Anul XLIV 1999

S T U D I A

UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

2

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

SUMAR – CONTENTS – SOMMAIRE

T. Jucan, C. Vidrascu, Concurency-Degrees for Petri Nets.....3

S. Motogna, F-Bounded Quantification and the Matching Relation.....16

P.P. Blaga, A General Class of Nonproduct Quadrature Formulas……23

T. Petrila, Complex Value Boundary Elements Methods (CVBEM) for Some Mixed BVP……..……..37

C. Popescu, Blind Signature and Blind Multisignature Schemes Using Elliptic Curves……..43

D. Radoiu, A. Roman, A Component Based Approach for Scientific Visualization of Experimental Data……...50

D.M. Suciu, Extending Statecharts for Concurrent Objects Modeling……65

F.M. Boian, C. Ferdean, Half Synchronized Transition Systems…….77

D. Dumitrescu, M. Oltean, An Evolutionary Algorithm for Theorem Proving in Propositional Logic……..87

A.Onet, D. Tatar, Semantic Representation of the Quantitative Natural Language Sentences……..99

ANIVERSARI – ANNIVERSARIES – ANNIVERSAIRES

E. Munteanu, M. Frentiu, Professor Grigor Moldovan at his 60th Anniversary….110

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

CONCURRENCY-DEGREES FOR PETRI NETS

TOADER JUCAN AND CRISTIAN VIDRAŞCU

Abstract. The goal of this paper is to extend some concepts (about concurrency-degrees) from the class of Place/Transition Petri nets (PTN) to the class of jumping Petri nets ($JPTN$). Also, we will present a simpler definition of concurrency-degree for PTN. Moreover, we will point out how we can compute these concurrency-degrees.

Keywords: distributed systems, concurrency, Petri nets, jumping Petri nets, concurrency-degrees.

1. Introduction

A Petri net is a mathematical model used for the specification and the analysis of parallel/distributed systems. It is very important to introduce a measure of concurrency for parallel/distributed systems. What is the meaning of the fact that in the system S_1 the concurrency is greater than in the system S_2 ? We will study the problem of concurrency for Petri nets, but, since the Petri nets are used as suitable models for real parallel/distributed systems, the results will be applicable also to these systems.

It is well-known that the behaviour of some distributed systems cannot be adequately modelled by classical Petri nets. Many extensions which increase the computational and expressive power of Petri nets have been thus introduced. One direction has led to various modifications of the firing rule of nets. One of these extension is that of jumping Petri net, introduced in [TiJ94].

The notion of concurrency-degree for Petri nets was first introduced in [TJD93]. In this paper we will give a simpler definition of concurrency-degree for Petri nets, and we will extend this notion for jumping Petri nets. Also, we will show how we can compute these concurrency-degrees.

The paper is organized as follows. Section 2 presents the basic terminology, notation and results concerning Petri nets and jumping Petri nets. In section 3, and respectively 4, we present the definition of concurrency-degree for Petri nets, respectively for jumping Petri nets, and we show how we can compute these

1991 *CR Categories and Descriptors.* D.2.2 [**Software Engineering**]: Tools and Techniques – *Petri nets*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation – *Parallelism and concurrency* .

concurrency-degrees. Finally, in section 5 we conclude this paper and formulate some open problems.

2. Preliminaries

In this section we will establish the basic terminology, notation, and results concerning Petri nets in order to give the reader the necessary prerequisites for the understanding of this paper (for details the reader is referred to [BeF86], [JuT99], [Rei85], [Rei87]). Mainly, we will follow [JuT99], [TiJ94], [TiM97].

2.1. **Petri nets.** A *Place/Transition net*, shortly *P/T-net* or *net*, (finite, with infinite capacities), abbreviated *PTN*, is a 4-tuple $\Sigma = (S, T; F, W)$, where S and T are two finite non-empty sets (of *places* and *transitions*, resp.), $S \cap T = \emptyset$, $F \subseteq (S \times T) \cup (T \times S)$ is the *flow relation* and $W : (S \times T) \cup (T \times S) \to \mathbf{N}$ is the *weight function* of Σ verifying $W(x, y) = 0$ iff $(x, y) \notin F$.

A *marking* of a *PTN* Σ is a function $M : S \to \mathbf{N}$; it will be sometimes identified with a vector $M \in \mathbf{N}^{|S|}$. The operations and relations on vectors are componentwise defined. $\mathbf{N}^S$ denotes the set of all markings of Σ.

A *marked PTN*, abbreviated *mPTN*, is a pair $\gamma = (\Sigma, M_0)$, where Σ is a *PTN* and M_0, called the *initial marking* of γ, is a marking of Σ.

In the sequel we often use the term "Petri net" (*PN*) or "net" whenever we refer to a *PTN* (*mPTN*) γ and it is not necessary to specify its type (i.e. marked or unmarked).

Let γ be a net, $t \in T$ and $w \in T^*$. The functions $t^-, t^+ : S \to \mathbf{N}$ şi $\Delta t, \Delta w : S \to \mathbf{Z}$ are defined by $t^-(s) = W(s, t)$, $t^+(s) = W(t, s)$, $\Delta t(s) = t^+(s) - t^-(s)$ and

$$\Delta w(s) = \begin{cases} 0, & \text{if } w = \lambda, \\ \sum_{i=1}^{n} \Delta t_i(s), & \text{if } w = t_1 t_2 \dots t_n (n \geq 1), \end{cases} \quad \text{for all } s \in S.$$

The sequential behaviour of a net γ is given by so-called *firing rule*, which consist of

- the *enabling rule*: a transition t is *enabled* at a marking M in γ (or t is *fireable* from M), abbreviated $M[t\rangle_\gamma$, iff $t^- \leq M$;
- the *computing rule*: if $M[t\rangle_\gamma$, then t may *occur* yielding a new marking M', abbreviated
  $M[t\rangle_\gamma M'$, defined by $M' = M + \Delta t$.

The notation "$[.\rangle_\gamma$" will be simplified to "$[.\rangle$" whenever γ is understood from context.

In fact, for any transition t of γ we have a binary relation on $\mathbf{N}^S$, denoted by $[t\rangle_\gamma$ and given by: $M[t\rangle_\gamma M'$ iff $t^- \leq M$ and $M' = M + \Delta t$. If $t_1, t_2, \dots, t_n$, $n \geq 1$, are transitions of γ, $[t_1 t_2 \dots t_n\rangle_\gamma$ will denote the classical product of the relations $[t_1\rangle_\gamma, \dots, [t_n\rangle_\gamma$, i.e. $[t_1 t_2 \dots t_n\rangle_\gamma = [t_1\rangle_\gamma \circ \dots \circ [t_n\rangle_\gamma$. Moreover, we consider the relation $[\lambda\rangle_\gamma$ given by $[\lambda\rangle_\gamma = \{(M, M) | M \in \mathbf{N}^S\}$.

Let γ be a marked Petri net, $M \in \mathbf{N}^S$ and M_0 its initial marking. The word $w \in T^*$ is called a *transition sequence* from M in γ if there exists a marking M' of γ such that $M[w\rangle_\gamma M'$. Moreover, the marking M' is called *reachable* from M in γ. We denote by $TS(\gamma, M) = \{w \in T^* | M[w\rangle_\gamma\}$ the set of all transition sequence from M in γ, and by $RS(\gamma, M) = [M\rangle_\gamma = \{M' \in \mathbf{N}^S | \exists w \in TS(\gamma, M) : M[w\rangle_\gamma M'\}$ the set of all reachable markings from M in γ.

In the case $M = M_0$, the set $TS(\gamma, M_0)$ is abbreviated by $TS(\gamma)$, and the set $RS(\gamma, M_0)$ is abbreviated by $RS(\gamma)$ (or $[M_0\rangle_\gamma$) and it is called *the set of all reachable markings* of γ.

The marking M is *coverable* in γ if there exists a marking $M' \in [M_0\rangle_\gamma$ such that $M \leq M'$.

Let γ be a P/T-net, and $T' \subseteq T$ a set of transitions, which is called *step*. The step-type concurrent behaviour of the net γ is given by so-called *step firing rule*, which consist of

- the *step enabling rule*: a step T' is *concurrently enabled* at a marking M in γ (or T' is *fireable* from M), abbreviated $M[T'\rangle_\gamma$, iff $\sum_{t \in T'} t^- \leq M$;
- the *step computing rule*: if $M[T'\rangle_\gamma$, then T' may *occur* yielding a new marking M', abbreviated $M[T'\rangle_\gamma M'$, defined by $M' = M + \sum_{t \in T'} \Delta t$.

2.2. **Jumping Petri nets.** Jumping Petri nets ([TiJ94], [TiM97]) are an extension of classical nets, which allows them to do "spontaneous jumps" from a marking to another one (this is similar to λ-moves in automata theory).

A *jumping P/T-net*, abbreviated *JPTN*, is a pair $\gamma = (\Sigma, R)$, where Σ is a *PTN* and R, called the *set of (spontaneous) jumps* of γ, is a binary relation on the set of markings of Σ (i.e. $R \subseteq \mathbf{N}^S \times \mathbf{N}^S$). In what follows the set R of jumps of any *JPTN* will be assumed *recursive*, that is for any couple of markings (M, M') we can effectively decide whether or not $(M, M') \in R$.

A *marked jumping net*, abbreviated *mJPTN*, is defined similarly as an *mPTN*, by changing "Σ" into "Σ, R".

Let $\gamma = (\Sigma, R)$ be a *JPTN*. The pairs $(M, M') \in R$ are referred to as *jumps* of γ. If γ has finitely many jumps (i.e. R is finite) then we say that γ is a *finite jumping net*, abbreviated *FJPTN*.

We shall use the term "*jumping net*" (*JN*) ("*finite jumping net*" (*FJN*), resp.) to denoted a *JPTN* or a *mJPTN* (a *FJPTN* or a *mFJPTN*, resp.) whenever it is not necessary to specify its type (i.e. marked or unmarked).

Pictorially, a jumping Petri net will be represented as a classical net and, moreover, the relation R will be separately listed.

The behaviour of a jumping net γ is given by the *j-firing rule*, which consist of

- the *j-enabling rule*: a transition t is *j-enabled* at a marking M (in γ), abbreviated $M[t\rangle_{\gamma,j}$, iff there exists a marking M_1 such that $MR^*M_1[t\rangle_\Sigma$ (Σ being the underlying net of γ and R^* the reflexive and transitive closure of R);

- the *j-computing rule*: if $M[t\rangle_{\gamma,j}$, then the marking M' is *j-produced* by occurring t at M, abbreviated $M[t\rangle_{\gamma,j}M'$, iff there exists two markings M_1, M_2 such that $MR^*M_1[t\rangle_\Sigma M_2R^*M'$.

