS^3 is not an *invariant manifold* if X_f we can not use F to compute $\{,\}_{TS^3}$. We have to modify F. Let $F^* = F + \alpha_1 F_1 + \alpha_2 F_2$ and choose $\alpha_1, \alpha_2 \in C^\infty(\mathbf{R}^4)$ so that $\{F^*, F_1\}|_{TS^3} = 0$ and $\{F^*, F_2\}|_{TS^3} = 0$. This means that TS^3 is invariant manifold of X_{F^*}. So, for $F \in C^\infty(T\mathbf{R}^4)$

$$F^* = F - \frac{1}{2}F_1 + \frac{1}{2}F_2,$$

the Poisson bracket $\{,\}_{TS^3}$ on $C^\infty(TS^3)$ is given by

$$\{F|_{TS^3}, G|_{TS^3}\}_{TS^3} = \{F^*, G^*\}|_{TS^3}.$$

Now we look at $H_0 = \sqrt{|p|^2|q|^2 - < q,p >^2}$. H_0 is smooth on the symplectic submanifold $\tilde{\mathcal{M}} = T\mathbf{R}^4 - \{H_0 = 0\}$ of $(T\mathbf{R}^4, \omega)$. Clearly $H_0|_{T^+S^3} = |p|$ and T^+S^3 is an invariant manifold of X_{H_0}. Another fact is that the flow $\varphi_t^{H_0}$ on $\tilde{\mathcal{M}}$ is given

$$\varphi_t^{H_0}(q,p) = \begin{pmatrix} -\frac{<q,p>}{H_0}\sin 2t + \cos 2t & \frac{|q|^2}{H_0}\sin 2t \\ -\frac{|p|^2}{H_0}\sin 2t & \frac{<q,p>}{H_0}\sin 2t + \cos 2t \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix}.$$

Clearly $\varphi_t^{H_0}$ is periodic with period π. See [6] for more details.

References

[1] W.W. Adams, P. Loustanau, *An Introduction to Gröbner Bases*, American Mathematical Society, 1994.

[2] D. Cox, J. Little, D. O'Shea, *Ideals, Varieties and Algorithms: an introduction to computational algebraic geometry and commutative algebra*, Springer-Verlag, Berlin, 1992.

[3] R. Cushman, *Normal Form for Hamiltonian Vectorfields with Periodic Flow* in *"Diffferential Geometric Methods in Mathematical Physics"*, pp. 125-144, ed. Sternberg, Reidel, Dordrecht, 1984.

[4] R. Cushman, J.A. Sanders, *The Constrained Normal Form algorithm*, Celestial Mechanics 45, pp. 181-187, 1989.

[5] R. Cushman, *A Survey of Normalization Techniques Applied to Perturbed Keplerian Systems*, Utrecht University, Department of Mathematics, preprint 598, 1990.

[6] R. Cushman, R. Bates, *Global Aspects of Classical Integrable Systems*, to appear.

[7] J,C. van der Meer, R. Cushman, *Constrained Normalization of Hamiltonian Systems and Perturbed Keplerian Motion*, Journal of Applied Mathematics and Physics (ZAMP), volume 37, pp. 402-424, 1986.

[8] J.C. van der Meer, R. Cushman, *Orbiting Dust under Radiation Pressure* in *"Diffferential Geometric Methods in Mathematical Physics"*, pp. 403-414, ed. Doebner, World Scientific, Singapore, 1987.

[9] P.J. Olver, *Applications of Lie Groups to Differential Equations*, Graduate Texts in Mathematics 107, Springer-Verlag, New York, 1986.

"Babeş-Bolyai" University, Faculty of Mathematics and Informatics, RO-3400 Cluj-Napoca, Romania

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

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

ANALYSING THE NOISE SENSITIVITY OF SKELETONIZATION ALGORITHMS

ATTILA FAZEKAS AND ANDRÁS HAJDU

Abstract. Many skeletonization algorithms have been analysed from several points of view in the past ten years to compare the results they produce. Our attempt here is to add a new comparative analysis to this research area which investigates the noise sensitivity of these algorithms. We examined the performances of five algorithms (they are based on different thinning models) on a huge picture set, and sorted the algorithms according to the results they produced. This analysis can be useful for those who would like to apply the most efficient algorithm for a special kind of noisy image.

1. INTRODUCTION

The necessity of designing skeletonization algorithms dates back to the early years of computer technology, to the 1950s. It was realised that in some applications (the first problem was the character recognition), it is enough to take only a reduced amount of information into account instead of the whole image, which is usually a line-drawing. The basic idea was to "peel" the original picture by iteratively removing certain contour points. This procedure is the so–called skeletonization, which creates a line-like shape (the skeleton), so the further analysis becomes easier to execute. The skeleton has the following advantages: there is less information to process, and the shape analysis can be made more easily. Since then many new challenges have occurred from several parts of life, and now skeletonization is applied in a very wide range, e.g., in the analysis of blood cells or chromosome shapes in medical science, or in identifying signatures and fingerprints. Many papers have been published to take a survey of the skeletonization processes without going into details, see [1,4,7]. These articles make the reader familiar with the most important concepts of skeletonization.

A lot of algorithms have been developed and implemented during the past ten years to find the skeletons of different images. It is very difficult to measure the "goodness" of such a method quantitatively. The analytical comparison of the methods is very sophisticated, since they are based on different models, cf. [8,

1991 *Mathematics Subject Classification.* 68U10 Image Processing.

p. 263]. That is the reason why the skeletonizations are compared according to the results they produce in the practice. There are papers about the technical parameters of these algorithms (like computation speed, memory requirement, etc.), and there are observations based on the result skeletons the algorithms produced. A possible way to classify the algorithms is to examine if the result skeletons meet the following (natural) conditions:

- The skeleton should accurately reflect the shape of the original image
- The topological properties (homotopy) of the object and the background should be preserved
- The thickness of the skeleton should be one pixel
- The skeletonization should preserve symmetry
- The skeletonization process should be immune to noise

A more detailed description about the requested properties of skeletons can be found in [8, p. 239].

Usually a reference skeleton is composed, and then the distance of the result and the reference skeleton is calculated by using a suitable distance function. The reference skeleton can be obtained for example, by asking humans to select the skeleton of the object and then averaging the selected skeletons. An interesting way of selecting the reference skeletons can be found in [8, p. 283], where a lot of humans were involved in the creation of the reference skeletons. The most frequently used distance functions are introduced in Section 2. Other functions which are useful to investigate the similarity of the result and the reference skeletons can be found in [8, p. 283]. In the case of using several distance functions, strong correlation can be measured between the distances. Few methods divide the skeletons into fragments, and use polygons to match the result skeletons to the reference instead of using these distance functions (see [8, p. 307]).

The aim of this paper is to analyse the "goodness" of skeletonizations from a special point of view. We found that the skeletonizations were not examined statistically (with a large number of experiments) to test how the noise corruption affects the extraction of the skeleton, and how the skeletonization processes can cope with noisy images. Unfortunately the input pictures are rarely ideal, but are corrupted with some kind of noise. For example, the contours in the image of a printed circuit board are often corrupted by a contour noise, which makes the contours disconnected, thicker, etc. Our purpose was to decide which is the most efficient algorithm for noisy images, among the investigated ones.

2. BASIC CONCEPTS AND NOTATIONS

In the following we need some concepts, so we give the most important definitions that are used.

A *binary picture* can be represented by a 2D-array of points which have the value either 0 or 1 (either white or black). The *background* is the set of the white points and the *object* is the set of the black points.

A *contour (edge) point* is an object point with at least one background neighbour. The *contour (edge)* consists of all the contour points of the object.

A *thinning algorithm* operates on the contours of the objects and eliminates contour points chosen by some kind of neighboring considerations. The common method is to remove iteratively all the contour points of the object except those points which belong to the skeleton. This type of thinning is the contour sequential algorithm, which is based on contour tracing.

Another possibility is to use a template of a given size to scan the picture, and remove the points that belong to an edge but not to the medial axis of the object. The procedure stops when no more changes are made.

To test the similarity (i.e., to calculate the distance) of the result and the reference skeletons one can use the Hamming-distance, which has the following form:

$$X(T,R) = \sum_{i=1}^{m}\sum_{j=1}^{n}(T_{ij}\,\mathrm{XOR}\,R_{ij}), \tag{1}$$

where pictures T and R of size $m \times n$, contain the test and the reference skeleton, i, j are pixel coordinates (ij means the position of the pixel in the ith row and jth column) and XOR means the logic operator exclusive OR. This function computes the value of the XOR operator according to the result and the reference skeleton.

Another useful function is the distance

$$D_1(T,R) = \frac{1}{N_T}\sum_{i=1}^{m}\sum_{j=1}^{n}\sqrt{d(T_{ij},R)} + \frac{1}{N_R}\sum_{i=1}^{m}\sum_{j=1}^{n}\sqrt{d(T,R_{ij})},$$

with

$$d(T_{ij},R) = \begin{cases} 0 & \text{if } T_{ij} = 0; \\ \min_{\{u,v\,|\,R_{uv}\neq 0\}}\{(u-i)^2 + (v-j)^2\} & \text{if } T_{ij} \neq 0 \end{cases}$$

where $1 \le u \le m$, $1 \le v \le n$, and N_T and N_R are the numbers of the object points in the test and the reference pictures, respectively. Thus this function computes the mean Euclidean distance of the pixels of the test and the reference skeletons.

We corrupt an picture with *additive noise* if we change background points to object points (the number of object points increases), and we generate *subtractive noise* when object points become background points (the number of the object points decreases).

The noise is *global* if the whole picture is involved, and we talk about *contour noise* when only the object contours become "noisy". A contour noise can be generated by eliminating contour points (subtractive noise) or by adding contour points from the background (additive noise). Additive and subtractive noise can be generated for a picture independently.

The *level* of the noise indicates the percentage of the picture points that can be changed during the noise generation.

For example, producing an additive global noise at the level of 50% means that we randomly choose the half of the picture points, and change them to object points. If a chosen point is an object point then it remains unchanged.

We used bold characters in the tables to highlight the minimum values.

3. DESCRIPTION OF THE EXPERIMENT

3.1. **Skeletonizations.** To test the noise sensitivity of the skeletonizations five algorithms were examined. The analysed algorithms are listed below with a brief description about the way they work:

1.: E.S. Deutch's algorithm (DE), cf. [2]

: This classical thinning algorithm has two subcycles and uses 3×3 templates.

2.: T. Pavlidis' algorithm (PA), cf. [6]

: This is a Contour Sequential Algorithm, so only the contour points are processed.

3.: V.K. Govindan's algorithm (GO), see [3]

: This Pattern Adaptive Thinning Algorithm examines the object points along the contour, while an adaptive algorithm adjusts the thinning process to picture shape.

4.: N.J. Naccache and colleague's algorithm (NA), see [5]

: This is a Safe Point Thinning Algorithm which uses 3×3 templates, but the input pictures have to be smoothed first, because the algorithm is very sensitive to the "salt and pepper" (global) noise.

5.: R.-Y. Wu's algorithm (WU), cf. [9]

: This is a One-pass Parallel Thinning which uses 3×4 and 4×3 templates instead of 3×3 ones to avoid excessive erosion during the deletion of edge points. The authors claim that this algorithm produces perfectly 8-connected and *noise insensitive* results. (They are right as we shall see.)

Some of these algorithms are known from the literature as very efficient ("good") ones (e.g., Pavlidis' algorithm). We tried to select algorithms which are based on different models, and try to find out whether a relationship exists between the type of algorithm and the noise sensitivity. In the following we refer to these algorithms by two capital letters, as algorithms DE, PA, GO, NA, WU.

3.2. **Original pictures.** To perform the analysis we used 10 binary pictures; their properties are summarized in Table 3.2:

The images can be grouped as printed circuit boards (#1,#3), pictographs (#6,#7,#8), text information (#4,#5,#9), and sophisticated structured line-drawings(#2,#10), respectively. We found that these kinds of images often occur in practice, that is

TABLE 1. Properties of test pictures involved in the analysis

#	Picture size	Number of object points (Area)	Number of contour points (Perimeter)
1	100 × 100	4014	769
2	200 × 200	8343	5204
3	100 × 100	4326	961
4	320 × 200	14068	4018
5	100 × 100	2366	970
6	320 × 200	3468	2179
7	160 × 160	15007	3132
8	200 × 160	5016	1671
9	200 × 160	7901	2908
10	320 × 200	15833	6477

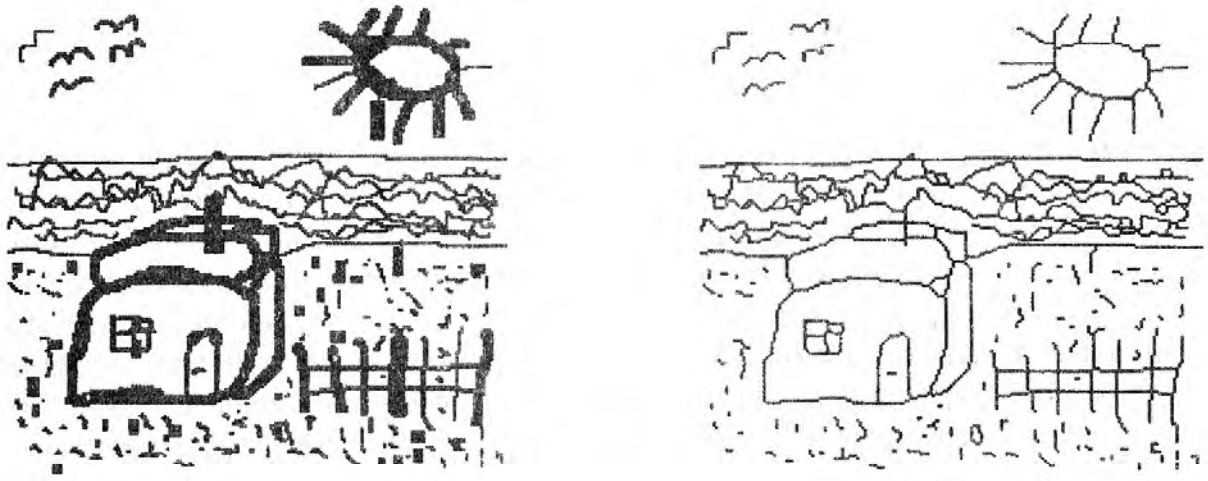


FIGURE 1. Picture #2 (a), and its skeleton (b) extracted by the algorithm DE

the reason why we chose them for this analysis. Figure 1(a) shows the test picture #2, and the result skeleton (extracted by DE) is shown in Figure 1(b).