The notation "$[.\rangle_{\gamma,j}$" will be simplified to "$[.\rangle_j$" whenever γ is understood from the context.

The notions of *transition j-sequence* and *j-reachable marking* are defined similarly as for Petri nets (the relation $[\lambda\rangle_{\gamma,j}$ is defined by $[\lambda\rangle_{\gamma,j} = \{(M, M')|M, M' \in \mathbf{N}^S, MR^*M'\}$).

The *set of all j-reachable markings* of a marked jumping net γ is denoted by $RS(\gamma)$ or by $[M_0\rangle_{\gamma,j}$ (M_0 being the initial marking of γ).

The marking M is *coverable* in γ if there exists a marking $M' \in [M_0\rangle_{\gamma,j}$ such that $M \leq M'$.

Some jumps of a marked jumping net may be never used. Thus we say that a marked jumping net $\gamma = (\Sigma, R, M_0)$ is *R-reduced* ([TiJ94]) if for any jump $(M, M') \in R$ of γ we have $M \neq M'$ and $M \in [M_0\rangle_{\gamma,j}$.

3. Concurrency-degrees for P/T nets

The notion of concurrency-degree for Petri nets was first introduced in [TJD93] (that definition can be found also in [JuT99]). Here we will give a simpler definition of this notion.

First, let us recall the definition of a (maximal) step:

Definition 3.1. *Let $\gamma = (S, T; F, W)$ be a Petri net and M an arbitrary marking of γ.*
i) $T' \subseteq T$ is called a set of transitions concurrently enabled at M *(or, briefly, a* step at M*) if $\sum_{t\in T'} t^- \leq M$;*
ii) $T' \subseteq T$ is called a maximal set of transitions concurrently enabled at M *(or, briefly, a* maximal step at M*) if T' is a step at M and, for each $t \in T - T'$, $T' \cup \{t\}$ is not a step at M.*

Notation 3.1. *Let $\gamma = (S, T; F, W)$ be a Petri net and M an arbitrary marking of γ.*
1) We denote by $T(M)$ the set of all transitions enabled at the marking M, i.e.

$$T(M) = \{t \in T|\ t^- \leq M\}\ ;$$

2) We denote by $CT(M)$ the set of all subsets of transitions concurrently enabled at M, i.e.

$$CT(M) = \{T' \subseteq T|\ \sum_{t\in T'} t^- \leq M\}\ ;$$

3) We denote by $MCT(M)$ the set of all maximal subsets of transitions concurrently enabled at the marking M, i.e.

$$MCT(M) = \{T' \subseteq T|T' \text{ is a maximal step at } M\}\ .$$

Generally speaking, there exist more maximal subsets of transitions concurrently enabled at a marking M. Moreover, the maximality of sets w.r.t. concurrency does not imply the maximality of sets w.r.t. cardinality of sets (i.e. we can have two maximal steps at M, T_1 and T_2, with $|T_1| < |T_2|$).

Example 3.1. *For the marked Petri net γ represented in figure 1, it is easy to see that the subsets $T' = \{t_1, t_2, t_3, t_4\}$ and $T'' = \{t_1, t_2, t_5\}$ are maximal steps at the initial marking of γ, and, moreover, these are the only ones, i.e. $MCT(M_0) = \{T', T''\}$.*

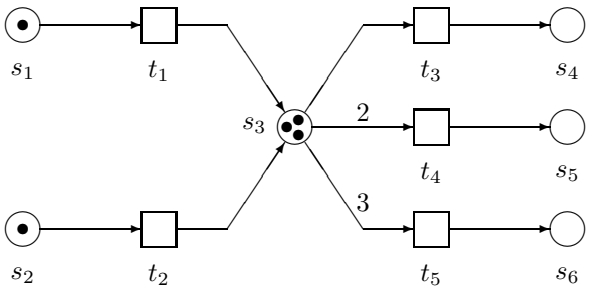


FIGURE 1. the net from example 3.1

Definition 3.2. *Let γ be a Petri net and M an arbitrary marking of γ. The* concurrency-degree at the marking M *of the net γ is defined by:*

$$d(\gamma, M) = \max\{\ |T'| \mid T' \in MCT(M)\}\ .$$

Definition 3.3. *Let $\gamma = (\Sigma, M_0)$ be a marked P/T-net.*
i) The inferior concurrency-degree *of the net γ is defined by:*

$$d^-(\gamma) = \min\{\ d(\gamma, M) \mid M \in [M_0\rangle_\gamma\}\ ;$$

ii) The superior concurrency-degree *of the net γ is defined by:*

$$d^+(\gamma) = \max\{\ d(\gamma, M) \mid M \in [M_0\rangle_\gamma\}\ .$$

Remark 3.1. *Directly from definitions we have*
1) $0 \leq d^-(\gamma) \leq d^+(\gamma) \leq |T|$;
2) The inferior concurrency-degree of the net γ, $d^-(\gamma)$, represents the minimum number of transitions concurrently enabled at any reachable marking of γ, and has the property that there exists at least one reachable marking $M \in [M_0\rangle_\gamma$ such that there exists $d^-(\gamma)$ transitions concurrently enabled at M;
3) The superior concurrency-degree of the net γ, $d^+(\gamma)$, represents the maximum number of transitions concurrently enabled at any reachable marking of γ, and has the property that there exists at least one reachable marking $M \in [M_0\rangle_\gamma$ such that there exists $d^+(\gamma)$ transitions concurrently enabled at M.

Definition 3.4. *Let $\gamma = (\Sigma, M_0)$ be a marked Petri net. If $d^-(\gamma) = d^+(\gamma)$, then we denote this number with $d(\gamma)$, i.e. $d(\gamma) = d^-(\gamma) = d^+(\gamma)$, and we called it the* concurrency-degree *of γ.*

Example 3.2. *For the marked Petri net γ represented in figure 2, it is easy to see that transition t_1 is fireable from any reachable marking, i.e. $t_1 \in T(M)$, for all $M \in [M_0\rangle_\gamma$, which means that the inferior concurrency-degree is at least one: $d^-(\gamma) \geq 1$. Let M_1 be the marking produced by the occurence of t_2 at the initial marking, i.e. $M_0[t_2\rangle_\gamma M_1$; $M_0 = (2, 1, 0, 0, 0)$ and $M_1 = (1, 0, 0, 1, 0)$. Since t_1 is the only transition fireable at M_1, i.e. $T(M_1) = \{t_1\}$, we have that $d(\gamma, M_1) = 1$, and therefore the inferior concurrency-degree is $d^-(\gamma) = 1$. Moreover, it is easy to see that the transition t_2 can occur at most one time in any transition sequence starting from the initial marking, and that the transition t_3 can occur also at most one time, and only after the occurence of t_2. This means that the set $T = \{t_1, t_2, t_3\}$ cannot be a step at any reachable marking, thus we have $d^+(\gamma) < 3$. Since the subset $T' = \{t_1, t_2\}$ is the only maximal step at M_0, i.e. $MCT(M_0) = \{T'\}$, we have that $d(\gamma, M_0) = 2$. Thus, the superior concurrency-degree is $d^+(\gamma) = 2$.*

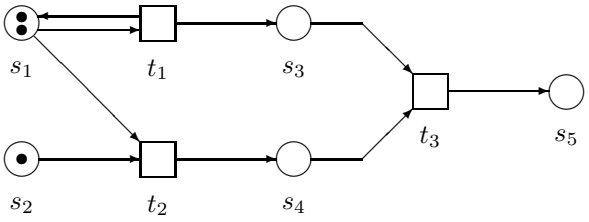


FIGURE 2. the net from example 3.2

In the sequel, we will show how we can compute the concurrency-degrees of a Petri net.

First of all, we will present the algorithm for computing the concurrency-degree at any marking of a Petri net.

Let γ be a Petri net, let M be an arbitrary marking, and let $|T| = n$. Obviously, $d(\gamma, M) \leq n$. The algorithm is the following:

Theorem 3.1. *The concurrency-degree at a marking, $d(\gamma, M)$, is computable for any PTN γ and for any marking M.*

Proof. It is easy to prove that the above algorithm is finite (i.e. it always stops) and it computes exactly the concurrency-degree at the marking M of γ. □

The complexity of the algorithm is $\mathcal{O}(2^{|T|} \cdot |S|)$.

procedure **concurrency_degree_at_a_marking** (γ: PTN, M: marking);
begin
for $i := n$ downto 0 do // where $n = |T|$
begin
// consider all the subsets of T with i elements
for each $T' \subseteq T$ such that $|T'| = i$ do
begin
$M' := \sum_{t \in T'} t^-$; // M' is the smallest marking at which T' is concurrently enabled
if $M' \leq M$ then goto STOP;
end;
end;
STOP: $d(\gamma, M) := i$;
return $d(\gamma, M)$;
end.

Now, we will present the algorithm for computing the superior concurrency-degree of a marked Petri net. Let $\gamma = (\Sigma, M_0)$ be a $mPTN$, and let $|T| = n$. Obviously, $d^+(\gamma) \leq n$. The algorithm is the following:

procedure **superior_concurrency_degree** (γ: $mPTN$);
begin
for $i := n$ downto 0 do // where $n = |T|$
begin
// consider all the subsets of T with i elements
for each $T' \subseteq T$ such that $|T'| = i$ do
begin
$M' := \sum_{t \in T'} t^-$; // M' is the smallest marking at which T' is concurrently enabled
if is_coverable(γ,M') then goto STOP;
end;
end;
STOP: $d^+(\gamma) := i$;
return $d^+(\gamma)$;
end.

boolean function is_coverable (γ: $mPTN$, M: marking);
begin
Let $\mathcal{MCG}(\gamma)$ be the minimal coverability graph of γ;
if (there exists at least one node M' in $\mathcal{MCG}(\gamma)$ such that $M \leq M'$)
then return true else return false;
end.

Theorem 3.2. *The superior concurrency-degree $d^+(\gamma)$ is computable for any mPTN γ.*

Proof. Since the coverability problem is decidable for *mPTN* ([KaM69]) (the function is_coverable solves this problem by using the minimal coverability graph, [Fin93]), it is easy to prove that the above algorithm is finite (i.e. it always stops) and it computes exactly the superior concurrency-degree of the net γ. □

The complexity of the algorithm is $\mathcal{O}(2^{|T|}) \cdot \mathcal{O}(CP)$, where $\mathcal{O}(CP)$ is the complexity of the coverability problem solved using the minimal coverability graph (for details, see [Fin93]).

Now, we will show how we can compute the inferior concurrency-degree of a marked Petri net.

Let $\gamma = (\Sigma, M_0)$ be a marked Petri net. First, let us remark that if the reachability set $[M_0\rangle_\gamma$ is finite (this problem is decidable, [KaM69]), then we can compute the inferior concurrency-degree of γ by using directly the definition: $d^-(\gamma) = \min\{\ d(\gamma, M) \mid M \in [M_0\rangle_\gamma\}$, because the minimum is computed on a finite set and $d(\gamma, M)$ is computable, for each marking M (theorem 3.1).

Now, let us consider that the reachability set $[M_0\rangle_\gamma$ is infinite. Then, there exists a finite subset $\mathcal{M} \subseteq [M_0\rangle_\gamma$ such that

$$(*) \quad \forall M \in [M_0\rangle_\gamma, \exists M' \in \mathcal{M} \text{ such that } M' \leq M.$$

Indeed, we can consider $\mathcal{M}$ as being the set of minimal reachable markings of γ, i.e.

$$\mathcal{M} = \{M \in [M_0\rangle_\gamma \mid \forall M' \in [M_0\rangle_\gamma - \{M\} \ : \ M' \not\leq M\}.$$

Then, we have the result:

Proposition 3.1. *The following equality holds:*

$$\min\{d(\gamma, M) | M \in [M_0\rangle_\gamma\} = \min\{d(\gamma, M) | M \in \mathcal{M}\}.$$

Proof. This equality follows easily from $(*)$ and from the fact that the concurrency-degree at a marking is a monotone increasing function, i.e. $M_1 \leq M_2 \Rightarrow d(\gamma, M_1) \leq d(\gamma, M_2)$. □

The following result about the usual quasi-ordering (i.e. the quasi-ordering on components) on $\mathbf{N}^k$ is well-known:

Lemma 3.1. (Dickson's lemma, [Dic13])
The usual quasi-ordering on $\boldsymbol{N}^k$ is a well quasi-ordering (i.e. from every infinite sequence of elements from $\boldsymbol{N}^k$, we can extract an infinite increasing sequence).

Proceeding from Dickson's lemma, it follows that any subset of $\mathbf{N}^k$ contains only finitely many incomparable vectors. Since, by its definition, the elements of $\mathcal{M}$ are incomparable, it follows that $\mathcal{M}$ is a finite set. Thus, since the set $\mathcal{M}$ of

minimal reachable markings of the P/T-net γ is computable, from proposition 3.1 it follows that:

Theorem 3.3. *The inferior concurrency-degree $d^-(\gamma)$ is computable for any mPTN γ.*

4. Concurrency-degrees for jumping Petri nets

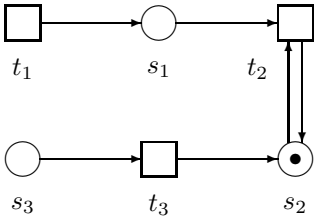
A jumping Petri net is a classical net Σ equipped with a (recursive) binary relation R on the markings of Σ. The meaning of a pair $(M, M') \in R$ is that the net Σ may "spontaneously jump" from M to M' (this is similar to λ-moves in automata theory).

We presented the definitions regarding jumping Petri nets in section 2. Now, we will present first an example of a jumping Petri net.

Example 4.1. *Let us consider a system consisting of a producer and a consumer, and a buffer with unlimited capacity, used for storing the products produced by the producer and consumed by the consumer. Moreover, we assume that the producer may take a break in any moment, and the consumer may take a break only when the buffer is empty (i.e., only when there are no products to consume).*

Such a system cannot be modelled by a classical Petri net ([JuT99]*). A modelling by an inhibitor Petri net was presented in* [JuT99]*. Here we will present a modelling of this system by a jumping Petri net.*

Let $\gamma = (\Sigma, R, M_0)$ be the marked jumping Petri net represented in figure 3. The place s_1 models the unlimited buffer, the transition t_1 models the producing of a product by the producer, and the transition t_2 models the consuming of a product by the consumer. The place s_2 models the active state of the consumer, and the place s_3 models the inactive state of him (i.e., the consumer is in a break). The fact that the consumer may take a break only when the buffer is empty, is modelled by the jump of this net, from the initial marking $M_0 = (0, 1, 0)$ to the marking $M_0' = (0, 0, 1)$, and the resuming of its activity by the transition t_3.



$$R = \{((0, 1, 0), (0, 0, 1))\}$$

Figure 3. the jumping net from example 4.1

Now, we will extend the notion of concurrency-degrees from P/T-nets to jumping Petri nets.

Definition 4.1. *Let $\gamma = (\Sigma, R)$ be a jumping Petri net, Σ being the underlying P/T-net of γ, and let M be an arbitrary marking of γ. The* concurrency-degree at the marking M *of the net γ is defined by:*

$$d(\gamma, M) = \max\{\ d(\Sigma, M') \mid MR^*M'\ \}\ .$$

Definition 4.2. *Let $\gamma = (\Sigma, R, M_0)$ be a marked jumping Petri net.*
i) The inferior concurrency-degree *of the net γ is defined by:*

$$d^-(\gamma) = \min\{\ d(\gamma, M) \mid M \in [M_0\rangle_{\gamma,j}\}\ ;$$

ii) The superior concurrency-degree *of the net γ is defined by:*

$$d^+(\gamma) = \max\{\ d(\gamma, M) \mid M \in [M_0\rangle_{\gamma,j}\}\ .$$

Moreover, the remarks about concurrency-degrees of P/T-nets (remark 3.1) hold for jumping Petri nets as well.

Definition 4.3. *Let $\gamma = (\Sigma, R, M_0)$ be a marked jumping Petri net. If $d^-(\gamma) = d^+(\gamma)$, then we denote this number with $d(\gamma)$, i.e. $d(\gamma) = d^-(\gamma) = d^+(\gamma)$, and we called it the* concurrency-degree *of the net γ.*

Example 4.2. *Let us recall the mFJPTN γ from example 4.1. We denote by M_n, M'_n the following markings: $M_n = (n, 1, 0)$, $M'_n = (n, 0, 1)$, for all $n \geq 0$. Thus, the set of jumps is $R = \{(M_0, M'_0)\}$, and it is easy to see that transition t_1 is fireable from any j-reachable marking, transition t_2 is fireable from all markings M_n, $n \geq 1$, and transition t_3 is fireable from all markings M'_n, $n \geq 0$. Therefore, the j-reachability set is $[M_0\rangle_{\gamma,j} = \{M_n | n \geq 0\} \cup \{M'_n | n \geq 0\}$, and the j-reachability graph of γ, $\mathcal{RG}(\gamma)$ (defined in* [ViJ99]*), is shown in figure 4.*

More precisely, $M_0RM'_0$ is the only jump in γ, and $T(M_0) = \{t_1\}$, with $M_0[t_1\rangle_\Sigma M_1$, which means that $d(\Sigma, M_0) = 1$. For all $n \geq 1$ we have that $T(M_n) = \{t_1, t_2\}$, with $M_n[t_1\rangle_\Sigma M_{n+1}$ and $M_n[t_2\rangle_\Sigma M_{n-1}$; moreover, $M_n[\{t_1, t_2\}\rangle_\Sigma M_n$ and $MCT(M_n) = \{\{t_1, t_2\}\}$. Therefore, $d(\Sigma, M_n) = 2$, $\forall n \geq 1$. Also, for all $n \geq 0$ we have that $T(M'_n) = \{t_1, t_3\}$, with $M'_n[t_1\rangle_\Sigma M'_{n+1}$ and $M'_n[t_3\rangle_\Sigma M_n$; moreover, $M'_n[\{t_1, t_3\}\rangle_\Sigma M_{n+1}$ and $MCT(M'_n) = \{\{t_1, t_3\}\}$. So, $d(\Sigma, M'_n) = 2$, $\forall n \geq 0$.

Now, let us compute the concurrency-degrees of the jumping net γ. Since $M_0RM'_0$ is the only jump in γ, we have that $d(\gamma, M_0) = max\{d(\Sigma, M_0), d(\Sigma, M'_0)\} = 2$, $d(\gamma, M_n) = d(\Sigma, M_n) = 2$, $\forall n \geq 1$, and $d(\gamma, M'_n) = d(\Sigma, M'_n) = 2$, $\forall n \geq 0$. Therefore, $d(\gamma) = d^-(\gamma) = d^+(\gamma) = 2$, so the concurrency-degree of γ is 2.

Let us notice that the inferior, respectively superior concurrency-degree of the underlying P/T-net of γ is $d^-(\Sigma) = 1$, resp. $d^+(\Sigma) = 2$; moreover, $d(\Sigma)$ is undefined, and the reachability set is $[M_0\rangle_\Sigma = \{M_n | n \geq 0\}$.

In the sequel, we will show how we can compute the concurrency-degrees of a jumping net.

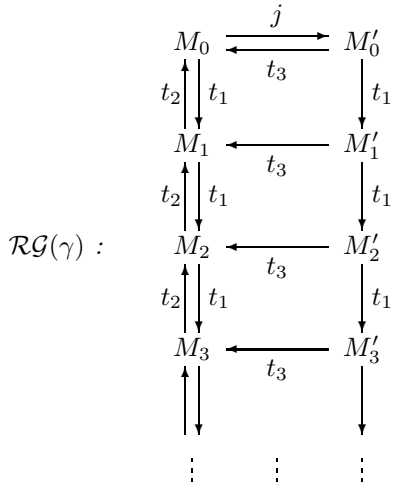


FIGURE 4. the j-reachability graph of the net γ

First of all, let us notice that the concurrency-degree at any marking of a *JPTN* γ can be computed if γ has the property: $\{M' \mid MR^*M'\}$ is finite, for each marking M (this follows easily from the definition 4.1 and the theorem 3.1, because we have to compute a maximum on a finite set). Let us observe that any finite jumping net has this property. Therefore, we have the result:

Theorem 4.1. *The concurrency-degree at a marking, $d(\gamma, M)$, is computable for any FJPTN γ and for any marking M.*

Now, let us notice that the algorithm for computing the superior concurrency-degree of a marked Petri net from section 3 works also for marked finite jumping Petri nets, because the coverability problem is decidable for *mFJPTN* ([TiJ94]) (the function is_coverable from section 3 solves the coverability problem for *mFJPTN* by using the minimal coverability graph, [ViJ99]). As a consequence, we have the result:

Theorem 4.2. *The superior concurrency-degree $d^+(\gamma)$ is computable for any mFJPTN γ.*

Now, we will show how we can compute the inferior concurrency-degree of a marked finite jumping Petri net.