TABLE 2. Noise corruptions applied to the original pictures

Noise	A	B	C	D	E	F
Level	+2%	+2%/−2%	+10%	−10%	+5%/−10%	+15%/−15%
Type	global	global	contour	contour	contour	contour

3.3. **Generating noise.** We corrupted our test pictures with uniformly distributed noises. Additive (background points become object points) and subtractive (objects points become background points) noises were used at different corruption levels for the whole pictures (global noise) and for only the object contours (contour noise). The levels of corruption are given in percents, a positive percentage values mean additive, negative values mean subtractive noise.

Table 3.3 shows the noise levels and types we generated in the test pictures. The noise corruptions used are denoted by A, B, C, D, E, and F, respectively.

Picture #2 is shown in Figure 2(a) after corrupting with a $+15\%/-15\%$ global noise, and the result of the skeletonization DE is on Figure 2(b). It is worth examining how the noise corruption affected the result of the skeletonization. For example, little circles can be seen on the sun in Figure 2(b), which are missing from Figure 1(b). The reason is that the subtractive noise (−15%) eliminated some points from the main circle of the sun, thus holes appeared there, and the skeletonization preserved the topological features. Little line segments, and points in the sky can be seen as the consequence of the additive noise corruption.

3.4. **Reference skeleton.** There are several ways to find a good reference skeleton for further analysis; one was mentioned in the Introduction. To obtain the reference skeletons for the test pictures we used another method which was simpler and more suitable for our investigations. Our test pictures are artificial (ideal) images, which means that they are not corrupted by noise, so we can consider the result skeleton of a skeletonization in the test picture as the reference skeleton of the given skeletonization. This is a simple way to obtain a reference skeleton, and, on the other hand, we investigate only the effects of noise corruption, so this method does not mean a restriction. For example, we use Figure 1(b) as the reference skeleton of the algorithm DE.

3.5. **Calculating the distance of the result and the reference skeletons.** We obtained 50 skeletons after performing the 5 algorithms for the 10 original pictures and these skeletons were used as references. From the 10 pictures we produced 70 pictures for every kind of noise corruption, thus we had 4200 pictures corrupted by noise.

The 5 skeletonizations were performed for all the 4200 pictures, thus altogether we had 21000 result skeletons in the end.

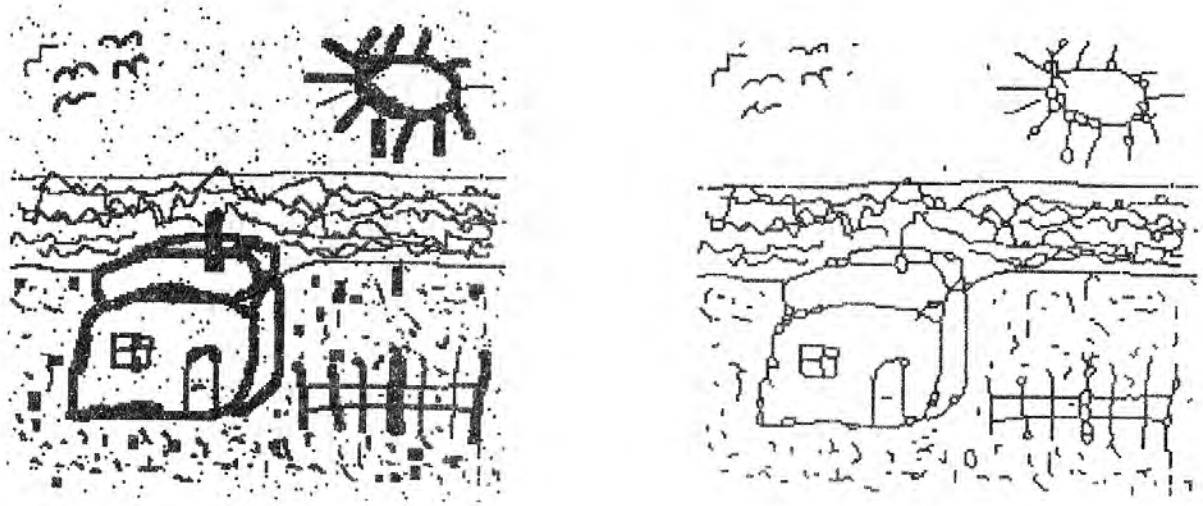


FIGURE 2. Picture #2 after corrupting with a $+15\%/-15\%$ global noise (a), and its skeleton (b) extracted by the algorithm DE

As the next step of our analysis, we compared the result skeletons with the corresponding reference skeletons. To perform this comparison, first we translated the result skeleton both horizontally and vertically to find the best matching to the reference (i.e., to make the result skeleton cover the reference one). We chose the position for which the distance of the result and the reference skeleton was minimal. This technical procedure has to be executed as a preprocession, since the translation of the object implies the translation of the skeleton. For example, an additive/subtractive contour noise corruption translates a solid rectangle by one pixel if the additive part of the noise corrupts all the points of one side of the rectangle and the subtractive part corrupts the points of the opposite side. The maximal translation allowed was 2 pixels in any of the four directions.

After translating the result skeleton, we calculated its Hamming-distance (1) from the reference skeleton. Thus at the end of the calculation we had a database of 21000 experimental distance values, which we could use for further statistical investigations. To evaluate this data set we used the statistical program package SPSS[1].

[1]Copyright © SPSS Inc.

TABLE 3. The mean values of computation times (the values are given in seconds)

#	DE	PA	GO	NA	WU
1	0.428	1.005	0.657	0.513	**0.358**
2	0.779	2.570	1.901	1.276	**0.752**
3	0.421	1.174	0.711	0.570	**0.399**
4	3.092	7.267	4.667	5.728	**2.699**
5	0.178	0.554	0.392	0.265	**0.168**
6	**0.311**	0.983	0.772	0.604	0.312
7	2.338	5.254	3.270	3.108	**2.114**
8	1.111	2.084	1.419	1.738	**0.829**
9	0.741	1.973	1.435	1.166	**0.623**
10	2.757	6.384	5.125	3.866	**2.238**

4. STATISTICAL EVALUATION OF THE RESULT DATA

4.1. **Computation time.** This paper does not focus on the technical parameters of the skeletonizations, but sometimes it can be useful to know how much it takes an algorithm to process a picture corrupted with a special kind of noise. Table 4.1 contains the mean values of the algorithms' computation times for each of the pictures. The computation time certainly depends on the size and on the difficulty level of the picture. From the skeletonizations examined we found the one-pass algorithm WU to be the fastest, according to this table. A table containing computation speed for lots of algorithms can be found in [8, p. 239].

4.2. **Picture – Distance, and Noise type – Distance relations.** We examined the performance of the algorithms with respect to the pictures and to the noise types. Figure 4.2 contains the mean distance values of the test and the reference skeletons for every picture, and the result of the skeletonizations are shown by polygons. Smaller distance values mean better matching to the reference skeletons, so that skeletonization produces the best result which has the smallest distance values (i.e., the lowest polygon).

Figure 4.2 contains information about a similar analysis, but here the noise types were examined instead of the pictures. The same conclusions can be drawn, i.e., that algorithm produces the best result which has the smallest values.

Detailed tables of the distance values can be found in the Appendix. Appendix I contains a table about the expected values (means) of the distance values, while Appendix II shows their standard deviations.

4.3. **Performance indices for skeletonizations.** We assigned a rank value to the algorithms according to the distance values they produced in the given test.

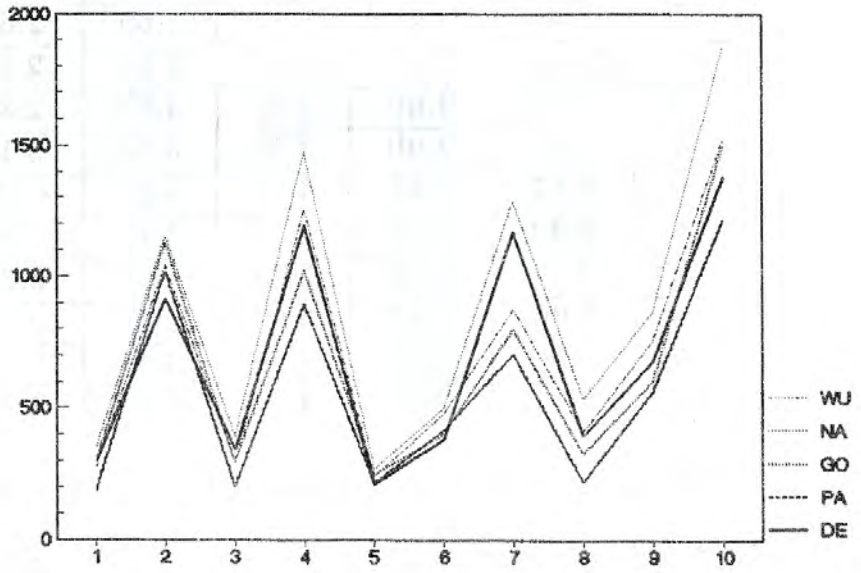


FIGURE 3. Picture – Distance relation

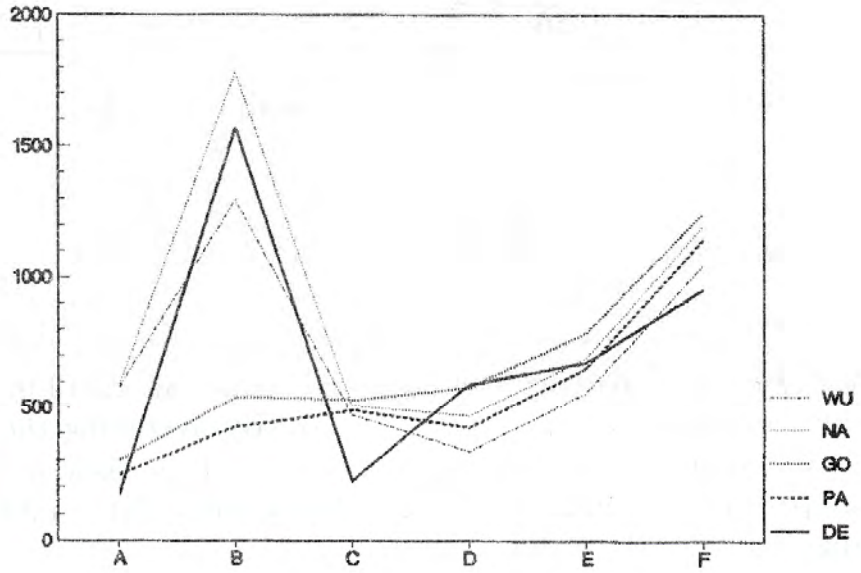


FIGURE 4. Noise type – Distance relation

TABLE 4. Performance indices for the pictures

#	DE	PA	GO	NA	WU
1	2.50	**2.17**	4.17	3.50	2.67
2	**2.00**	3.17	4.00	3.50	2.33
3	2.67	**2.00**	4.00	4.00	2.33
4	2.67	**2.00**	3.50	3.50	3.17
5	**2.17**	2.83	4.17	3.33	2.50
6	**2.33**	3.50	3.17	3.17	2.83
7	2.67	**2.33**	3.33	4.33	**2.33**
8	3.00	**1.83**	3.83	4.17	2.17
9	**2.50**	**2.50**	3.50	3.67	2.83
10	**2.17**	2.33	4.17	4.17	**2.17**

TABLE 5. Performance indices for the noise types

	DE	PA	GO	NA	WU
A	**1.00**	2.10	2.90	4.50	4.50
B	3.80	**1.00**	2.00	4.70	3.50
C	**1.00**	3.30	4.00	3.40	3.30
D	4.60	2.30	4.20	2.90	**1.00**
E	3.20	2.70	4.70	3.40	**1.00**
F	**1.20**	3.40	4.90	3.50	1.90

TABLE 6. Overall performance indices for the algorithms

DE	PA	GO	NA	WU
2.46	**2.46**	3.78	3.73	2.53

The algorithm with the smallest distance value got rank 1, and the largest value got rank 5. Identical distance values got the same rank, and in this case the next rank value was skipped (e.g., 1, 1, 3, 3, 5). By averaging these rank values we were able to assign a performance index to each of the skeletonizations. Table 4.3 shows the performance indices for the pictures, while Table 4.3 shows the same for noise types.