Let $\gamma = (\Sigma, R, M_0)$ be a *mFJPTN*. First, let us remark that if the reachability set $[M_0\rangle_{\gamma,j}$ is finite (this problem is decidable, [TiJ94]), then we can compute the inferior concurrency-degree of γ by using directly the definition: $d^-(\gamma) = \min\{\ d(\gamma, M) \mid M \in [M_0\rangle_{\gamma,j}\}$, because the minimum is computed on a finite set and $d(\gamma, M)$ is computable, for each marking M (theorem 4.1).

Now, let us consider that the reachability set $[M_0\rangle_{\gamma,j}$ is infinite. Then, there exists a finite subset $\mathcal{M} \subseteq [M_0\rangle_{\gamma,j}$ such that

$$(*) \quad \forall M \in [M_0\rangle_{\gamma,j}, \exists M' \in \mathcal{M} \text{ such that } M' \leq M.$$

Indeed, we can consider $\mathcal{M}$ as being the set of minimal reachable markings of γ, i.e.

$$\mathcal{M} = \{M \in [M_0\rangle_{\gamma,j} | \forall M' \in [M_0\rangle_{\gamma,j} - \{M\} : M' \not\leq M\}.$$

Then, we have the result:

Proposition 4.1. *The following equality holds:*

$$\min\{d(\gamma, M) | M \in [M_0\rangle_{\gamma,j}\} = \min\{d(\gamma, M) | M \in \mathcal{M}\}.$$

Proof. This equality follows easily from $(*)$ and from the fact that the concurrency-degree at a marking is a monotone increasing function, i.e. $M_1 \leq M_2 \Rightarrow d(\gamma, M_1) \leq d(\gamma, M_2)$. □

Proceeding from Dickson's lemma (lemma 3.1), it follows that any subset of $\mathbf{N}^k$ contains only finitely many incomparable vectors. Since, by its definition, the elements of $\mathcal{M}$ are incomparable, it follows that $\mathcal{M}$ is a finite set.

Let us show how the set $\mathcal{M}$ can be constructed. Let $\gamma = (\Sigma, R, M_0)$ be a *mFJPTN*, with $R \neq \emptyset$, i.e.

$$R = \{ \ (M_i', M_i'') \ | 1 \leq i \leq n\}, \quad n \geq 1,$$

such that $M_i' \in [M_0\rangle_{\gamma,j}$ (this can be done, see [TiJ94]). As in [TiJ94], we associate to γ the following *mPTN*s:

$$\gamma_0 = (\Sigma, M_0) \ \text{ and } \ \gamma_i = (\Sigma, M_i''), \ \text{for each } 1 \leq i \leq n,$$

and then, let $\mathcal{M}_i$ be the set of minimal reachable markings of γ_i, for each $1 \leq i \leq n$. These sets are finite (it follows from Dickson's lemma) and we have that:

$$\mathcal{M} = \{M \in \mathcal{M}' \mid \forall M' \in \mathcal{M}' - \{M\} \ : \ M' \not\leq M\},$$

where $\mathcal{M}' = \cup\{\mathcal{M}_i | 1 \leq i \leq n\}$. Thus, since the sets $\mathcal{M}_i$, $1 \leq i \leq n$, are computable, the set $\mathcal{M}$ is also computable, and from proposition 4.1 it follows that:

Theorem 4.3. *The inferior concurrency-degree $d^-(\gamma)$ is computable for any mFJPTN γ.*

5. Conclusions

In this paper we have extended some concepts (mainly, concurrency-degrees) from the class of Place/Transition Petri nets (*PTN*) to the class of jumping Petri nets (*JPTN*). Also, we have presented a simpler definition of concurrency-degree for *PTN* and we have shown how we can compute concurrency-degrees.

Many problems remain to be studied, for example:

- finding an efficient algorithm for computing the set $\mathcal{M}$ for Petri nets;

- finding better algorithms for computing the concurrency-degrees for Petri nets;
- extending the computability results regarding concurrency-degrees for *FJPTN* for the larger class of jumping Petri nets.

References

[BeF86] E. Best, C. Fernandez: *Notations and Terminology on Petri Net Theory*, Arbeitspapiere der GMD 195, 1986.

[Dic13] L.E. Dickson: *Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors*, American Journal of Mathematics 35, 1913, pp. 413-422.

[Fin93] A. Finkel: *The Minimal Coverability Graph for Petri Nets*, Advanced in Petri Nets 1993, LNCS 674, Springer-Verlag, 1993, pp. 210-243.

[JuT99] T. Jucan, F.L. Ţiplea: *Reţele Petri. Teorie şi practică*, The Romanian Academy Publishing House, Bucharest, 1999.

[KaM69] R.M. Karp, R.E. Miller: *Parallel Program Schemata*, Journal of Computing System Science (4), 1969, pp. 145-195.

[Rei85] W. Reisig: *Petri Nets. An Introduction*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[Rei87] W. Reisig: *Place Transition Systems*, Advanced in Petri Nets 1986, Part I, LNCS 254, Springer-Verlag, 1987, pp. 117-141.

[TJD93] F.L. Ţiplea, T. Jucan, Şt. Dumbravă: *Modeling Systems by Petri Nets with Different Degrees of Concurrency*, Proc. 14^{th} International Symposium on Automatic Control and Computer Science SACCS'93, 1993, pp. 48-54.

[TiJ94] F.L. Ţiplea, T. Jucan: *Jumping Petri Nets*, Foundations of Computing and Decision Sciences, vol. 19, no.4, 1994, pp. 319-332.

[TiM97] F.L. Ţiplea, E. Mäkinen: *Jumping Petri Nets. Specific Properties*, in Fundamenta Informaticae 32, 1997, pp. 373-392.

[ViJ99] C. Vidraşcu, T. Jucan: *On Coverability Structures for Jumping Petri Nets*, to appear in Scientific Annals of the "Al. I. Cuza" University of Iaşi, Computer Science Section, Tome IX, 2000.

Faculty of Computer Science, "Al. I. Cuza" University, Iaşi, România
E-mail address: {jucan,vidrascu}@infoiasi.ro

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

F-BOUNDED QUANTIFICATION AND THE MATCHING RELATION

SIMONA MOTOGNA

Abstract. The introduction of F-bounded polymorphism had proved to have great benefits in specification of object oriented languages, and the most important one, concerning binary methods and recursion definition, is also presented here. Then many other systems had appeared, such as PolyTOIL, based on the matching relation, and we prove that F-bounded quantification and the matching relation are equivalent.

1. F-bounded polymorphism

The extension proposed by the Abel group [CHC90] is based on introducing recursive types. The specification of polymorphic functions over objects cannot be done correctly through bounded quantification, and the recursive functions give a suitable solution. That's why we will study their behaviour in the next paragraph.

1.1. **Subtyping and recursion using bounded quantification.** The bounded quantification was introduced in the language Fun [CW85] in order to type polymorphic functions, that were defined over "simple" objects, represented as records. But in most cases, class definitions include the so called binary methods, namely methods with parameters of type representing that class. This situation imposes the condition that objects should be described as recursive records[CHC90], destroying their "simplicity". We will see what happens with the behaviour of the polymorphic functions over recursive objects and we will show that, in this case, bounded quantification fail to produce a correct answer [Ghel93].

When describing the recursive types, the two possible situations can be specified using the notion of polarity, that is "borrowed" from logic [CCHOM89].

In a type expression $s \to t$, the subexpression s occurs *negatively*, and the subexpression t occurs *positively*.

Let's consider a recursive type $SortList$, representing a sorted chained list, whose elements are of an arbitrary type t. We have also defined a method for inserting elements in the list, and the list is described as having a *head* (an element

1991 *CR Categories and Descriptors.* D.3.1 [**Programming Languages**]: Formal Definitions and Theory – *Semantics*; D.3.3 [**Programming Languages**]: Language Constructs and Features – *Recursion*.

of type t) and the rest of the list, the *tail* of type $SortList$. In addition, in order to have a sorted list, it is necessary to have a total order relation defined between the list elements, so the type t will be a subtype of the type:

$$TotalOrder = \lambda\sigma.\{smaller : \sigma \to Bool, equal : \sigma \to Bool\}$$

(the method *insert* will be written knowing that each element is comparable with the others).

So, the sorted chained list will have the following form:

$$SortList = \forall t \leq TotalOrder.\mu sl.\{insert : t \to sl, head : t, tail : sl\}$$

If we want to have such a list of integer numbers, than the parameter t must be replaced with Int:

$$SortIntList = \mu sil.\{insert : Int \to sil, head : Int, tail : sil\}$$

that seems to be intuitively correct.

But, if t was replaced by Int, then the following condition must be satisfied:

$$Int \leq TotalOrder$$

and the type Int is, in general, defined as:

$$Int = \mu n.\{smaller : n \to Bool, equal : n \to Bool, ...\}$$

and if we decompose under the recursion variable, we obtain:

$$\begin{array}{ll} \{ & smaller : Int \to Bool, equal : Int \to Bool, ...\} \leq \\ & \{smaller : TotalOrder \to Bool, equal : TotalOrder \to Bool\} \end{array}$$

In order to satisfy this equation, each common component from the two records must obbey the subtyping rule for functions:

$$\frac{s \leq s', t' \leq t}{s' \to t' \leq s \to t}$$

and we obtain:

$$TotalOrder \leq Int$$

which represents a contradiction of what we really want.

So, we notice that the only valid substituion for t is $TotalOrder$. We can end with the following conclusion: bounded quantification does not specify correctly the behaviour of the polymorphic function defined over recursive objects.

1.2. **F-bounded quantification. Definition 1.1**: We say that a universal quantified type is ***F-bounded*** if it has the following form:

$$\forall t \leq F[t].\sigma$$

where $F[t]$ is an expression that, in general, contains the type variable t.

The F-bounded polymorphic types differ from the ordinary bounded types in the sense that both the result type σ and the bound $F[t]$ of the type depend upon the type variable t.

If $F[t]$ is a type of the form $F[t] = \{a_i : \sigma_i[t]\}$, then the condition $A \leq F[A]$ says, in fact, that A must contain all the methods a_i and these methods should have the arguments specified by $\sigma_i[A]$, that are defined depending on A. In conclusion, A will often be a recursive type, suggesting that the functional bounded quantification is strongly connected with type recursivity.