By averaging these performance indices we could calculate an overall performance index for every skeletonization. The Table 4.3 contains these indices:

According to this table we can conclude that algorithms DE, PA, and WU produce better skeleton matching than GO or NA. This result explains why

TABLE 7. Performance indices for pictures (only for contour noise)

#	DE	PA	GO	NA	WU
1	2.25	2.50	5.00	3.25	**2.00**
2	2.00	4.00	4.75	2.75	**1.50**
3	2.50	2.25	4.75	3.75	**1.75**
4	3.00	2.25	4.00	3.25	**2.25**
5	2.00	3.50	5.00	2.75	**1.75**
6	2.50	4.25	3.75	2.50	**2.00**
7	3.00	2.75	3.75	4.00	**1.50**
8	3.25	2.00	4.50	3.75	**1.50**
9	2.50	3.00	4.00	3.25	**2.25**
10	2.00	2.75	5.00	3.75	**1.50**

TABLE 8. Overall performance indices for the algorithms (only for contour noise)

DE	PA	GO	NA	WU
2.50	2.92	4.45	3.30	**1.80**

the authors of algorithm NA warn for smoothing before skeletonizing, as algorithm NA has the worst results in the case of global noise which has a "salt and pepper" effect.

4.4. **Performance indices only for contour noise corruption.** During our investigations we found that it was a bit unfair to involve global noise corruption into the analyses, as mainly contour noise occurs in practice, and these algorithms are quite sensitive to global noise. That is why we calculated performance indices of the algorithms for only contour noise corruption. The Table 4.4 is similar to Table 4.3, but global noises were excluded from this analysis.

According to the above discussed method, we also calculated the overall performing indices of the algorithms for the contour noise corruptions (Table 4.4).

This table indicates larger difference between the "goodness" of the algorithms. Algorithm WU seems to be the most reliable one (so the authors are right about noise immunity), but DE is better in coping with higher noise level. Algorithm GO produced the worst performance with respect to noise sensitivity.

4.5. **Variance analysis.** In our analysis we grouped the distance values on the basis of the algorithm name as a factor. It turned out that this factor is significant for the model, and the following groups were obtained:

TABLE 9. Pictures grouped according to the skeletonizations

#	1	3	5	6	7	8	9
1		DE, PA, GO	DE, GO, NA, WU			DE, GO	
3	DE, PA, GO		DE, GO, NA, WU			DE, GO, WU	
5	DE, GO, NA, WU	DE, GO, NA, WU					PA, GO
6					PA		
7				PA			PA, GO
8	DE, GO	DE, GO, WU	PA, GO				
9					PA, GO		

- DE
- PA
- GO, NA
- WU

This result shows that algorithms GO and NA produce similar results for any kind of noise and picture type.

We examined every algorithm if it worked similarly when only contour noise corruption was generated in the pictures. In this analysis the distance values were grouped on the basis of the picture number as a factor for all the skeletonizations. The following table shows the cases when at least two pictures belong to the same group. A picture can belong to more than one group (it is known from factor analysis), which means that there are similarities in some details, but this relation is not transitive. Two pictures belong to the same group if the algorithm produces similar results. For example, Pictures #1 and #3 usually belong to the same group (because they have the same type, as both of them are pictures of printed circuit boards). Further similar conclusions can be derived, e.g., Pictures #2 and #10 always form separate groups, as they have sophisticated structure.

4.6. **Regression analysis.** Our statistical investigations discovered strong relationship between the level of the noise corruption, and the distance values. They are correlated at the level of $r = 0.7584$, and the hypothesis that these variables are uncorrelated should be rejected at every normally used significance level (95%, 99%). Moreover, a linear relationship was conjectured and a regression analysis proved this hypothesis. From R statistics we obtained that the linear model is acceptable, and the hypothesis that the linear model is not suitable should be rejected at every normally used significance level (95%, 99%).

5. CONCLUSIONS

This paper presents statistical results about the tolerance of five skeletonization algorithms with respect to noisy images. A large database of 21000 skeletons was used to obtain performance indices for the algorithms. Linear correlation was detected between the level of the noise and the distance of the reference and the test skeletons. The algorithms could be grouped according to their tolerance with respect to different types of noises and images. The calculated rank values of the algorithms may help one to choose an algorithm which produces the most reliable result for a given type of image which is corrupted with a given type of noise. It seems to be interesting to go on with analysing other skeletonizations which are based on other models, or to make investigations in a higher dimension (3D).

APPENDIX I

The Table 5 contains the expected value (mean) of the distance values produced by the algorithms. The column headings represent the generated noise (heading $*$ indicates the case when every picture corrupted with every kind of noise was involved in the analysis). Row numbers represent the test pictures. The smallest distance values are highlighted with bold numbers for every picture–noise pair.

APPENDIX II

The Table 5 contains the standard deviation of the distance values produced by the algorithms. The column headings represent the generated noise (heading $*$ indicates the case when every picture corrupted with every kind of noise was involved into the analysis). Row numbers represent the test pictures. The smallest values are highlighted with bold numbers for every picture–noise pair.

TABLE 10.

#	*	A	B	C	D	E	F
1	296 **185** 295 345 269	**56** 84 126 159 162	988 **170** 250 983 633	**54** 151 238 149 160	181 134 283 159 **109**	207 206 357 229 **189**	**290** 367 518 389 358
2	**914** 1018 1129 1152 1042	**246** 378 479 730 725	1089 **596** 775 1430 1169	**409** 915 884 788 772	959 881 1130 821 **686**	1113 1246 1404 1159 **1041**	**1670** 2093 2102 1980 1860
3	343 **201** 295 394 291	**52** 72 104 158 151	1048 **177** 262 1017 643	**64** 159 232 211 188	252 141 250 196 **128**	275 232 358 291 **223**	**367** 426 565 488 414
4	1192 **896** 1021 1474 1249	306 439 498 1175 1181	2925 **749** 905 3290 3530	**344** 785 819 828 852	971 598 829 680 **431**	1099 991 1183 1054 **847**	**1505** 1815 1893 1815 1652
5	**211** 217 247 271 243	**64** 94 117 182 192	359 **147** 180 442 348	**88** 190 207 162 189	197 179 231 181 **134**	226 261 301 245 **214**	**334** 428 449 411 384
6	**380** 416 403 501 478	**126** 184 176 483 487	406 **268** 280 693 643	**185** 370 293 318 335	400 347 399 304 **265**	469 503 507 449 **413**	**692** 825 765 760 727
7	1172 **706** 801 1288 878	123 269 324 484 403	1633 **661** 830 3488 2235	**302** 727 810 808 538	818 410 392 493 **290**	934 749 844 857 **570**	**1223** 1421 1605 1598 1234
8	392 **219** 326 534 409	**118** 133 192 580 570	993 **221** 310 1279 984	60 125 202 225 166	337 148 288 224 **110**	374 238 367 325 **206**	470 447 594 570 **417**
9	681 **564** 605 871 759	**205** 280 314 730 748	1475 **407** 470 1747 1338	**237** 516 478 490 531	566 394 503 **448** **309**	660 641 707 663 **555**	**944** 1148 1156 1146 1075
10	1382 **1219** 1506 1880 1528	**380** 530 654 1156 1142	2788 **878** 1131 3417 2427	**485** 977 1119 1090 991	1197 1010 1482 1171 **875**	1389 1440 1840 1656 **1336**	**2051** 2481 2812 2787 2395

References

[1] Y.-S. Chen, W.-H. Hsu, *A comparison of some one-pass parallel thinnings*, Pattern Recognition Letters, 11 (1990), pp. 35–41.

[2] E. S. Deutsch, *Thinning algorithms on rectangular, hexagonal and triangular arrays*, Communications of the ACM, 15 (1972), pp. 827–836.

[3] V. K. Govindan, A. P. Shivaprasad, *A pattern adaptive thinning algorithm*, Pattern Recognition, 20 (1987), pp. 623–637.

[4] L. Lam, S.-W. Lee, C. Y. Suen, *Thinning methodologies – A comprehensive survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14 (1992), pp. 869–885.

TABLE 11.

#	*	A	B	C	D	E	F
1	323	**17**	71	**17**	29	35	**32**
	97	24	**50**	36	32	**38**	45
	130	38	69	43	48	47	41
	300	18	64	28	29	32	40
	184	19	63	26	**23**	**29**	37
2	477	**28**	85	**29**	55	**56**	62
	556	45	63	69	65	74	84
	530	87	98	92	101	96	69
	448	34	68	43	46	58	59
	409	38	**60**	46	**42**	**56**	**54**
3	337	**16**	68	**19**	29	39	**35**
	118	22	56	34	28	37	49
	152	33	78	44	51	45	61
	302	19	62	27	27	36	42
	186	18	**53**	26	**26**	**33**	42
4	885	**34**	116	**48**	71	**70**	73
	453	57	**103**	80	66	101	104
	447	74	113	91	84	80	93
	892	50	111	64	58	**70**	**72**
	687	48	105	75	**48**	**70**	83
5	115	17	**48**	**18**	32	30	31
	111	23	**28**	27	29	33	29
	110	29	32	32	30	37	32
	117	**16**	39	22	22	**25**	**27**
	95	19	30	27	**20**	29	33
6	190	**18**	46	**21**	38	43	43
	212	37	46	39	43	39	51
	197	34	52	44	49	44	49
	176	25	**41**	26	**27**	38	**32**
	167	25	**41**	36	32	**36**	40
7	1165	**27**	**98**	**45**	70	76	84
	386	122	186	123	110	115	101
	442	148	221	130	75	149	105
	1055	38	**98**	65	49	75	**82**
	681	29	102	69	**38**	**59**	84
8	308	**22**	81	17	42	38	49
	117	26	**54**	32	27	42	41
	142	41	60	45	31	45	51
	366	32	74	31	23	30	40
	305	34	65	33	**19**	**29**	**35**
9	440	**29**	99	**37**	48	49	57
	291	51	**58**	71	61	58	65
	278	52	66	54	56	54	60
	456	35	88	47	**38**	**44**	**45**
	353	35	62	51	39	45	50
10	847	**43**	101	**40**	66	67	**78**
	629	54	**75**	66	73	76	99
	695	70	99	84	94	97	112
	906	44	96	57	63	67	**78**
	645	50	83	63	**56**	**62**	101

[5] N. J. Naccache, R. Shinghal, *SPTA: A proposed algorithm for thinning binary patterns*, IEEE Transactions on Systems, Man, and Cybernetics, 14 (1984), pp. 409–418.

[6] T. Pavlidis, *A thinning algorithm for discrete binary images*, Computer Graphics and Image Processing, 13 (1980), pp. 142–157.

[7] C. Ronse, *A topological characterization of thinning*, Theoretical Computer Science, 43 (1986), pp. 31–41.

[8] C. Y. Suen, P. S. P. Wang (ed), *Thinning methodologies for pattern recognition*, World Scientific Publishing Co., Singapore, 1994.

[9] R.-Y. Wu, W.-H. Tsai, *A new one-pass parallel thinning algorithm for binary images*, Pattern Recognition Letters, 13 (1992), pp. 715–723.

ATTILA FAZEKAS AND ANDRÁS HAJDU

LAJOS KOSSUTH UNIVERSITY, 4010, DEBRECEN PO BOX 12, HUNGARY
E-mail address: {fattila,hajdua}@math.klte.hu

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

SYMBOLIC MODELLING OF DYNAMIC EQUATIONS FOR NON-COMPRESSIBLE STATIONARY FLUIDS

A. ANDREICA

Abstract. This paper deals with automatical implementation of particular theoretical computations which appear in the mechanics of non-compressible stationary fluids. For certain classical types of movements, there were written Mathematica packages which calculate their physical characteristics, such as: velocity, friction tensions, flow, pressure. The results are obtained symbolically, as an expression depending on some variables, but these variables can also be given numerical values, in this case the result will be "partially" or "totally" numerical. Therefore, symbolic computation supports a quick finding of the theoretical desired results, which would otherwise demand a considerable amount of time since they are obtained by differentiations, integrations, and solving quite complicated equations or systems of equations / differential equations.

1. Introduction

Until the last decades, the problems of mathematical physics were solved almost exclusively by numerical methods, therefore no appropriate solution could be found for some of them. Begining with the '70s and especially during the '80s, there took place a huge development of symbolic computation systems for pure mathematics, biology, chemistry, but most of all physics: celestial mechanics, high energy physics, general relativity, electronic optics, molecular physics, fluid mechanics, quantum mechanics [3]. This evolution pursued two directions: building specialized systems for specific domain problems and building applications using the existing general purpose symbolic computation systems (such as MACSYMA, REDUCE, MAPLE, MATHEMATICA) [1]. As some of these systems had an appropiate interface for numerical computations, these types of problems were solved, too.

Dynamic phenomena are described in fluid mechanics by a system of three partial differential equations, which describes the movements on the three axes of

Received by the editors: January 27, 1997.

1991 *CR Categories and Descriptors.* G.4 [Mathematical Software]: Algorithm analysis; J.2 [Physical Sciences and Engineering]: Mathematics and statistics.

coordinates, depending on velocity, density, pressure. Considering the complicated form of these equations, which are known as the Navier-Stokes equations, they cannot be solved in the most general case. From the physical point of view, the sequences of computations which lead to the formulas of velocity, fluid flow and friction tensions - for particular situations - are of interest. Here we enumerate such types of movements: newtonian movements between two parallel plates: without pressure gradient and with o moving plate (Couette movement), with pressure gradient between two immobile plates (Poiseuille movement), with a free surface (laminar movement), combined movement (Couette-Poisseuille); movement in cylindrical pipes and between two circular cylinders; non-newtonian parallel plane movements or in cylindrical pipes; non-compressible stationary movements following concentrical circles, between two coaxal cylinders (with a number of special cases), following concurrent lines or between two plane walls.