Intutively, F-bounded quantification characterizes the types that have a "recursive structure" similar to the type $\mu t.F[t]$. The type $F[A]$ describes a set of operations, which can accept values of type A as arguments and may return such values as results. The elements of type A have these operations, if we consider each element of type A as an element of type $F[A]$, namely $A \leq F[A]$.

A type that always satisfies $A \leq F[A]$ is the recursive type $A = \mu t.F[t]$. More generally, if $G[t]$ is a type expression and $G[t] \leq F[t]$ for any t, then the recursive type $A = \mu t.G[t]$ satisfies $A \leq F[A]$.

The F-bounded quantification has a major impact upon the relation between inheritance and subtyping in object oriented programming: two types t_1 and t_2 can satisfy a F-bound ($t_1 \leq F[t_1]$ and $t_2 \leq F[t_2]$) but may not be in a subtyping relation ($t_1 \not\leq t_2$ and $t_2 \not\leq t_1$). This means that a F-bounded function can be applied to (or inherited by) an object with incomparable types, proving that the inheritance hierarchy is different from the subtyping hierarchy.

The F-bounded polymorphism will allow, in general, to write functions that work in the same time on objects belonging to classes that are in an inheritance relation, in the same way in which bounded polymorphism allows to write functions that work in the same time on types and their subtypes.

1.3. **Subtyping and recursion using F-bounded polymorphism.** In the first paragraph we noticed that Int is not a subtype of the type $TotalOrder$. However, the types Int and $TotalOrder$ have the same binary operations: *smaller* and *equal*. So, the expression $x.smaller(y)$ is correctly typed if x and y have both one of these two types and incorrectly typed if they have different types.

In the following, we will see that for such a behaviour the subtyping relation between the two types is not necessary, and we can introduce another relation that guarantees the desired behaviour.

This common structure of the two types can be described through a functional type, derived from the recursive definition of the type $TotalOrder$. This functional

type represents a F-bound. The "F-" notation before the type indicates that it represents a F-bound:

$$F - TotalOrder[t] = \{smaller : t \to Bool, equal : t \to Bool\}$$

Applying this function for Int we obtain:

$$F - TotalOrder[Int] = \{smaller : Int \to Bool, equal : Int \to Bool\}$$

and

$$Int \leq F - TotalOrder[Int]$$

because the contravariance is respected for each component: $Int \leq Int$.

We can now build the F-bounded polymorphic definition of the sorted chained list:

$$\begin{aligned} SortList \quad = \quad & \forall t \leq F - TotalOrder[t].\mu sl. \\ & \{insert : t \to sl, head : t, tail : sl\} \end{aligned}$$

and then

$$SortIntList = \mu sil.\{insert : Int \to sil, head : Int, tail : sil\}$$

will be a valid definition.

Let's notice that $Int \not\leq TotalOrder$, but $Int \leq F - TotalOrder[Int]$ which is enough.

As a conclusion we may say that F-bounded quantification allows us to use generic polymorphism together with inheritance in object-oriented languages: we have a generic class $SortList$ that can be specialised substituting the parameter, obtaining, for example, $SortIntList$. Also, the other operation for creating derived classes, adding new characteristics, will function correctly. For example, if we want to obtain a merged sorted list, we have:

$$\begin{aligned} MergedSortList \quad = \quad & \forall t \leq F - TotalOrder[t].\mu msl. \\ & \{insert : t \to msl, head : t, tail : msl, merge : msl \to msl\} \end{aligned}$$

The recursion variable will ensure that the methods always return the desired type, namely $MergedSortList$ and any object of this class is a member of class $SortList$.

2. The matching relation

Kim Bruce's contribution to object oriented programming specification consists in the description of the type systems and the semantics of several object oriented languages, each of them introducing new elements: TOOPLE [Bruc93], TOIL [BvG93] and the most remarkable one PolyTOIL [BSvG95].

PolyTOIL is an object oriented programming language, polymorphic, with static typing, and its type system had been proved to be correct.

Bruce's idea that inovates this domain is the separation of the subtyping from the inheritance definition, assigning name to the type of $self$ and defining the type checking rules. Inheritance is no longer expressed as subtyping (the system contains two hierarhies, one for subtypes and one for subclasses) but as a *matching* between the object types and is defined as follows:

Definition 2.1 $ObjectType\ \tau'$ *matches* (is in a *matching* relation) with $ObjectType\ \tau$, denoted $ObjectType\ \tau' <\#\ ObjectType\ \tau$, if for each method name m from τ there exists a corresponding method in τ' and m's type in τ' is a subtype of the method type from τ;

or (more formally)

$$ObjectType\,\{m_j : S_j\}_{1\leq j\leq m} <\#\ ObjectType\,\{m_i : T_i\}_{1\leq i\leq n}$$

if and only if

$$n \leq m \text{ and for every } i \leq i \leq n, S_i \leq T_i$$

where $ObjectType\,\{m_j : S_j\}$ represents the type of objects that have methods m_j with type S_j.

We will notice that the definition doesn't contain instance variables, since it is assumed that they are not visible from outside and can be accessed only through the object's method.

The subtyping relation is denoted as $S \leq T$. If SC is a subclass of the class C then $SCType <\# CType$.

The subtyping relation is defined as follows [BSvG95]:

Definition 2.2 $ObjectType\,\{m_j : S_j\}_{1\leq j\leq m} \leq ObjectType\,\{m_i : T_i\}_{1\leq i\leq n}$ if and only if

1. $n \leq m$ and for every $1 \leq i \leq n, S_i \leq T_i$
2. $\forall T_i, 1 \leq i \leq n, T_i$ is NOT contravariant to $MyType$ (there are no parameters of type $MyType$, the type of the $self$ variable, in supertypes).

Remark 2.1: The only difference between subtyping and matching is the second condition from the definition. So, if two object types are in a subtyping relation then they are in a matching relation, but the reverse statement is false [AC95].

3. The equivalence between the matching relation and F-bounded quantification

Theorem 2.1 [Moto00] If there exists a matching relation between two object types:

$$ObjectType\ t' <\#\ ObjectType\ t$$

then *ObjectType t* is a F-bound for *ObjectType t′* and the reverse statement is also true, namely there exists a matching relation between an object type and its F-bound.

Proof:
"⇒"

Let *ObjectType t* be the object type with the following form: $\{m_i : T_i\}_{1 \leq i \leq n}$. For every $1 \leq i \leq n$, the type T_i has the following form: $\sigma_i \rightarrow \tau_i$, so they are functional types, because m_i represent methods (*ObjectType t* is an object type, and the instance variables are hidden and only the methods are visible). In addition, σ_i and/or τ_i can depend on $MyType$, which represents the $self$ type of the object. The $self$ variable can be considered as a recursion variable and then we can describe the object type as follows:

$$ObjectType\ t = \mu MyType.\{m_i : T_i[MyType]\}_{1 \leq i \leq n}$$

If we create an object of type *ObjectType t′* of the form: $\{m_j : S_j\}_{1 \leq j \leq m}$, that inherits from *ObjectType t*, then the two types are in matching relation and:

$$n \leq m \text{ and } S_i \leq T_i.$$

On the other hand, if we use F-bounded quantification, any object definition based on the object type *ObjectType t* must respect the F-bound condition:

$$ObjectType\ t' \leq F - ObjectType\ t[t'] \tag{1}$$

Using the model described in the paragraph 1.3, the computation of the F-bound will produce the following result:

$$F - ObjectType\ t[MyType] = \{m_i : T_i\}_{1 \leq i \leq n}$$

and the condition (1) is satisfied because *ObjectType t′* has all the m_i methods (and probably more), and the arguments of these methods are correctly specified, since $S_i \leq T_i$.

In conclusion, the proof of this implication consists in noticing that $MyType$ implies a recursive definition of the type.

"⇐"

According to the definition of F-bounded quantification, we have: if $F[t]$ is a type of the form $F[t] = \{a_i : T_i[t]\}_{1 \leq i \leq n}$, then the condition $A \leq F[A]$ says, in fact, that: [1] A must contain all a_i methods and [2] these methods must have the arguments specified by $T_i[A]$, which are defined depending on A.

So, from the condition [1] we obtain that *ObjectType t′* will have the form $\{a_i : S_i, a_{n+1} : S_{n+1}, ...\}$, and from [2] we can deduce that the types S_i, for $1 \leq i \leq n$, are obtained substituting the t parameter with the object type, so:

$$S_i = T_i[t/ObjectType\ t']$$

and it is obvious that $S_i \leq T_i$.

So, if

$$ObjectType\ t' \leq F - ObjectType\ t[ObjectType\ t']$$

then

$$ObjectType\ t' < \#\ F - ObjectType\ t[ObjectType\ t']$$

Remark 2.2: This implication can be proved easier if we use ***Remark 2.1***, that states that if two object types are in a subtyping relation, then they are also in a matching relation and noticing that a F-bound respects the structure of an object type.

References

[AC95] M Abadi, L Cardelli - *On subtyping and matching*, Proc. 9th European Conf. Object-Oriented Prog., Aarhus, Denmark, 1995

[Bruc93] K.Bruce - *Safe type checking in a statically-typed object-oriented programming language*, Proc. ACM Symposium on Principles of Programming Languages, 1992, pag. 316-327

[BSvG95] K.Bruce, A. Schuett, R. van Gent - *PolyTOIL: A type-safe polymorphic object-oriented language*, ECOOP '95 Proceedings, LNCS 952, Springer-Verlag, pag. 27-51.

[BvG93] K. Bruce, R. van Gent - *TOIL: A new types-safe object-oriented imperative language*, Technical Report, Williams College, 1993

[CCHOM89] P. Canning, W. Cook, W. Hill, W. Olthoff, J. Mitchell - *F- bounded quantification for object oriented programming*, in Proc. Functional Programming Languages and Computer Architectures, 1989, pag. 273-280

[CHC90] W. Cook, W. Hill, P. Canning - *Inheritance is not subtyping*, Proc. 17th ACM Symp. Principles of Prog. Lang.,1990, pag. 125-135.

[CW85] L. Cardelli, P. Wegner - *On understanding types, data abstraction and polymorphism*, ACM Computing Surveys, 17(4),1985, pag. 471-521.