For all these types of movements there were written Mathematica packages which contain functions for computing the necessary quantities. The built-in Mathematica functions allow to make a simple description of the complicated operations involved in these computations: differentiations, integrations, and solving quite complicated equations or systems of equations / ordinary differential equations. Moreover, working with Mathematica packages, allows a natural and easy extension of the basic capabilities which are available in a Mathematica session. The variables representing arguments for the newly written Mathematica functions can be symbolical or numerical, therefore influencing the result.

In the following paragraphs, we shall present the physical aspects of the problem, the principles used in writing the Mathematica packages, together with a few ideas in respect with the possibility of expanding the computations and an easier interpretation of the result (finding the type of movement based on some characteristics, a graphical representation of the solutions).

2. The problem from the physical point of view

The physical aspects of the problem, together with their mathematical modelling are presented in fluid dynamics [2].

The study of fluid dynamics is a phenomenal one; fluids are considered continuous and deformable media.

The effect of fluid deformation under a shearing force is continuous; fluids flow. The study of real fluids is based on research performed on models, such as the models of perfect fluid (a homogeneous, deformable, non-resisting medium) and perfect viscous fluid.

The perfect viscous fluid, or the newtonian fluid, doesn't immediately react to an action. Its deformation depends on the duration and on the intensity of the solicitation; when the action stops, the deformation doesn't recover and the consumed mechanical work spreads in the whole mass of the fluid as heat. Unless the solicitation modifies, the deformation continues and the deformation velocity

remains constant. The newtonian fluid has the property of viscosity - it opposes a resistance to a shearing or compression deformation.

The hypothesis that for newtonian fluids, there is a linear relation between the deformation tensions (shearing tensions $\tau_{ij}, i, j \in \{x, y, z\}, i \neq j$ or compression tensions $\tau_{ij}, i \in \{x, y, z\}$ and the deformation velocities, implies the existence of two viscosity coefficients: dynamic η and volumic η_V. Using cartesian coordinates, the relations between deformation tensions and velocities are:

$$\begin{cases} \tau_{xx} &= 2\eta\left(\frac{\partial v_x}{\partial x} - \frac{1}{3}\text{div } v\right) + \eta_v \text{div } v - p \\ \tau_{yy} &= 2\eta\left(\frac{\partial v_y}{\partial y} - \frac{1}{3}\text{div } v\right) + \eta_v \text{div } v - p \\ \tau_{zz} &= 2\eta\left(\frac{\partial v_z}{\partial z} - \frac{1}{3}\text{div } v\right) + \eta_v \text{div } v - p \end{cases} \tag{1}$$

$$\begin{cases} \tau_{xy} = \tau_{yx} &= \eta\left(\frac{\partial v_x}{\partial y} + \frac{\partial v_y}{\partial x}\right) \\ \tau_{yz} = \tau_{zy} &= \eta\left(\frac{\partial v_y}{\partial z} + \frac{\partial v_z}{\partial y}\right) \\ \tau_{zx} = \tau_{xz} &= \eta\left(\frac{\partial v_z}{\partial x} + \frac{\partial v_x}{\partial z}\right), \end{cases} \tag{2}$$

where v, with the cartesian components v_x, v_y, v_z is the velocity, p is the pressure and

$$\text{div } v = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$$

is the divergence of the velocity.

These formulas represent the basis for the mathematical (deductive) study of viscous compressible and non-compressible fluids' movements [2].

The differential equations, in tensions, for newtonian fluid flow in non-stationary, isothermic conditions can be deduced from the equilibrium of: inertia $\rho dv/dt$, exterior ρf and surface $\frac{\partial \tau_x}{\partial x} + \frac{\partial \tau_y}{\partial y} + \frac{\partial \tau_z}{\partial z}$ forces for an element of volume. Under gravitational field, the force corresponding to an element of volume is ρg, where ρ is the density and g - the gravitation. Therefore, in cartesian coordinates we have:

$$\begin{aligned} \rho\frac{Dv_x}{Dt} &= \rho f_x + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} \\ \rho\frac{Dv_y}{Dt} &= \rho f_y + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} \\ \rho\frac{Dv_z}{Dt} &= \rho f_z + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}, \end{aligned}$$

where $\frac{D}{Dt} = \frac{\partial}{\partial t} + v_x\frac{\partial}{\partial x} + v_y\frac{\partial}{\partial y} + v_z\frac{\partial}{\partial z}$ is the substantial derivative.

By substituting the formulas for tensions $\tau_{ij}, i, j \in \{x, y, z\}$ (1,2) in these last equations, we obtain the differential equations for newtonian fluid flow depending on the velocity components, known as the Navier-Stokes equations:

$$\begin{aligned}
\rho\left(\frac{\partial v_x}{\partial t}+v_x\frac{\partial v_x}{\partial x}+v_y\frac{\partial v_x}{\partial y}+v_z\frac{\partial v_x}{\partial z}\right) &= \eta\left(\frac{\partial^2 v_x}{\partial x^2}+\frac{\partial^2 v_x}{\partial y^2}+\frac{\partial^2 v_x}{\partial z^2}\right)\\
&+\left(\frac{\eta}{3}+\eta_v\right)\left(\frac{\partial^2 v_x}{\partial x^2}+\frac{\partial^2 v_y}{\partial x\partial y}+\frac{\partial^2 v_z}{\partial x\partial z}\right)-\frac{\partial \rho}{\partial x}+\rho f_x\\
\rho\left(\frac{\partial v_y}{\partial t}+v_x\frac{\partial v_y}{\partial x}+v_y\frac{\partial v_y}{\partial y}+v_z\frac{\partial v_y}{\partial z}\right) &= \eta\left(\frac{\partial^2 v_y}{\partial x^2}+\frac{\partial^2 v_y}{\partial y^2}+\frac{\partial^2 v_y}{\partial z^2}\right)\\
&+\left(\frac{\eta}{3}+\eta_v\right)\left(\frac{\partial^2 v_x}{\partial y\partial x}+\frac{\partial^2 v_y}{\partial y^2}+\frac{\partial^2 v_z}{\partial y\partial z}\right)-\frac{\partial \rho}{\partial y}+\rho f_y\\
\rho\left(\frac{\partial v_z}{\partial t}+v_x\frac{\partial v_z}{\partial x}+v_y\frac{\partial v_z}{\partial y}+v_z\frac{\partial v_y}{\partial z}\right) &= \eta\left(\frac{\partial^2 v_z}{\partial x^2}+\frac{\partial^2 v_z}{\partial y^2}+\frac{\partial^2 v_z}{\partial z^2}\right)\\
&+\left(\frac{\eta}{3}+\eta_v\right)\left(\frac{\partial^2 v_x}{\partial z\partial x}+\frac{\partial^2 v_y}{\partial z\partial y}+\frac{\partial^2 v_z}{\partial z^2}\right)-\frac{\partial \rho}{\partial y}+\rho f_z
\end{aligned} \tag{3}$$

These equations describe the non-stationary isothermal flow of compressible fluids.

The vectorial form of the equations (3) is:

$$\rho\frac{Dv}{Dt}=\eta\triangle v+\left(\frac{\eta}{3}+\eta_v\right)\nabla(\nabla v)-\nabla p+\rho f,$$

where the ∇ and $\triangle$ operators are:

$$\nabla=\frac{\partial}{\partial x}\vec{i}+\frac{\partial}{\partial y}\vec{j}+\frac{\partial}{\partial z}\vec{k},\ \text{(Hamilton operator)}$$

$$\triangle=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2},\ \text{(Laplace operator)}$$

The integration of Navier-Stokes equations (3) can be performed by analytical methods, which generate exact solution (but they often solve only limited particular cases), by numerical methods, which give approximate solutions and by experimental modelling methods, the latter being based on similarity and dimensional analysis. It is of utmost importance for the theoretical researchers to obtain exact, correct, symbolic or numeric results. Further on, we shall deal with this aspect.

3. Implementing the computations into Mathematica packages

Because of the complexity of the Navier-Stokes equations (3), they are solved in certain specific cases.

Thus, the Mathematica packages that were written are to compute, for certain types of movements, by means of the functions they contain, the physical quantities that characterize those movements, such as: velocity, friction tension, flow etc. These functions have a symbolic result, solving the problem from the theoretical point of view (the most important aspect for theoretical researchers). But if numerical arguments are wanted for some or all variables which appear in the computation of a certain quantity (function), they can be used and will generate an appropriate result.

Before presenting some examples, we mention that the numeric direct substitution of a differential or integration variable or of a solution for an equation (system of equations) is not, obviously, possible. To solve this problem, we defined auxiliary functions, which verified whether the argument matches a numeric pattern; if so, we applied a transformation rule upon the symbolic expression, substituting the variable by the numeric specified value. Another possible solution of the problem would have been defining the function as a block with local variables and similar effect.

Examples. A. *Stationary non-compressible fluid movements following concurrent lines. Specific case: movement between two plane walls*

In order to make the computations, we use cylindrical coordinates (x, r, ω), the components of the velocity being: $v_x = 0, v_r = v_r(r, \omega), v_\omega = 0$. From the continuity equation, we have $v_r = f(\omega)/r$, where f is a function depending on r. Navier-Stokes equations are (in cylindrical coordinates) [2]:

$$\begin{cases} 0 = -\frac{\partial p}{\partial x} \\ \rho v_r \frac{\partial v_r}{\partial r} = -\frac{\partial p}{\partial r} + \mu \left[\frac{1}{r} \frac{\partial p}{\partial r} \left(r \frac{\partial v_r}{\partial r} \right) \frac{1}{r^2} \frac{\partial^2 v_r}{\partial \omega^2} - \frac{v_r}{r^2} \right] \\ 0 = -\frac{\partial p}{r \partial \omega} + \frac{2\mu}{r^2} \frac{\partial v_r}{\partial \omega} \end{cases} \tag{4}$$

We intend to find the formula for the pressure p, depending on f, r, η (dynamic viscosity coefficient) and an integration constant $C1$. The expression obtained after reducing equations (4) (integrated in respect to r, ω, respectively), will be named c; therefore we shall obtain the equation $c = 0$. The pressure p (which has the corresponding Mathematica function P) is deduced by integrating equation (6); subsequent to this operation we shall obtain function t, depending only on r. The velocity v_r has the corresponding Mathematica function V_r; all the others are auxiliary functions. Thus we obtain the following Mathematica functions:

```
Vr[r_,w_,f_]:=f[w]/r
A0[r_,w_,ro_,eta_,p_,f_]:=Integrate[ -ro Vr[r,w,f] D[Vr[r,w,f],r]-D[p[x,r,w],r]+
   eta(1/r D[r D[Vr[r,w,f],r],r]+D[Vr[r,w,f],{w,2}]/r^2-Vr[r,w,f]/r^2),r]
Axx0[r_,w_,ro_,eta_,p_,f_]:=If[ MatchQ[r,n_Integer] || MatchQ[r,n_Real],
      A0[rr,w,ro,eta,p,f] /. rr->r, A0[r,w,ro,eta,p,f]]
Ax0[r_,w_,ro_,eta_,p_,f_]:=If[ MatchQ[w,n_Integer] || MatchQ[w,n_Real],
      Axx0[r,ww,ro,eta,p,f] /. ww->w, Axx0[r,w,ro,eta,p,f]]
A1[r_,w_,eta_,p_,f_]:=Integrate[-D[p[x,r,w],w]/r + 2 eta/r^2 D[Vr[r,w,f],w],w]
Ax1[r_,w_,eta_,p_,f_]:=If[ MatchQ[w,n_Integer] || MatchQ[w,n_Real],
      A1[r,ww,eta,p,f] /. ww->w, A1[r,w,eta,p,f]]
c[w_,ro_,eta_,p_,f_]:=Simplify[Numerator[Simplify[
      -Ax0[r,w,ro,eta,p,f] + r Ax1[r,w,eta,p,f]]]/eta]
Axp0[r_,w_,eta_,f_,t_]:=Integrate[2 eta/r^2 f'[w],w]+t[r]
Axp[r_,w_,eta_,f_,t_]:=If[ MatchQ[w,n_Integer] || MatchQ[w,n_Real],
      Axp0[r,ww,eta,f,t] /. ww->w, Axp0[r,w,eta,f,t]]
Axt0[r_,w_,ro_,eta_,f_,t_]:=Simplify[(-ro Vr[r,w,f] D[Vr[r,w,f],r] -
```

```
    D[Axp[r,w,eta,f,t],r] + eta(1/r D[r D[Vr[r,w,f],r],r] +
    D[Vr[r,w,f],{w,2}]/r^2 - Vr[r,w,f]/r^2)) r^3/eta] (* ==0 ==>t *)
Axt1[r_,w_,ro_,eta_,f_,t_]:=If[ MatchQ[w,n_Integer] || MatchQ[w,n_Real],
      Axt0[r,ww,ro,eta,f,t] /. ww->w, Axt0[r,w,ro,eta,f,t]]
Axt[r_,w_,ro_,eta_,f_,t_]:=If[ MatchQ[r,n_Integer] || MatchQ[r,n_Real],
      Axt1[rr,w,ro,eta,f,t] /. rr->r, Axt1[r,w,ro,eta,f,t]]
ec2p[r_,w_,ro_,eta_,f_,t_]:=Simplify[ Numerator[ Simplify[
     Ax0[r,w,ro,eta,p,f] /. p[x,r,w] -> Axp[r,w,eta,f,t]]] / (-eta)]
taux[r_,w_,ro_,eta_,p_,f_,C1_]:=Simplify[((t[r] /. Part [ Solve[
     Integrate[(c[w,ro,eta,p,f]-Axt[r,w,ro,eta,f,t]) eta/r^3, r]
     ==Integrate[C eta/r^3,r] ,t[r]],1]) + C1) /. C->c[w,ro,eta,p,f]]
t[r_,w_,ro_,eta_,p_,f_,C1_]:=If[ MatchQ[r,n_Integer] || MatchQ[r,n_Real],
taux[rr,w,ro,eta,p,f,C1] /. rr->r, taux[r,w,ro,eta,p,f,C1] ]
P[r_,w_,ro_,eta_,f_,t_,C1_]:=Simplify[ Axp[r,w,eta,f,t] /. t[r]->t[r,w,ro,eta,p,f,C1]]
```

B. *Combined Couette-Poiseuille movement of newtonian media, with pressure gradient.*

Suppose the fluid flows between two parallel plates, one of them moves by velocity V, there is pressure gradient and a nucleus with $hb - ha$ thickness, which moves by constant velocity `vxc`. We use cartesian coordinates (x, y) and the velocity has only the component $v_x(y)$.

Velocity profile is given by Navier-Stokes equation [2]:

$$\frac{d^2 v_x}{dy^2} = \frac{1}{\eta}\frac{dp}{dx}$$

on the intervals $0 \leq y \leq ha, ha \leq y \leq hb$ and $hb \leq y \leq h$, with $v_x(0) = V$, $v_x(ha) = vxc$, $v_x(hb) = vxc$ and $v_x(h) = 0$. The friction tension τ_{xy} is given by:

$$\tau_{xy} = \mp\tau_0 + \eta\frac{\partial v_x}{\partial y},$$

where $\frac{dp}{dx}(ha - hb) = -2\tau_0$ and the flow has the usual formula [2]:

$$q_x = \int_0^x v_x dx.$$

We intend to calculate the velocity for the three intervals (`Vx1`, `Vx2`, `Vx3` in the Mathematica functions), the friction tension τ_{xy} for the intervals $0 \leq y \leq ha$ ($Tensxy1$) and $hb \leq y \leq h$ (`Tensxy3`) and the flow q_x (in Mathematica `Qx`). Thus we obtain the following Mathematica functions:

```
Aux1[y_,V_,ha_,eta_,p_,vxc_]:=Simplify[Collect[Part[vx[y] /.
     DSolve[{eta vx''[y]==p,vx[0]==V,vx[ha]==vxc},vx[y],y] ,1],{p,ha}]]
Aux3[y_,h_,hb_,eta_,p_,vxc_]:=Simplify[Collect[Part[vx[y] /.
     DSolve[{eta vx''[y]==p,vx[h]==0,vx[hb]==vxc},vx[y],y] ,1],{p,vxc}]]
Vx1[y_,V_,ha_,eta_,p_,vxc_]:=
     If[ MatchQ[p,_'[_]],
 If[ MatchQ[y,n_Integer] || MatchQ[y,n_Real],
      Aux1[yy,V,ha,eta,p,vxc] /. yy->y,
      Aux1[y,V,ha,eta,p,vxc]],
```

```
 If[ (MatchQ[p,m_Integer] || MatchQ[p,m_Real] || MatchQ[p,n_Symbol])
      && (MatchQ[y,n_Integer] || MatchQ[y,n_Real]),
 Aux1[yy,V,ha,eta,p,vxc] /. yy->y,
 Aux1[y,V,ha,eta,p,vxc]]]
Vx2[vxc_]:=vxc
Vx3[y_,h_,hb_,eta_,p_,vxc_]:=
      If[ MatchQ[p,_'[_]],
 If[ MatchQ[y,n_Integer] || MatchQ[y,n_Real],
       Aux3[yy,h,hb,eta,p,vxc] /. yy->y,
       Aux3[y,h,hb,eta,p,vxc]],
 If[ (MatchQ[p,m_Integer] || MatchQ[p,m_Real] || MatchQ[p,n_Symbol])
      && (MatchQ[y,n_Integer] || MatchQ[y,n_Real]),
 Aux3[yy,h,hb,eta,p,vxc] /. yy->y,  Aux3[y,h,hb,eta,p,vxc]]]
Tensxy1[y_,V_,ha_,eta_,p_,vxc_]:=Simplify[Collect[
      If[ MatchQ[y,n_Integer] || MatchQ[y,n_Real],
Simplify[-eta D[Vx1[yy,V,ha,eta,p,vxc],yy]] /. yy->y,
Simplify[-eta D[Vx1[y,V,ha,eta,p,vxc],y]]],{eta,p}]]
Tensxy3[y_,h_,hb_,eta_,p_,vxc_]:=Simplify[Collect[
      If[ MatchQ[y,n_Integer] || MatchQ[y,n_Real],
Simplify[-eta D[Vx3[yy,h,hb,eta,p,vxc],yy]] /. yy->y,
Simplify[-eta D[Vx3[y,h,hb,eta,p,vxc],y]]],{eta,p}]]
Qx[V_,h_,ha_,hb_,eta_,p_,vxc_]:=Simplify[Apart[
      Integrate[Vx1[y,V,ha,eta,p,vxc],{y,0,ha}]+Integrate[vxc,{y,ha,hb}]+
      Integrate[Vx3[y,h,hb,eta,p,vxc],{y,hb,h}]]]
```

Subsequent to these calculations, we should consider the system of three equations obtained from the Navier-Stokes equation for the three specified intervals as a system with the unknown variables vxc, ha, hb depending on dp/dx and (0 and calculate their expressions.

An extension of the system of packages described above could be made by introducing input data as sets of numeric values characterizing the evolution of a quantity (velocity, for example), the expected result being the type of movement. Moreover, for an intuitive interpretation of the results, we can include in the packages functions which create a graphic representation of the expressions obtained for the computed quantities; this would not be much trouble considering Mathematica's facilities for drawing function plots.

References

[1] B. Buchberger, G. E. Collins, R. Loos, R Albrecht (ed.), *Computer Algebra and Symbolic Computation*, Springer Verlag, Berlin, 1982.

[2] V. N. Constantinescu, *Dinamica fluidelor vscoase n regim laminar*, Ed. Academiei, 1987.

[3] P. Gianni (ed.), *Symbolic and Algebraic Computation* (Proceedings), ISSAC '88 Springer-Verlag, Rome, 1989.

A. ANDREICA

"EMIL RACOVIŢĂ" THEORETICAL HIGH-SCHOOL, CLUJ-NAPOCA

E-mail address: Ghergari@hera.ubbcluj.ro

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

ACTIVE LEARNING FOR IMPROVING THE PERFORMANCES OF NEURAL NETWORKS

CĂLIN ENĂCHESCU

Abstract. Neural networks learn, they absorb experience, and modify their internal structure in order to accomplish a given task. For a neural network you do not need to find "rules" that characterise a given task. From this point of view, the approximation of a function is equivalent with the problem of training a neural network. In other words, to approximate a function is equivalent to synthesise an associative memory that generates the appropriate output when an input is presented at the input layer and generalises correctly when a new input is presented at the input layer. In the classical forms of supervised learning, the training set is chosen according to some known or random given distribution. The trainer is a passive agent in the sense that he is not able to interact with the training set in order to improve the performances of the neural networks learning. We will investigate some possibilities that allow the trainer to become active and we will analyse the performances of such supervised learning.

1. Introduction

The main feature of the neural computing is the learning capability. Learning is a general concept that must be studied from a mathematical perspective. In this paper we will focus on supervised learning, where we have a "trainer" that provides the desired responses for the neuronal network, when an input is presented to the neural network. In this way we will have a training set $T = \{(\mathbf{x}_i, \mathbf{z}_i) | i = 1, 2, \ldots, N\}$, where $\mathbf{x}_i$ is the input and $\mathbf{z}_i$ is the desired output of the neural network provided by the trainer.

Usually, the trainer has a passive role, being capable to indicate only the desired output values $\mathbf{z}_i$, without having any role in indicating where should be chosen the training samples. In the training process we encounter regions where the learning process is "difficult" or "easy" related to some error measure. In order to improve the performances of the supervised learning process of a neural network, the trainer should give a bigger attention to the "difficult" regions, by

Received by the editors: July 23, 1997.

1991 *CR Categories and Descriptors.* I.2.6 [Artificial Intelligence]: Learning – connectionism and neural nets.

choosing more training samples from that regions. In this way the trainer's roles become more active, having a dynamic influence in the learning process.

2. Mathematical aspects of supervised learning

The supervised learning of the neural networks uses a training set has the following form:

$$T = \{(\mathbf{x}_i, \mathbf{z}_i) | i = 1, 2, \ldots, N\} \tag{1}$$

where $\mathbf{x}_i \in \mathbf{R}^n$ is the n-dimensional input vector, and $\mathbf{z}_i \in \mathbf{R}^m$ is the m-dimensional target vector that is provided by a trainer. $N \in \mathbf{N}$ is a constant that represents the number of training samples. Usually the training set T is obtained from a probabilistic known distribution. In the classical supervised learning strategy [7] the trainer is a static agent. Using the probabilistic distribution he selects a certain input vector $\mathbf{x}_i$, and provides the appropriate target vector $\mathbf{z}_i$. The learning algorithm will compute the difference between the output generated by the neural network $\mathbf{y}_i$ and the desired target vector $\mathbf{z}_i$, which will represent the error signal:

$$e_i = y_i - z_i, i = 1, 2, \ldots, N \tag{2}$$

The signal error is used to adapt the synaptic weights w_{ji} using a gradient descendent strategy [7]:

$$w_{ji} = w_{ji} + \eta \frac{\partial E}{\partial w_{ji}} \tag{3}$$

where $\eta \in (0, 1)$ is the learning rate, controlling the descent slope on the error surface which is corresponding to the error function E:

$$E = \frac{1}{2} \sum_{i=1}^{N} (y_i - z_i)^2 \tag{4}$$

Let us consider a physical phenomena described by a vector $\mathbf{x} \in \mathbf{R}^n$ which corresponds to a set of independent random variables, and a real[1] number $z \in \mathbf{R}$ that represents a dependent variable. Let us consider that we have N distinct measurements (observations) of variable $\mathbf{x}$:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_N \tag{5}$$

[1] A neural network with m output neurons can be considered as m distinct neural networks with only one output neuron. This is why it is allowed to consider, without loosing the generality, a neural network with a single output neuron instead of a neural network with m output neurons. In conclusion, we are allowed to consider, when it is necessary $y, z \in \mathbf{R}$ instead of $\mathbf{y}, \mathbf{z} \in \mathbf{R}^m$.

And a corresponding set of scalars z:

$$z_1, z_2, z_3, \ldots, z_N \tag{6}$$

Usually we do not have enough information about the exact relationship that exist among the variable $\mathbf{x}$ and z. For this reason we will consider a relation represented by the following equation:

$$z = f(\mathbf{x}) + \varepsilon \tag{7}$$

where f is a function which depends on the variable $\mathbf{x}$, and ε is the error represented by a random variable. The error ε represents the mistake made in order to estimate the existing functional relation between variables $\mathbf{x}$ and z. Equation (7) is a known statistical model, named regressive model. Using statistical relations [22], we are now able to express the function f of the regressive model as:

$$f(\mathbf{x}) = E[z|\mathbf{x}] \tag{8}$$

where $E[z|\mathbf{x}]$ represents the conditional statistical average, namely, we will have on average the target value z if he has a particular realisation of variable $\mathbf{x}$. In particular, if the functional relation between variables $\mathbf{x}$ and z is known precisely, then we can consider in the regressive model the ideal case $\varepsilon = 0$.

A neural network is a physical mechanism to implement what we have stated in the regressive model: the prediction of z on the bases of $\mathbf{x}$. This main goal is achieved by encoding the information content in the training set (1) in the synaptic weights w_{ji}. It is quite clear that from the neural computing point of view $\mathbf{x}$ represents the input vector presented at the input layer, and z represents the target vector that we wish to obtain at the output layer of the neural network.

We will note with $\mathbf{w}$ the synaptic weights vector of the neural network that is supposed to approximate the regressive model (7). By applying the input vector to the input layer of the neural network, and by propagating it through the output layer we can write the following equation [2]:

$$y = F(\mathbf{x}, \mathbf{w}) \tag{9}$$

Because the training set $T = \{(\mathbf{x}_i, z_i)|i = 1, 2, \ldots, N\}$ contains also the target vectors z provided by the trainer it is clear the equivalence with the supervised learning paradigm. For this reason the modification of the synaptic weights is done using an iterative process, as a response to the error signal (2).

The supervised learning algorithm will optimise the following error function, in respect with the synaptic weights $\mathbf{w}$ of the neural network:

$$\mathbf{E}(\mathbf{w}) = \frac{1}{2}E[e^2] = \frac{1}{2}E[(z - y)^2] = \frac{1}{2}E[(z - F(\mathbf{x}, \mathbf{w}))^2] \tag{10}$$

This mathematical background related to supervised learning would not be complete if we do not outline the fact that a neural network is a universal approximator of any continuos functions [3, 4, 8, 9]. The architecture of such a

neural network is equivalent to a multylayer perceptron having three layers (an input layer, an output layer and a hidden layer). The activation function of the neurons in the hidden layer must be any non-polynomial function and the activation function of the neuron (neurons) in the output layer can be a linear function or an average function of the output values of the hidden neurons [10]. From this point of view a neural network is an approximation scheme that permits us to approximate at any accuracy any continuos function, provided we have an enough number of hidden neurons [1, 5, 8]. The approximation is obtained through the supervised learning process, which is based on an iterative modification of the synaptic weights of the neural network. Presenting repetitively the training set we will be capable to attain a good generalisation power with the neural network.