[Ghel93] G. Ghelli - *Recursive types are not conservative over* $F_{\leq}$, in Typed Lambda-Calculus and Applications, ed. M. Dezani-Ciancaglini, G. Plotkin, Springer Verlag, 1993

[Moto00] S. Motogna - *Formal approach to object oriented languages*, Ph.D. Thesis,"Babes-Bolyai" University, Cluj-Napoca, Romania, September 2000

Faculty of Mathematics and Informatics, "Babeş–Bolyai" University, 3400 Cluj–Napoca

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

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

A GENERAL CLASS OF NONPRODUCT QUADRATURE FORMULAS

PETRU P. BLAGA

ABSTRACT. **Abstract.** A general class of fifth degree approximate integration formulas for hypercubes is constructed. If the integrand is a real function of n independent real variables, then a $2^n + \binom{n}{k}2^k + 1$ point class nonproduct quadraturae is obtained. A lot of known multiple quadrature formulas are included in this class. In the particular cases $n = 2, 3$, comparative numerical examples are considered.

1. INTRODUCTION

Here we consider an approximate evaluation to the multiple definite integral

$$I_n[f] = \int_{-1}^{1} \dots \int_{-1}^{1} f(x_1, \dots, x_n)\, dx_1 \dots dx_n. \tag{1}$$

A very wide class of $2^n + \binom{n}{k}2^k + 1$ point nonproduct quadrature rules of 5th degree is obtained. The evaluation points of integrated function f are symmetrically placed inside of the domain $S_n = [-1, 1]^n$ of the integral $I_n[f]$. As such the rule is required to be exact for the monomials of degree 0, 2, 4. Constructed approximate integration formulas extend the 2^n+2n+1 point quadrature presented in [2] and $2^n + 2^{n-1}n + 1$ point quadrature considered in [3], and also that of Das and Pradham [5] (see also Blaga [1]). For $n = 2$ and $n = 3$ other known multiple quadrature formulas are obtained: Blaga [1],[2],[3], Burnside [4] (see also Stroud [9], p. 248), Hammer and Stroud [8] (see also Stroud [9], p. 231), Mustard, Lyness, Blatt [7], and product Gauss formula (see Stroud [9], p. 249).

2. FIFTH DEGREE INTEGRATION FORMULA

Let us take the following $N = 2^n + \binom{n}{k}2^k + 1$ points: $(\beta_1, \dots, \beta_n)$, $(\gamma_1, \dots, \gamma_n)$ and $(0, \dots, 0)$, where each β_i $(1 \leqslant i \leqslant n)$ is either $-\lambda\alpha$ or $+\lambda\alpha$, i.e. the corners of the hypercube $[-\lambda\alpha, +\lambda\alpha]^n$, and $n - k$ (and only $n - k$) of the γ_i $(1 \leqslant i \leqslant n)$ are zero and all the others k of γ_i equal either $-\alpha$ or α $(1 \leqslant k < n)$. On the one hand we have that the number of $(\beta_1, \dots, \beta_n)$ type points is 2^n, on the other hand the number of $(\gamma_1, \dots, \gamma_n)$ type points is $\binom{n}{k}2^k$, and the last type of points

1991 *Mathematics Subject Classification.* 65D32.

are situated at the corners of hypercubes $[-\alpha,\alpha]^k$ from the hyperplanes given by $x_{j_s}=0$ $(s=\overline{1,n-k})$, $1\leqslant j_1<\cdots<j_{n-k}\leqslant n$, respectively. Taking into account that the center of the hypercube S_n is also considered it results that $N=2^n+\binom{n}{k}2^k+1$. In order to all evaluation points belong to the hypercube S_n the real positive parameters λ and α must satisfy the conditions $\alpha<1$ and $\lambda\alpha\leqslant 1$.

We shall construct an approximate rule to $I_n[f]$ of the following type

$$\begin{aligned} Q_{n,k}[f]=A_0 f(0,\ldots,0)+A_1\sum\nolimits_1 f(\gamma_1,\ldots,\gamma_n)\\ +A_2\sum\nolimits_2 f(\beta_1,\ldots,\beta_n). \end{aligned} \tag{2}$$

As we have seen, in the formula (2) the the first sum, $\sum_1$, has $\binom{n}{k}2^k$ terms, and the second sum, $\sum_2$, has 2^n terms. Such that the quadrature (2) will be a $2^n+\binom{n}{k}2^k+1$ point (generally) nonproduct formula.

The coefficients A_0, A_1, A_2 and the parameters α and λ will be determined such as to make the rule exact for all monomials of degree less or equal to five, i.e.

$$Q_n[f]=I_n[f], \tag{3}$$

for

$$f=x_1^{k_1}\ldots x_n^{k_n},\quad \text{where}\quad 0\leqslant k_1+\cdots+k_n\leqslant 5. \tag{4}$$

We remark the exactness of the formula (3) for all monomials (4) containing at least one odd power k_i. On the other hand, taking into account that the formula (2) has the evaluation points of the function f symmetrically situated over the integration domain S_n (if $\alpha\in(0,1)$ and $\lambda\alpha\in(0,1]$), we have to require that (3) to be exact only for the monomials

$$f=1,\ x_1^2,\ x_1^4, x_1^2x_2^2, \tag{5}$$

to obtain a fifth degree exactness quadrature formula.

The exactness conditions of the formula (3) for the monomials (5) give the following nonlinear algebraic system in A_0, A_1, A_2 and α, λ:

$$\begin{cases} A_0 & +\binom{n}{k}2^kA_1 & +2^nA_2 & =2^n\\ & \binom{n-1}{k-1}2^k\alpha^2A_1 & +2^n\lambda^2\alpha^2A_2 & =\frac{1}{3}2^n\\ & \binom{n-1}{k-1}2^k\alpha^4A_1 & +2^n\lambda^4\alpha^4A_2 & =\frac{1}{5}2^n\\ & \binom{n-2}{k-2}2^k\alpha^4A_1 & +2^n\lambda^4\alpha^4A_2 & =\frac{1}{9}2^n, \end{cases} \tag{6}$$

with $\binom{n-2}{-1}=0$.

To obtain the solution of system (6), we must remark the special case $5n-9k+4=0$, i.e. $(n,k)\in\left\{(9\ell+1,5\ell+1)\ \middle|\ \ell=1,2,\ldots\right\}$. In this case, from the last

two equations of (6), it is obtained that $A_2 = 0$, and the system (6) is equivalent with

$$\left\{\begin{array}{lll} A_0 & +\binom{n}{k}2^k A_1 & = 2^n \\ & \binom{n-1}{k-1}2^k\alpha^2 A_1 & = \frac{1}{3}2^n \\ & \binom{n-1}{k-1}2^k\alpha^4 A_1 & = \frac{1}{5}2^n. \end{array}\right. \tag{7}$$

From the last two equations of (7) results that $\alpha^2 = \frac{3}{5}$, and then

$$A_1 = \frac{5}{9}\frac{2^{n-k}}{\binom{n-1}{k-1}} \quad \text{and} \quad A_0 = \frac{2^{n+2}}{9k}\cdot$$

So, in this case we obtain the following $\binom{n}{k}2^k + 1$ point 5th degree formula

$$\begin{aligned} Q_{n,k}[f] =& \frac{2^{n-k}}{9}\left[\frac{2^{k+2}}{k}f(0,\ldots,0)\right. \\ &\left. + \frac{5}{\binom{n-1}{k-1}}\sum_{1\leqslant i_1<\cdots<i_k\leqslant n} f\Bigg(0,\ldots,\underset{\substack{\uparrow\\ i_1}}{\pm\sqrt{\frac{3}{5}}},\ldots,\underset{\substack{\uparrow\\ i_k}}{\pm\sqrt{\frac{3}{5}}},\ldots,0\Bigg)\right]. \end{aligned} \tag{8}$$

One remarks that the smallest n dimensional formula is obtained for $n = 10$ ($k = 6$), and it is a 13341 point 5th degree formula.

In the following we consider that $5n - 9k + 4 \neq 0$.

From the fourth and third equations of (6) we get

$$A_2 = \frac{5n - 9k + 4}{45\,(n-k)\,\lambda^4\alpha^4},$$

and then taking into account the second equation of (6) we obtain the following equivalent relations between the parameters λ and α:

$$\alpha^2 = \frac{4\,(n-1)\,\lambda^2 + 5n - 9k + 4}{15\,(n-k)\,\lambda^2}, \quad \lambda^2 = \frac{5n - 9k + 4}{15\,(n-k)\,\alpha^2 - 4\,(n-1)}\cdot$$

Finally, we get the solution of the system (6)

$$\begin{aligned} A_0 &= 2^n\frac{5k\,(n-k)\left(9\lambda^4\alpha^4 - 1\right) - 4n\,(n-1)\,\lambda^4 + 4k\,(k-1)}{45k\,(n-k)\,\lambda^4\alpha^4} \\ &= -2^{n+2}\frac{45k\,(k-1)\,\alpha^4 - 30k\,(n-1)\,\alpha^2 + (n-1)\,(5n+4)}{45k\,(5n-9k+4)\,\alpha^4}, \\ A_1 &= \frac{2^{n-k+2}}{45\binom{n-2}{k-1}\alpha^4}, \\ A_2 &= \frac{\left[15\,(n-k)\,\alpha^2 - 4\,(n-1)\right]^2}{45\,(n-k)\,(5n-9k+4)\,\alpha^4}\cdot \end{aligned} \tag{9}$$

Thus we have obtained the following quadrature rule

$$
\begin{aligned}
Q_{n,k}[f] = \frac{1}{45\alpha^4}\Bigg[& -2^{n+2}\frac{45k(k-1)\alpha^4-30k(n-1)\alpha^2+(n-1)(5n+4)}{k(5n-9k+4)} \\
& \times f(0,\ldots,0)+\frac{2^{n-k+2}}{\binom{n-2}{k-1}}\sum\nolimits_1 f(\gamma_1,\ldots,\gamma_n) \\
& +\frac{\left[15(n-k)\alpha^2-4(n-1)\right]^2}{(n-k)(5n-9k+4)}\sum\nolimits_2 f(\beta_1,\ldots,\beta_n)\Bigg].
\end{aligned} \tag{10}
$$

In the following table we resume the conditions for α^2 and λ^2 such that all evaluation points in the quadrature formula (10) are placed inside of the domain S_n.