3. Active supervised learning

In our presentation of the supervised learning we have outlined the equivalence between the statistical regressive model and the supervised learning of a feedforward neural network. The trainer has a passive role of in the learning process being a simple recipient of passive information about the target function (function to be approximated). 7 We want to determine, if we can consider a more active role for the trainer in the learning process, so, instead of giving only the target vector z for a specific input vector $\mathbf{x}$, to try to indicate which input vector should be selected from the training set, in order to improve the learning capabilities of the neural network, which is equivalent to approximate better with the neural network F the target function f. We can consider that for a specific target function f we have some areas where the function is more "difficult" to be learned (approximated) and so the trainer should choose more examples in order to reduce the approximation error.

In conclusion, we can speak about "difficult" and "easy" regions where the target function is approximated. A "difficult" region will be considered a region with a high approximation error and an "easy" region will be considered a region with a low approximation error (close to zero). This definition is not a very rigorous one because we did not establish the limit that delimits "high approximation error" from "low approximation error". We will see in the next pages that these definitions are not so important, because from the learning algorithm that we will consider, "high approximation error" will be considered the maximum error obtained on the regions which constitute the definition domain of the target function f.

It is obvious that our active learning is based on a fundamental assumption: the trainer is allowed to choose his own examples in order to accomplish the task of approximating the target function as well it is possible. In this assumption

the trainer should be capable to decide which are the "difficult" regions to approximate the target function and to pick up a learning sample from that "difficult" region.

In our analyse we will compare passive supervised learning with the active supervised learning, keeping unchanged the other parameters that influence the learning process. In this approach the only difference between passive and active learning consists only in the way the learning examples are chosen. Another goal of our paper will be to develop a general frame for choosing the examples for the approximation of real functions. We will make also some simulations in order to prove the validity of the theoretical results presented here.

We will need to introduce the following terms:

• $\mathcal{F}$ the set of functions defined on set D with values in set Y, where $Y \subseteq \mathbf{R}$,

$$\mathcal{F} = \{f : D \subseteq \mathbf{R}^n \to Y \subseteq \mathbf{R}\} \tag{11}$$

The target function f that should be approximated by an approximation scheme (a neural network) belongs to the set $\mathcal{F}$ of functions.

• The training (learning) set T is composed by pairs of elements:

$$T = \{(\mathbf{x}_i, z_i) | \mathbf{x}_i \in D, z_i = f(\mathbf{x}_i), i = 1, 2, \ldots, N\} \tag{12}$$

• H is an approximation scheme. This means that H does not contain only a set of functions defined on the set D with values in the set Y, but also the algorithm what the trainer is using to choose the approximator function $F \in H$, based on the learning set T. In other words we will denote by H a couple $< H, A >$, where H is a the set of functions from where we will chose the approximator function F, and A is an algorithm which has as input the learning set T, and generates at the output the approximator function $F \in H$.

• d_C will represent a metric that measures how good is the approximation made by the trainer. More precisely, the metrics d_C measures the error on a subset $C \subseteq D$. This metrics will have the following properties:

  • $\forall C_1, C_2 \subset D, C_1 \subset C_2, d_{C_1}(f_1, f_2) \leq d_{C_2}(f_1, f_2)$;

  • $d_D(f_1, f_2)$ is the distance between two functions on the whole definition set D; it represents the basic criterion to measure the quality of approximation.

• C represents a partition of domain D. We will suppose that all the data points from domain D, which will be chosen to approximate the target function f, partition the domain D in a number of disjoint sets $C_i \in C, \bigcup_{i=1}^{N} C_i = D$.

The trainer's main objective can be stated as follows: operating with an approximation scheme H, based on the learning set T, obtain the approximator function $F \in H$ of the target function f.

In the literature the most widely used criteria to measure the performance of the learning algorithms is the PAC criteria (**P**robably **A**pproximately **C**orrect)

[14].

4. Algorithms for active learning

In the previous paragraphs we have introduced the concepts of passive supervised learning. As we have mentioned, in this case of passive supervised learning, the trainer is supposed to choose his training samples according to some probabilistic distribution defined on domain D. If the passive learning process will be successful, the neural network implemented to learn the training samples will correspond to an approximator function F, so that we have to obtain the relation $d_C(F, f) < \varepsilon$ with a probability greater than $1 - \delta$.

As an alternative, in the active learning the training will have the possibility to choose according to some strategy the training examples from the domain D where the target function f is defined. At a certain moment in learning process, the training set will contain some valuable information about the target function f that has to be approximated by the means of the neural network. Particularly, the training set contains information about the "difficult regions" to be learned, where there is a "high" approximation error. Of course, the trainer will choose more examples in this "difficult region", in order to decrease the total approximation error. In conclusion, we have to develop a learning strategy that will be active in the sense that the trainer can decide which will be the next training sample in the learning process.

First, let us establish the mathematical arguments that describe the mechanism of active learning.

Considering the domain D, the trainer can access all the data from the following general training set:

$$T = \{(\mathbf{x}_i, z_i) | \mathbf{x}_i \in D, z_i = f(\mathbf{x}_i), i = 1, 2, \ldots, N\} \tag{13}$$

The approximation schema H (the neural network), after the learning process, will generate an approximator function $F \in H$, using a learning algorithm A that corresponds in the best way to the training set.

We will use also the following notations:

• $C = \{C_1, C_2, \ldots, C_p\} C_i \subset D, i = 1, \ldots, p_N$[2], a partition of domain D;

$$\mathcal{F}_T = \{f \in \mathcal{F} | f(\mathbf{x}_i) = z_i, \forall (\mathbf{x}_i, z_i) \in T\} \tag{14}$$

The functions belonging to the class of functions $\mathcal{F}_T$ are the functions that are passing through the points of the training set T. Evidently, the target function is a member of the set $\mathcal{F}_T$.

[2]The number p of regions in which the domain D is partitioned by N points depends on the specific geometry of domain D. For example, if D is a real interval then $p = N + 1$

We are now capable to define the following error criteria [11]:

$$e_C(H, T, \mathcal{F}) = \sup d_C(F, f), f \in \mathcal{F}_T \tag{15}$$

The meaning of this error is very important. $e_C(H, T, \mathcal{F})$ measures the maximum error of the approximation schema (neural network) on the region C. This error is dependant on the training set and on the class of functions to which the target function belongs. As we can see it does not depend directly on the target function (function to be approximated), but we cannot forget that this dependency is already captured in the training set, so the target function is present in the above equation.

In this moment we have a certain measurement schema of the uncertainty on different regions of the domain D. In other words, we have now the possibility to define what is a "difficult region" for learning. From now on, we will consider a **difficult region for learning** a region C_i that has the biggest error according to equation (16).

In this way, we have a natural approach for active learning:

Choose the next training sample from the difficult region for learning.

Let us suppose we define the procedure that gives us the possibility to choose the next training sample from the most difficult region for learning with $\mathcal{P}$. This procedure can be very simple:

Procedure $\mathcal{P}$: Choose the sample as the gravity centre of region C_i that is the most difficult region for learning.

Of course this procedure can be adapted to the wishes of the trainer and to the particular form of the target function f.

If we are approximating a one dimensional real function, as we have seen before a region is an interval $C_i = [x_i, x_{i+1}]$, then the next training sample will be:

$$x_{new} = \frac{x_i + x_{i+1}}{2} \tag{16}$$

We have now a possible active strategy for supervised learning. Let us suppose that at one moment the trainer has obtained the new training sample $\mathbf{x}_{new} \in D$. The next thing the trainer will want to know will be the value of the target function in this point. This value will belong to the following data set:

$$\mathcal{F}_T(\mathbf{x}) = \{f(\mathbf{x}) | f \in \mathcal{F}_T\} \tag{17}$$

If the requested value is $z \in \mathcal{F}_T(\mathbf{x})$, in this moment the trainer has a new supervised training pair $(\mathbf{x}_{new}, z)$ which can be added to the existing training set

T, obtaining a new training set:

$$T^* = T \cup (\mathbf{x}_{new}, z) \tag{18}$$

The approximation schema H can now reconsider the approximator function F^*, on the basis of the new training set T^*. We have:

$$e_C(H, T^*, \mathcal{F}) = \sup(F^*, f), f \in F_{T^*} \tag{19}$$

In this moment the error $e_C(H, T^*, \mathcal{F})$ represents the highest possible error related to the new training set T^*. When we choose the new training sample $\mathbf{x}_{new} \in D$ we do not know if we have the necessary information about the value of the target function in this point. A possible strategy to avoid this problem is to choose the "worst case", namely the value which is producing the highest error if $\mathbf{x}_{new} \in D$ is the new training sample.

With this approach, the total error on domain D will be:

$$\sup_{z \in \mathcal{F}_T(\mathbf{x})} e_D(H, T^*, \mathcal{F}) = \sup_{z \in \mathcal{F}_T(\mathbf{x})} e_D(H, T \cup \{\mathbf{x}_{new}, z\}, \mathcal{F}) \tag{20}$$

Our intention is to obtain the training sample that minimises the maximal error. In this respect the new training sample should be chosen according to the following formula:

$$x_{new} = arg \min_{x \in D} \sup_{z \in \mathcal{F}_T(\mathbf{x})} e_D(H, T \cup \{\mathbf{x}, z\}, \mathcal{F}) \tag{21}$$

Using this strategy we are now able to define the following algorithm for active supervised learning. This algorithm gives the trainer the necessary tool to choose the optimal training samples that will improve the supervised learning performances of the approximation schema (neural network).

Step 1: $j := 1$. Choose the first training sample $(\mathbf{x}_j, z_j)$ according to procedure $\mathcal{P}$.

Step 2: Based on the new training example, partition domain D in the regions $C_1, C_2 \ldots, C_{p_j}$.

Step 3: Compute the errors e_{C_i}, for every $i = 1, 2, \ldots, p_j$.

Step 4: Suppose at **Step j** domain D is partitioned in the regions $C_1, C_2, \ldots, C_{p_j}$. According to procedure P we will choose in the most difficult region for learning the new training point $\mathbf{x}_{new} \in D$. Let us consider the new training sample $(\mathbf{x}_{j+1}, z_j) := (\mathbf{x}_{new}, z)$.

IF $e_D(H, T, \mathcal{F}) < \varepsilon$ **THEN**
{
  $N := j$;
  exit;
}
ELSE
{
  $j := j + 1$;

GOTO Step 2;

}

An important calculation is made in our algorithm to obtain the error over the entire domain $e_D(H, T, \mathcal{F})$. This error represents a measure of the highest possible error made by the approximation schema (neural network) in order to approximate a target function from $\mathcal{F}$, using the training set T. If we want an independent approximation schema we have to minimise the error $e_D(H, T, \mathcal{F})$ relative to all the possible approximation schemes:

$$\inf_H(H, D, F) \tag{22}$$

5. Experiments, simulations and conclusions

The learning process was carried out in two phases [4, 13]:

- Unsupervised learning phase [3] in order to determine the following unknown parameters: $\mathbf{t}_i \in \mathbf{R}^n$ the centres of the clusters of the input data and σ_i the radius of the clusters.
- Supervised learning phase in order to determine the synaptic weights $\mathbf{k}_i \in \mathbf{R}$.

The supervised learning phase was done using three different types of training:

1. **Random passive** - the training set was generated randomly from the domain D.
2. **Uniform passive** - the training set was generated using a uniform distribution on domain D.
3. **Active** - the training set was determined using the active learning algorithm presented earlier in this paper.

The experiments were made in order to approximate the following target function:

$$f : [0, 1] \to \mathbf{R}, f(x) = \left(x - \frac{1}{3}\right)^3 + \frac{1}{27} \tag{23}$$

The training set generated by one of the three methods was presented repeatedly in epochs of 1000, 5000 and 10.000 times.

The rulers situated in the bottom of each figure represent the distribution of the training points.

One can observe in Figures 3 that correspond to the active supervised learning the way in which the training samples are distributed. The difficult regions for learning are those where the training points have a higher density, and in our case these regions correspond to the regions where the target function has a higher slope. The regions where the target function it is easy to be approximated the trainer needs just a few examples. These are the easy regions for learning, and in our case for these regions correspond a very slow slope.

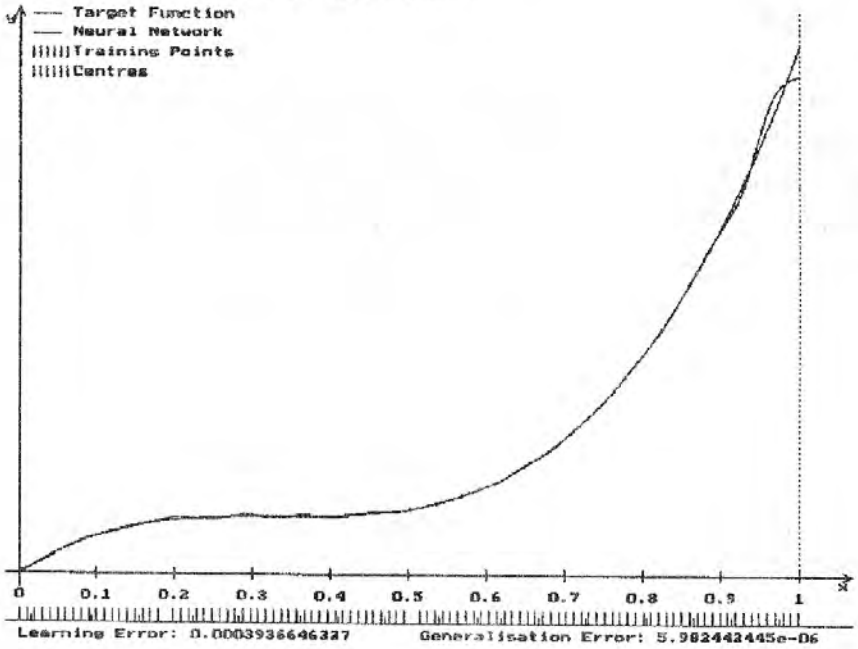


FIGURE 1. Approximation of the target function (24) by an RBF neural network using a random passive supervised learning algorithm: $N = 100$, 10000 epochs, 25 centres

epochs	random passive	uniform passive	active
1.000	$E_l = 0.00111933647$	$E_l = 0.00538671535$	$E_l = 0.005939686434$
	$E_g = 2.00543792\text{e-}5$	$E_g = 9.25674175\text{e-}5$	$E_g = 0.000305306076$
5.000	$E_l = 0.00042799210$	$E_l = 6.77417526\text{e-}6$	$E_l = 8.411126178\text{e-}5$
	$E_g = 6.62271543\text{e-}6$	$E_g = 1.19316687\text{e-}6$	$E_g = 1.167369815\text{e-}5$
10.000	$E_l = 0.00039366463$	$E_l = 5.59375032\text{e-}5$	$E_l = 5.386507373\text{e-}5$
	$E_g = 5.98244244244$	$E_g = 1.01227192\text{e-}5$	$E_g = 5.824087429\text{e-}7$

TABLE 1. Results of the learning process to learn the target function with N = 100 training data samples, 25 centres using random passive, uniform passive and active supervised learning

Analysing the learning performances we will take in consideration not only the learning error E_l but also the generalisation error E_g that correspond to the error generated by the neural networks in points that does not belong to the training set. This generalisation error is the real measure of comparing the performances of different approximation schemes.