	α^2	λ^2
$5n+4>9k$	$\left[\frac{2(n-1)}{5n-3k-2};1\right)$	$\left(\frac{5n-9k+4}{11n-15k+4};\frac{5n-3k-2}{2(n-1)}\right]$
$5n+4<9k$	$\left(0;\frac{2(n-1)}{5n-3k-2}\right]$	$\left(\frac{9k-5n-4}{4(n-1)};\frac{5n-3k-2}{2(n-1)}\right]$

For the first two values of n ($n=2$,$n=3$), the coefficients of quadrature formula (10) and the relation between the parameters α and λ are given in the following table

	A_0	A_1	A_2	
$n=2,\ k=1$	$\frac{32\left(15\alpha^2-7\right)}{225\alpha^4}$	$\frac{8}{45\alpha^4}$	$\frac{\left(15\alpha^2-4\right)^2}{225\alpha^4}$	$\alpha^2=\frac{4\lambda^2+5}{15\lambda^2}$
$n=3,\ k=1$	$\frac{32\left(30\alpha^2-19\right)}{225\alpha^4}$	$\frac{16}{45\alpha^4}$	$\frac{\left(15\alpha^2-4\right)^2}{225\alpha^4}$	$\alpha^2=\frac{4\lambda^2+5}{15\lambda^2}$
$n=3,\ k=2$	$\frac{32\left[1-5\left(3\alpha^2-2\right)^2\right]}{45\alpha^4}$	$\frac{8}{45\alpha^4}$	$\frac{\left(15\alpha^2-8\right)^2}{45\alpha^4}$	$\alpha^2=\frac{8\lambda^2+1}{15\lambda^2}$

3. Particular Cases

1. *Generalized Das-Pradham quadrature formula* [5] (see also [1],[2] and [3]). This quadrature is obtained considering $\alpha^2=\frac{2(n-1)}{5n-3k-2}$ or equivalently $\lambda^2=\frac{5n-3k-2}{2(n-1)}$

($\lambda\alpha=1$). In this case from (9) we get

$$A_0 = -2^n \frac{5n\,(5n-9k-4)+4\,(9k+1)}{45k\,(n-1)},$$
$$A_1 = 2^{n-k} \frac{(5n-3k-2)^2}{45\,(n-1)^2\binom{n-2}{k-1}},$$
$$A_2 = \frac{5n-9k+4}{45\,(n-k)},$$

and the corresponding quadrature formula is

$$\begin{aligned} Q^{[1]}_{n,k}[f]) = \frac{1}{45}\Bigg[&- 2^n \frac{5n\,(5n-9k+4)+4\,(9k+1)}{k\,(n-1)} f\,(0,\ldots,0) \\ &+ 2^{n-k} \frac{(5n-3k-2)^2}{(n-1)^2\binom{n-2}{k-1}} \sum\nolimits_1 f\,(\gamma_1,\ldots,\gamma_n) \\ &+ \frac{5n-9k+4}{n-k} \sum\nolimits_2 f\,(\beta_1,\ldots,\beta_n)\Bigg]. \end{aligned} \tag{11}$$

This formula has been constructed in [1].

We also remark that formula (11) gives the Das–Pradham quadrature formula (see [5] and [3]), in the case $k = n-1$, and Mustard–Lyness–Blatt quadrature (see [7] and [2]), in the case $k = 1$.

2. *Quadrature formula with the same coordinates of evaluation points.* This quadrature is obtained considering $\lambda^2 = 1$ or equivalently $\alpha^2 = \frac{3}{5}$. In this case from (9) we get

$$A_0 = 2^{n+2} \frac{5+9k-5n}{81k}, \quad A_1 = \frac{5}{81} \frac{2^{n-k+2}}{\binom{n-2}{k-1}}, \quad A_2 = \frac{5}{81} \frac{5n-9k+4}{n-k},$$

and the corresponding quadrature formula is

$$\begin{aligned} Q^{[2]}_{n,k}[f] = \frac{1}{81}\Bigg[&2^{n+2} \frac{5+9k-5n}{k} f\,(0,\ldots,0) \\ &+ 2^{n-k+2} \frac{5}{\binom{n-2}{k-1}} \sum\nolimits_1 f\,(\gamma_1,\ldots,\gamma_n) \\ &+ \frac{5\,(5n-9k+4)}{n-k} \sum\nolimits_2 f\,(\beta_1,\ldots,\beta_n)\Bigg]. \end{aligned} \tag{12}$$

Cases $k = 1$ and $k = n-1$ are presented in [2] and [3] respectively. In the case $n = 2$ the product Gauss quadrature formula is obtained (see [9], p.249) which is also a particular case of Hammer and Stroud quadrature formula (see [6] and [9], p. 231).

Here is also included the quadrature formula (8) ($5n-9k+4=0$).

3. *Quadrature formula with* $A_0 = 0$. This quadrature is a nonproduct $\left[2^k + \binom{n}{k}\right] 2^{n-k}$ point of 5th exactness degree formula. From the condition $A_0 = 0$, it results the algebraic equation

$$45k\,(k-1)\,\alpha^4 - 30k\,(n-1)\,\alpha^2 + (n-1)\,(5n+4) = 0. \tag{13}$$

It must be considered the following two cases:

Case $k = 1$, when $\alpha^2 = \frac{5n+4}{30}$ or equivalently $\lambda^2 = \frac{10}{5n-4}$ ($\lambda^2\alpha^2 = \frac{5n+4}{3(5n-4)}$). One remarks that only for $2 \leqslant n \leqslant 5$ all evaluation points are inside of S_n. Using formulas (9) we have

$$A_1 = 2^{n+3}\frac{5}{(5n+4)^2}, \quad A_2 = \left(\frac{5n-4}{5n+4}\right)^2.$$

The corresponding quadrature has been given in [2].

In the case $1 < k < n$, the equation (13) with the unknown α^2 has real solutions only for $5n - 9k + 4 > 0$, i.e. $2 \leq k \leq \left[\frac{5n+4}{9}\right]$, and necessarily $n > 2$.

Using results of the two cases and the formulas (9), in Table 1 we give the admissible solutions and the corresponding elements of $Q_{n,k}^{[2]}[f]$ for $n = \overline{2,9}$.

Cases $n = 2$ and $n = 3$ with $k = 1$ lead to the Burnside quadrature formula [4] (see also [8], [9] p. 233 and p. 248, and [2], [3]), and respectively the Hammer–Stroud quadrature formula [6] (see also [9], p.263 and [2]).

From the Table 1, we observe there are two formulas when $n = 8, k = 4$ and $n = 9, k = 5$, and there not exists any quadrature when $n = 9, k = 2$.

One also remarks that in the two cases the coefficients A_1 and A_2 are positive.

4. *Quadrature with positive coefficients*. We observe that all the time the coefficient A_1 is positive. Consequently, in order to obtain quadrature formulas of the type (10) with positive coefficients it must that $A_0 > 0$ and $A_2 > 0$.

Again, it must be considered the following two cases:

Case $k = 1$, when $\alpha^2 > \frac{5n+4}{30}$. One remarks that only for $2 \leqslant n \leqslant 5$ all evaluation points are inside of S_n. Using formulas (9) we have

$$A_0 = 2^{n+2}\frac{30\alpha^2 - 5n - 4}{225\alpha^4} > 0, \quad A_1 = 2^{n+1}\frac{1}{45\alpha^4} > 0, \quad A_2 = \frac{1}{\lambda^4\alpha^4} > 0.$$

Such a quadrature has been obtained in [2].

In the case $1 < k < n$, from $A_2 > 0$ and $A_0 > 0$, it must be determined $\alpha^2 \in \left(\frac{2(n-1)}{5n-3k-2}; 1\right)$ satisfying the inequality

$$45k\,(k-1)\,\alpha^4 - 30k\,(n-1)\,\alpha^2 + (n-1)\,(5n+4) < 0.$$

Using results of the two cases, in Table 2 we give the admissible intervals of parameter α^2 for $n = \overline{2,9}$.

4. Error Analysis

If the integrated function $f \in C^6(S_n)$, then the Taylor's formula is valid:

$$f(x) = \sum_{i=0}^{5} \frac{1}{i!} \left(x_1 \frac{\partial}{\partial x_1} + \cdots + x_n \frac{\partial}{\partial x_n} \right)^{(i)} f(0) + \frac{1}{6!} \left(x_1 \frac{\partial}{\partial x_1} + \cdots + x_n \frac{\partial}{\partial x_n} \right)^{(6)} f(\xi), \ \xi \in S_n.$$

Taking into account the error

$$R_{n,k}[f] = I_n[f] - Q_{n,k}[f] = 0,$$

for all monomials $f(x) = x_1^{k_1} \dots x_n^{k_n}$, $0 \leqslant k_1 + \cdots + k_n \leqslant 5$, it results that

$$R_{n,k}[f] = \frac{1}{6!} R_{n,k} \left[\left(x_1 \frac{\partial}{\partial x_1} + \cdots + x_n \frac{\partial}{\partial x_n} \right)^{(6)} f(\xi) \right].$$

Moreover, having the evaluation points symmetrically situated inside of S_n, we obtain that

$$R_{n,k}[f] = \frac{1}{6!} \Bigg\{ R_{n,k}\left[x_1^6\right] \sum_{i=1}^{n} \frac{\partial^6 f(\xi)}{\partial x_i^6} + 15 R_{n,k}\left[x_1^4 x_2^2\right] \sum_{i \neq j} \frac{\partial^6 f(\xi)}{\partial x_i^4 \partial x_j^2} + 90 R_{n,k}\left[x_1^2 x_2^2 x_3^2\right] \sum_{i<j<\ell} \frac{\partial^6 f(\xi)}{\partial x_i^2 \partial x_j^2 \partial x_\ell^2} \Bigg\},$$

and consequently

$$|R_{n,k}[f]| \leqslant \frac{1}{6!} \left\{ L \left| R_{n,k}\left[x_1^6\right] \right| + 15M \left| R_{n,k}\left[x_1^4 x_2^2\right] \right| + 90N \left| R_{n,k}\left[x_1^2 x_2^2 x_3^2\right] \right| \right\},$$

where

$$L = \sup_{x \in S_n} \left| \sum_{i=1}^{n} \frac{\partial^6 f(x)}{\partial x_i^6} \right|, \qquad M = \sup_{x \in S_n} \left| \sum_{i \neq j} \frac{\partial^6 f(x)}{\partial x_i^4 \partial x_j^2} \right|,$$

$$N = \sup_{x \in S_n} \left| \sum_{i<j<\ell} \frac{\partial^6 f(x)}{\partial x_i^2 \partial x_j^2 \partial x_\ell^2} \right|.$$