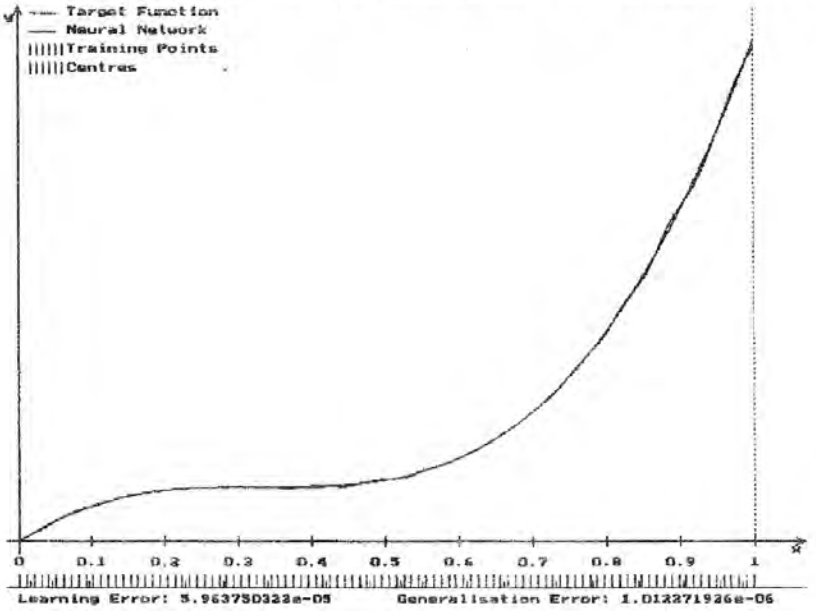


FIGURE 2. Approximation of the target function (24) by an RBF neural network using a uniform passive supervised learning algorithm: $N = 100$, 10000 epochs, 25 centres.

References

[1] Cotter, N.E. (1990). The Stone - Weierstrass Theorem and Its Application to Neural Networks. *IEEE Transactions on Neural Networks*, **4**, 290-295.

[2] Enăchescu. C. (1995) Properties of Neural Networks Learning, *5th International Symposium on Automatic Control and Computer Science, SACCS'95*, Vol.2, 273-278, Technical University "Gh. Asachi" of Iasi, Romania.

[3] Enăchescu, C. (1996) Neural Networks as approximation methods. *International Conference on Approximation and Optimisation Methods, ICAOR'96*, " Babes-Bolyai University ", Cluj-Napoca.

[4] Enăchescu, C., (1995) *Learning the Neural Networks from the Approximation Theory Perspective. Intelligent Computer Communication ICC'95 Proceedings*, 184-187, Technical University of Cluj-Napoca, Romania.

[5] Funahashi, K. (1989). On the approximate realisation of continuos mappings by neural networks. *Neural Networks*, **2**, 183-192.

[6] Girosi, F., T. Pogio (1990). Networks and the Best Approximation Property. *Biological Cybernetics*, **63**, 169-176.

[7] Haykin, S. (1994), Neural Networks. A Comprehensive Foundation. *IEEE Press*, MacMillian.

[8] Hornik, K., Stinchcombe, M., White, H. (1989). Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*, **2**, 359-366.

[9] Irie, B., Miyake, S. (1988). Capabilities of Three-Layers Perceptrons. *IEEE International Conference on Neural Networks*, **1**, 641-648.

[10] Kreinovich, V.Y. (1991). Arbitrary Nonlinearity Is Sufficient to Represent All Functions by Neural Networks: A Theorem. *Neural Networks*, **4**, 381-383.

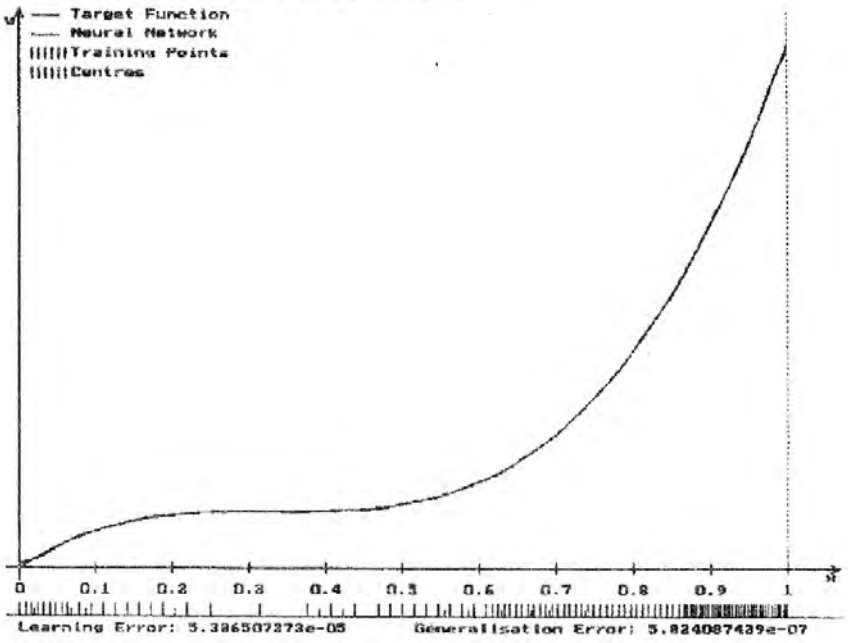


FIGURE 3. Approximation of the target function (24) by an RBF neural network using an active supervised learning algorithm: $N = 100$, 10000 epochs, 25 centres.

[11] Niyogi, P.(1995), Active Learning by Sequential Optimal Recovery. *A.I. Memo No. 1514*, C.B.C.L. Paper No. **113**, M.I.T, Massachusetts.

[12] Petri, A. J. (1991). A Nonlinear Network Model for continuos learning. *Neurocomputing* **3**, 157-176.

[13] Pogio, T., F. Girosi (1990). Networks for Approximation and Learning. *Proceedings of the IEEE*, Vol. **78**, **9**, 1481-1497.

[14] Valiant, L.G (1984), A theory of learnable. *Communication of ACM* **27** (11), 1134-1142.

"PETRU MAIOR" UNIVERSITY OF TÂRGU-MUREŞ, ROMANIA
E-mail address: ecalin@uttgm.ro

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

A FORMAL SUPPORT SYSTEM FOR THE OBJECT ORIENTED ANALYSIS OF AN APPLICATION DOMAIN

S. BERAR, M. VANCEA, AND A. VANCEA

Abstract. The paper introduces a formal system aimed to assist the programmer in the object oriented analysis of an arbitrary application domain. The object oriented models shift the modelling semantics towards a conceptual modelling in the application's domain. Being given a graphical representation of the application's domain structure, we further generate a correct and consistent specification in an intermediary code, independent of the programming language in which the application will be developed.

1. Preliminaries

In approaching complex applications domains, classical data models show some limitations, comming mainly from the impossibility of modelling implicit knowledge, the limits imposed by the capacity of abstraction and the difficulties that arise when trying to access some composite parts of the involved objects. Object oriented models (OOM) shift the modelling semantics towards a conceptual modelling in the application's domain [2] ignoring the intensive use of notions like pointers or trees. OOM reduces the gap between reality and the model.

The paper presents the design of a system (partially implemented) which assists the programmer in object oriented analysis of an arbitrary application domain.

It is generally simpler to specify graphically the structure of an application domain. Starting from such a representation, one tries to generate a correct and consistent specification, in an intermediary code, independently from the programming language in which the application will be programmed. Such a specification can be further processed for obtaining an object oriented source code.

The system has to fulfil the following functions:

a): Offering a friendly user interface for the graphical specification of the involved classes and the relationships between them; based on the graphical

1991 *Mathematics Subject Classification.* 68M15, 68Q60, 68N05.

1991 *CR Categories and Descriptors.* D.1.5 [Programming Techniques]: Object-oriented Programming; D.2.1 [Software Engineering]: Requirements and Specification – languages, methodologies; D.2.4 [Software Engineering]: Program Verification – reliability, validation.

specification, the algorithm presented in section 3 will generate an adequate model for the chosen application;

b): Verifying the corectness and the consistence of the user defined model, signaling the possible occuring errors;

c): Offering the possibility of the dynamic modification of the model and of completion of an existing one;

d): Generating a specification file based on the graphical representation developed by the user.

2. The formal description of the problem

We will further consider the *object* as introduced by Booch [5]: an object is an entity which has a *state*, *behaviour* and *identity*, the structure and behaviour of similar objects being defined in their common class.

An *object oriented model* is a tuple $(C, IS-A, O)$, where C is a type system, $IS-A$ represents the structural and behavioural inheritance relationship (which introduces a partial ordering relation in the type system) and O is a type that generalizes all others types, being the root of the type hierarchy [4].

The set C can be divided in three disjoint sets:

- C_A - the type set identified in the application domain;
- C_{AUX} - the auxiliary type set from the solution domain;
- C_B - the basic type set.

Considering that the C_A types are completed with the implementation component, we can approach the classes identified in the application domain. Our system intends to support the user in the specification process of these classes.

We will assume that the inheritance relation $IS-A$ is represented as the tuple set

$$SI = \{< C_i, C_j > \in C \times C | (C_i IS\text{-}A\ C_j)\}.$$

We consider also that a class C_i is internally represented in our system by a data structure of the form

```
class_name: string;
structure : set_of_attributes;
interface : set_of_methods;
```

We will denote by an *attribute* a tuple of the form:

$$< attr\_name{:}string,\ attr\_domain : C_A \cup C_B > .$$

A *method* is declared by a tuple of the form:

$$< method\_name{:}string,\ method\_domain{:}tuple\_of(C_B \cup C_A),\ method\_result : C_A \cup C_B > .$$

We have further to define the following elements:

- a basic set G of parameterized graphical symbols with which the description will be made;

- a couple of functions $(T_C : G \rightarrow C_A, T_I : G \rightarrow SI)$ which express the correspondence between the graphical symbols instantiated by the user and classes, and the inheritance relationship between classes, respectively;
- a set of restrictions $SC_I : SI- > BOOL$, which expresses the validity of the defined inheritance relationships;
- a set of restrictions $SC_C : C_A- > BOOL$, which expresses the structural and behavioural consistence and correctness requirements of the classes.

3. The basic algorithm of our system

The functionning of the support system is defined by the following sequence:

Step 1: Initialization of the C_A and SI sets with the empty set (for the cases in which we start a new model description) or with C_{A0} and SI_0 (when we start from a partially existent model);

Step 2: Starting from C_A and SI we build the set of graphical instantiated symbols, D_0, which describes the initial state of the system. We initialize the work set $D = D_0$.

Step 3: The user will instantiate a sequence of graphic symbols $g_i \in G$ by giving values to their parameters. It is also possible the elimination of some elements $ge_j \in D$. The set $D = D \cup \{g_i|\forall i\} \backslash \{ge_j|\forall j\}$ is updated accordingly.

Step 4: T_C and T_I are applied on D, resulting the new forms of C_A and SI.

Step 5: The validity of the generated model is evaluated with the formula $V = CV \cap IV$, where

$$CV = \cap CC_i(c_j), \text{ for all } CC_i \in SC_C, c_j \in C_A,$$

$$IV = \cap CI_i(s_j), \text{ for all } CI_i \in SC_I, s_j \in SI.$$

Step 6: If V evaluates to false, the algorithm goes on from step 3. Otherwise, a procedure for building and printing the specification is started, based on C_A and SI. The algorithm of this operation is simple and depends on the system's internal implementation.

The algorithm can be restarted at the user's request until obtaining a satisfactory specification, adequate to the considered application domain. The output of the algorithm is a text file which contains the description (following the BNF syntax presented below in section 4) of the classes generated at step 6.

Our implementation was programmed in C++ under Windows, for having graphic facilities regarding the development of the graphical procedures.

4. The syntax of the specification language

The syntax of the generated specification language given in BNF form is adapted from [1] and follows below:

<class>::=**class** <name>
superclass <list_domains>
structure <list_attributes>
interface <list_methods¿
end <name>.
<list_domains> ::= <domain> | <list_domains>;<domain>
<list_attributes> ::= <attribute>;<list_attributes> | ε
<list_methods> ::= <method>; <list_methods> | ε
<domain> ::= <name> | <base_type>
<base_type> ::= **int** | **real** | **string** | **char** | **boolean**
<attribute> ::= <name>:<domain>
<method> ::= <input> $\rightarrow$ <output>
<input> ::= ε | <domain> | <domain>x<input>
<output> ::= <domain>
<name> ::= **identifier**

Steps 5 and 6 of the algorithm presented in section 3 assures the validation and the correct generation (meaning to follow the restrictions imposed at step 5) of the specification language described above. We have thus that the generated specification will follow the defined restrictions set.

5. The set of graphical symbols

The structure of a graphical symbol is formed by a tuple of parameters $(p_1, p_2, \ldots p_n)$, n varying upon the type of that symbol.

We will use the following graphical symbols: **class (a)**, **inherit (b)**, **aggregate one (c)**, **aggregate many (d)**, based on the OMT notation [6]. These assure the expressiveness of the classes specification and of the orthogonal hierarchies *IS- A* and PART-OF. For specifying the methods of a class one will use also a graphical symbol, namely **method (e)**.

A common parameter to all those symbols is *shape*, which refers to the graphical aspect presented to the user. In figure 1 it is presented the value of this parameter for every symbol used.

6. Model validation

Model validation means verifying the restrictions set $SC_C : C_A -> BOOL$, which expresses the correctness and structural and behavioural consistency requests of the classes, and respectively of the restrictions set $SC_I : SI -> BOOL$, which corresponds to the inheritance relationships [2].

The *classes validation* is accomplished at the moment at which all the classes were described. This validation is composed of:

- verifying if the classes names (class_name) are unique and if the attributes and methods names are unique in the frame of the same class;

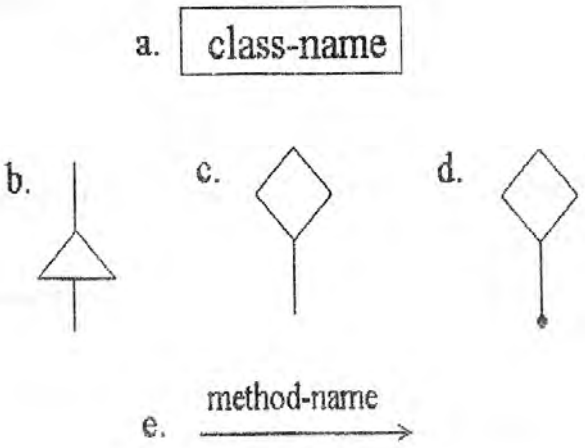


FIGURE 1. The values of the *shape* parameter for the graphical symbols from G

- verifying the existence of the attributes definition domains and also of the domains on which the methods are defined; these could be implicit domains or user defined classes.

The *relationships validation* refers particulary to multiple inheritance restrictions verification. Multiple inheritance is allowed when there are no two distinct descendant paths [4]. But, considering that in the model of an arbitrary application domain all the classes emerge from an abstract class called ROOT, it follows that any multiple inheritance relationship leads us to multiple paths. From this reason, in the program, multiple inheritance will be signaled as a validation error.

7. Conclusions

In the paper a theoretical model for designing, and afterwards for syntactic and semantic validation of the object oriented applications is introduced and developed. These actions take place in the context proposed in section 4.

The main advantage of using this specification and validation method resides in simplifying the design and generation of the object oriented applications. Our implemented system has mainly a didactic goal and helps the user in the correct design and reliability of his object oriented applications.

References

[1] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley Series in Computer Science, 1986.

[2] B. Eckel, *Using C++*, Osborne/McGraw-Hill, 1990.

[3] * * *, *BORLAND C++: Programmer's Guide*, 1990.

[4] I. Salomie, *Object Oriented Programming Techniques*, Ed. Microinformatica, Cluj-Napoca, 1995 (in Romanian).

[5] G. Booch, *Object Oriented Design with Applications*, The Benjamin/Cummings Publishing Company Inc., 1991.

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modelling and Design*, Prentice Hall International, 1991.

BABEŞ-BOLYAI UNIVERSITY, FACULTY OF ECONOMICS, RO 3400 CLUJ-NAPOCA, STR. KOGĂLNICEANU 1, ROMANIA

E-mail address: {sanda,vancea}@econ.ubbcluj.ro

BABEŞ-BOLYAI UNIVERSITY, FACULTY OF MATHEMATICS AND INFORMATICS, RO 3400 CLUJ-NAPOCA, STR. KOGĂLNICEANU 1, ROMANIA

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

STUDIA UNIV. "BABEŞ-BOLYAI", INFORMATICA, Volume I, Number 2, October 1996

THE DENSITY – A NUMERICAL CHARACTERISTIC FOR LANGUAGES

A. F. BOER

Abstract. In this paper we present the density of languages, that was defined by Stotzky in [3], we study some properties, and establish the density for some classes of languages. With this numerical characteristic it is possible to show that some languages can't be generated by grammars from some classes.

Definition 0.1 (3). *Let M an infinite subset of the set N of the natural numbers and $d : N \to N$ a function with $d(n) \leq d(m)$ if $n < m$. We say that the set M has the density d, if there exists a number $n_0 \in N$ so that for each natural number $n \geq n_0$ there exists an element m in M with the property $n \leq m \leq n + d(n)$.*

We remark that the density of a set isn't unique, but it is possible to find a "maximal" density.

Definition 0.2 (3). *The density of the language L is the density of the set of lengths of the words from L, i.e. the density of the set $M(L) = \{n \in N | \exists x \in L | x| = n\}$.*

We remark that we can reformulate this definition so: the infinite language L has the density d (where $d : N \to N$ is a function with $d(n) \leq d(m)$ if $n < m$), there exists a natural number n_0 such that for each $n \geq n_0$ there exists a word x in L such that $n \leq |x| \leq n + d(n)$.

Definition 0.3. *If $d = c$ (where c is a constant number) then the density is constant.*

Definition 0.4. *If the density is $d(n) = cn$ (where c is a constant number), then we say that it is a linear density.*

Remark 0.5. *In this case the definition may be reformulated in the following equivalent form: $\exists n_0 \leq 0 \exists c_1 > 1$ such that for each n from N if $n \geq n_0$ then there exists an element m in M such that $n \leq m \leq c_1 n$, where $c_1 = c + 1$.*

1991 *Mathematics Subject Classification.* Completeaza aici AMS classification.
1991 *CR Categories and Descriptors.* Completeaza aici CR categories and descriptors.

Proposition 0.6. *Let M be the set of the terms of an arithmetical progression, i.e. $M = \{a + bn | n \in N\}$ where a and $b > 0$ are natural constants. Then M has constant density.*

Proof. Let $n_0 = a$; for each n, between n and $n + b$ there exists (exactly) one term of the progression, so the constant function $d(n) = b$ is the density for the set M. □

Remark 0.7.
1. *If the set $M_1 \subseteq N$ has the density $d(n)$ and $M1 \subseteq M2 \subseteq N$ then the set $M2$ has the density $d(n)$ too.*
2. *From the proposition 0.6 and remark 0.7 it results that if a set contains an arithmetical progression then it has constant density.*
3. *If the set $M \subseteq N$ has the density $d(n)$ and $d' : N \to N$ is another function such that $d'(n) \geq d(n)$, for each n, then the set M has the density $d'(n)$ too.*

Proposition 0.8. *Let M the set of the terms of a geometrical progression, that means: $M = \{ab^n | n \in N\}$ where a and b $(b > 0, b \neq 1)$ are natural constants. Then the set M has linear density and has no constant density.*

Proof. Because of $b > 1$, the progression is increasing. Let $n_0 = a$; for each n, between n and bn there exists at least one term of the progression: if m is the first natural number for which $ab^m \geq n$ then we have $ab^m < ab^{m+1} = bn$, also the linear function $d(n) = bn$ is density for the set M.

Now we show that M has no constant density. Supposse that M has constant density, i.e. we have the function $d : N \to N, d(n) = c$ for each natural n and there exists n_0 such that for each $n \geq n_0$ there exists an element m in M for which $n \leq m \leq c + n$. It means that there exists a natural number p so that $n \leq ab^p \leq c+n$. i. e. $\frac{n}{a} \leq b^p \leq \frac{n+c}{a}$, and from this $\log_b\left(\frac{n}{a}\right) \leq p \leq \log_b\left(\frac{n+c}{a}\right)$. But this is impossible because for a sufficient large p the difference between $\log_b\left(\frac{n}{a}\right)$ and $\log_b\left(\frac{n+c}{a}\right)$ is less than any real number

$$\log_b\left(\frac{n+c}{a}\right) - \log_b\left(\frac{n}{a}\right) = \log_b\left(\frac{n+c}{n}\right) < \varepsilon$$

for $n > \frac{c}{b^\varepsilon - 1}$, and so there is no natural number between n and $n + c$, and the density can not be constant. □

Proposition 0.9. *Each infinite regular set has constant density.*

Proof. From the theorem 2.8. in [2] results that if $L \subseteq T*$ is a regular set, then there exists the natural numbers (constants) p and q so that for each word z in L with $|z| > p$ there exists the words x, u, y in $T*$ such that $z = xuy$, $u1e(i.e.|u| > 0)$, $|uy| \leq q$, for each $k \geq 1$ the word $xu^k yis$ in the language L. We have: $|xu^k y| = |x| + k|u| + |y| = a + kb$, where $a = |x| + |y|$ and $b = |u| > 0$, also $M(L)$ contains an arithmetical progression and then according to the proposition 0.6, the set $M(L)$ has the constant density $d(n) = b = |u|$. □

Proposition 0.10. *Each context-free language has constant density.*

Proof. We can proceed as in the proof of the proposition 0.9. Using the Bar-Hillel lemma [2, 3], where we will have for each word of the form xu^kwv_ky the relations: $|xu^kwv_ky| = |x| + k|u| + |w| + k|v| + |y| = a + kb$, where $a = |x| + |w| + |y| and b = |u| + |v|$, and so $M(L)$ contains an arithmetical progression and then according to the proposition 0.6 the set $M(L)$ (and the language L) has the constant density $d(n) = b = |u| + |v|$. □

Remark 0.11.
1. *There are languages with constant density which aren't context-free languages. For example the language* $L = \{a^nb^nc^n | n \geq 1\}$ *in the alphabet* $\{a, b, c\}$ *isn't context-free language [1], but has constant density.*
2. *The language* $L = \{a^{k^n} | n \geq 1, k \in N\backslash\{0, 1\}, k$ *is a constant*$\}$ *has linear density. In fact, we can use the proposition 0.8 when we observe that the lengths of the words from L form a geometrical progression.*

Example 0.12. *The set* $M = \{2^{2^k} | k > 0\}$ *has higher density than linear.*

We suppose that the density is linear, also that there exists the constant n_0 and the constant such that for each $n \geq n_0$ there exists an m in M for which $n \leq m \leq cn$. That means that for each $n \geq n_0$ it is a natural number k for which we have $n \leq 2^{2^k} \leq cn$. From this follows $\log_2 n \leq 2^k \leq \log_2 c + log_2 n$, also for each k we have $\log_2(\log_2 n) \leq k \leq \log_2(\log_2 n + \log_2 c)$ (beginning from a value). But this is impossible because for n sufficient large the difference between the two margins - which generally aren't natural numbers - will be less than any real number ε; let $\varepsilon < 1$. Moreover we have $\log_2(\log_2 n + \log_2 c) - \log_2(\log_2 n) < \varepsilon$ if and only if $\log_2 \frac{\log_2 n + \log_2 c}{\log_2 n} < \varepsilon$, also if $\log_2 \left(1 + \frac{\log_2 c}{\log_2 n}\right) < \varepsilon$. That is, if we have $\frac{\log_2 c}{\log_2 n} < \varepsilon$, which holds for $n < c^{1/\varepsilon}$. Also if $\log_2(\log_2 n)$ is not a natural number and $n > c^{1/\varepsilon}$ for a given ε, then we have no element from M between n and $n + cn$, and so the density cannot be linear.

Remark 0.13. *The density of the union of two set is less or equal to the density of each set. This is obviously from the remark 0.7.*

Remark 0.14. *The density of the intersection of two sets can be greater as the density of this sets.*

For example, let $M_1 = \{2^n | n = 1, 2,\}$ and $M_2 = \{n2^n | n = 1, 2,\}$. These sets have linear density, but the intersection $M_1 \cap M_2 = \{2^n 2^{2^n} | n = 1, 2, ...\}$ has no linear density. Another example is the intersection of two sets with constant density, which has no constant, but linear density: $M_3 = \{2n | n \in N\}$,

$$M_4 = \{q_n | q_n = \begin{cases} 2n, & \text{if } \exists k \in N, n = 2^k, \\ 2n + 1, & \text{otherwise} \end{cases} , n \in N\}$$

Obviously $M_3 \cap M_4 = \{2^n | n \in N^*\}$, and this set has linear, but not constant density.

Conclusion. The density characterises a set of natural numbers and so a language too. According to the remark 17, the intersection of two languages can have a greater density as the two original languages, and so it is possible e.g. to prove that some classes of languages are not closed under intersection.

References

[1] S. Ginsburg, *The Mathematical Theory of Context-Free Languages*, Mc Graw Hill Book Company, New York, 1966.

[2] Gh. Paun, *Probleme actuale n teoria limbajelor formale*, Editura Stiintifica si Enciclopedica, Bucuresti, 1984.

[3] E.D. Stotzky, *Uslovnie grammatiki s rasseiannim kontekstom*, Dokl. AN SSSR, 207(1972), No. 4.