It must observe the last term in these formulas is zero when $n = 2$. The general error bound of quadrature in this case is given by:

$$|R_{2,1}[f]| \leqslant \frac{1}{180} \left(\left| \frac{1}{7} - \frac{4}{45}\alpha^2 - \frac{1}{9}\lambda^2\alpha^2 \right| L + \left| 1 - \frac{5}{3}\lambda^2\alpha^2 \right| M \right) = \frac{1}{135\,|15\alpha^2 - 4|} \left(\frac{1}{35} \left| 35\alpha^4 - 51\alpha^2 + 15 \right| L + \left| 5\alpha^2 - 3 \right| M \right),$$

and the corresponding error bounds for the quadrature formulas considered in the previous section are:

$$\begin{aligned}
&\left|R_{2,1}^{[1]}[f]\right| \leqslant \frac{1}{270}\left(\frac{1}{175}L+M\right), && \left(\alpha^2=\frac{2}{5},\ \lambda^2\alpha^2=1,\ \text{see [5]}\right),\\
&\left|R_{2,1}^{[2]}[f]\right| \leqslant \frac{1}{7875}L, && \left(\alpha^2=\frac{3}{5},\ \lambda^2\alpha^2=\frac{3}{5}\right)\\
&\left|R_{2,1}^{[3]}[f]\right| \leqslant \frac{1}{1215}\left(\frac{53}{525}L+2M\right), && \left(\alpha^2=\frac{7}{15},\ \lambda^2\alpha^2=\frac{7}{9}\right),\\
&\left|R_{2,1}^{[4]}[f]\right| \leqslant \frac{1}{2430}\left(\frac{31}{105}L+M\right), && \left(\alpha^2=\frac{2}{3},\ \lambda^2\alpha^2=\frac{5}{9}\right).
\end{aligned}$$

We also consider the error bounds for $n=3$, ($k=1$ and $k=2$).
In the case $k=1$ we have

$$|R_{3,1}[f]| \leqslant \frac{1}{90}\left(\left|\frac{1}{7}-\frac{4}{45}\alpha^2-\frac{1}{9}\lambda^2\alpha^2\right|L+\left|1-\frac{4}{3}\alpha^2-\frac{5}{3}\lambda^2\alpha^2\right|M + 10\left|\frac{1}{3}-\lambda^2\alpha^2\right|N\right),$$

and in the particular cases from the previous section we get

$$\begin{aligned}
&\left|R_{3,1}^{[1]}[f]\right| \leqslant \frac{1}{45}\left(\frac{1}{525}L+\frac{3}{5}M+\frac{10}{3}N\right), && \left(\alpha^2=\frac{2}{5},\ \lambda^2\alpha^2=1\right),\\
&\left|R_{3,1}^{[2]}[f]\right| \leqslant \frac{2}{225}\left(\frac{1}{35}L+M+\frac{10}{3}N\right), && \left(\alpha^2=\frac{3}{5},\ \lambda^2\alpha^2=\frac{3}{5}\right),\\
&\left|R_{3,1}^{[3]}[f]\right| \leqslant \frac{1}{1485}\left(\frac{587}{1575}L+\frac{199}{15}M+40N\right), && \left(\alpha^2=\frac{19}{30},\ \lambda^2\alpha^2=\frac{19}{33}\right),\\
&\left|R_{3,1}^{[4]}[f]\right| \leqslant \frac{1}{405}\left(\frac{31}{315}L+\frac{11}{3}M+10N\right), && \left(\alpha^2=\frac{2}{3},\ \lambda^2\alpha^2=\frac{5}{9}\right).
\end{aligned}$$

In the case $k=2$ we have:

$$|R_{3,2}[f]| \leqslant \frac{1}{90}\left(\left|\frac{1}{7}-\frac{8}{45}\alpha^2-\frac{1}{45}\lambda^2\alpha^2\right|L+\left|1-\frac{4}{3}\alpha^2-\frac{1}{3}\lambda^2\alpha^2\right|M + 10\left|\frac{1}{3}-\frac{1}{5}\lambda^2\alpha^2\right|N\right),$$

and in the particular cases from the previous section we get

$$\left|R_{3,2}^{[1]}[f]\right| \leqslant \frac{1}{135}\left(\frac{1}{35}L+\frac{1}{7}M+2N\right), \qquad \left(\alpha^2=\frac{4}{7},\ \lambda^2\alpha^2=1\right),$$

$$\left|R_{3,2}^{[2]}[f]\right| \leqslant \frac{2}{1125}\left(\frac{1}{7}L+\frac{40}{3}N\right), \qquad \left(\alpha^2=\frac{3}{5},\ \lambda^2\alpha^2=\frac{3}{5}\right),$$

$$\left|R_{3,2}^{[3]}[f]\right| \leqslant \frac{2}{675}\left(\frac{28\sqrt{5}-55}{315}L+\frac{3\sqrt{5}-5}{3}M+4\left(5-\sqrt{5}\right)N\right),$$

$$\left(\alpha^2=\frac{10+\sqrt{5}}{15},\ \lambda^2\alpha^2=\frac{8\sqrt{5}-15}{15}\right),$$

$$\left|R_{3,2}^{[4]}[f]\right| \leqslant \frac{4}{135}\left(\frac{2}{315}L+M\right), \qquad \left(\alpha^2=\frac{2}{3},\ \lambda^2\alpha^2=\frac{1}{3}\right).$$

5. Numerical Examples

To compare the quadrature rules obtained in the previous sections, we have considered the four quadratures when $n=2$ ($k=1$) and $n=3$ ($k=1$ and $k=2$).

The values of parameters α, λ and the coefficients A_0, A_1, A_2 are given in Tables 3–5.

Table 6, Table 7 and Table 8 contain the exact and approximate values obtained by the four quadrature formulas, and also the error bounds, for the following four integrated functions:

(1) $f(x,y)=\frac{1}{(3+x+y)^2}$, with $I_2[f]=\ln\frac{9}{5}=\ln 1.8$,
(2) $f(x,y)=e^{xy}$, with $I_2[f]$ computed by series expansion,
(3) $f(x,y)=\sqrt{2+x+y}$, with $I_2[f]=\frac{32}{15}\left(4-\sqrt{2}\right)$,
(4) $f(x,y)=\frac{1}{\sqrt{3+x+y}}$, with $I_2[f]=\frac{4}{3}\left(5\sqrt{5}-6\sqrt{3}+1\right)$,

respectively

(1) $f(x,y,z)=\frac{1}{(4+x+y+z)^3}$, with $I_3[f]=\frac{1}{2}\ln\frac{189}{125}$,
(2) $f(x,y,z)=e^{xyz}$, with $I_3[f]$ computed by series expansion,
(3) $f(x,y,z)=\sqrt{3+x+y+z}$, with $I_3[f]=\frac{64}{35}\left(9\sqrt{6}-16+\sqrt{2}\right)$,
(4) $f(x,y,z)=\frac{1}{\sqrt{4+x+y+z}}$, with $I_3[f]=\frac{8}{15}\left(49\sqrt{7}-75\sqrt{5}+27\sqrt{3}-1\right)$.

References

[1] Blaga, P. P., *An Approximate Formula for Multiple Integrals*, Facta Universitatis (Niš) (to appear).
[2] Blaga, P. P., *A Class of Multiple Nonproduct Quadrature Formulas*, in "Analysis, Functional Equations, Approximation and Convexity", Carpatica, Cluj–Napoca, 1999, pp. 32–39.
[3] Blaga, P. P., *A New Class of Multiple Nonproduct Quadrature Formulas*, to appear.
[4] Burnside, W., *An Approximate Quadrature Formula*, Messenger of Math. **37** (1908), 166–167.

[5] Das, R. N. and Pradham, G., *A Numerical Quadrature of Function of More Than One Real Variable*, Facta Universitatis (Niš) **11** (1996), 113–118.
[6] Hammer, P. C. and Stroud, A. H., *Numerical Evaluation of Multiple Integrals II*, Math. Tables Aids Comput. **12** (1958), 272–280.
[7] Mustard, D., Lyness, J. N. and Blatt, J. M., *Numerical Quadrature in N Dimensions*, Computer J. **6** (1963–1964), 75–87.
[8] Stroud, A. H., *Some Fifth Degree Integration Formulas for Symmetric Regions*, Math. of Comput. **20** (1966), 90–97.
[9] Stroud, A. H., *Approximate Calculation of Multiple Integrals*, Prentice–Hall, Englewood Cliffs, New Jersey, 1971.

Faculty of Mathematics and Informatics, "Babeş–Bolyai" University, 3400 Cluj–Napoca
E-mail address: blaga@math.ubbcluj.ro

		n=2	n=3	n=4	n=5
k=1	α^2	$\frac{7}{15}$	$\frac{19}{30}$	$\frac{4}{5}$	$\frac{29}{30}$
	α	0.68313	0.79582	0.89443	0.98319
	$\lambda\alpha$	0.88192	0.75879	0.70711	0.67847
	A_1	0.81633	0.88643	1.11111	1.52200
	A_2	0.18367	0.33518	0.44444	0.52438
k=2	α^2		$\frac{10+\sqrt{5}}{15}$	$\frac{5-\sqrt{5}}{5}$	$\frac{20-\sqrt{110}}{15}$
	α		0.90318	0.74350	0.79632
	$\lambda\alpha$		0.43883	0.85065	0.74595
	A_1		0.26716	0.58179	0.58947
	A_2		0.59926	0.12732	0.26316
k=3	α^2				$\frac{30+\sqrt{30}}{45}$
	α				0.88791
	$\lambda\alpha$				0.45395
	A_1				0.19068
	A_2				0.52329

		n=6	n=7	n=8	n=9
k=2	α^2	$\frac{5-2\sqrt{2}}{3}$	$\frac{10-\sqrt{35}}{5}$	$\frac{35-\sqrt{455}}{15}$	
	α	0.85080	0.90376	0.95461	
	$\lambda\alpha$	0.70305	0.67850	0.66230	
	A_1	0.67858	0.85273	1.14174	
	A_2	0.36383	0.44039	0.50049	
k=3	α^2	$\frac{15-\sqrt{21}}{18}$	$\frac{15-\sqrt{30}}{15}$	$\frac{105-\sqrt{1785}}{90}$	$\frac{60-2\sqrt{165}}{45}$
	α	0.76075	0.79678	0.83500	0.87317
	$\lambda\alpha$	0.81874	0.73528	0.69852	0.67680
	A_1	0.35384	0.35288	0.39008	0.46602
	A_2	0.11539	0.22808	0.31736	0.38835
k=4	α^2		$\frac{20+\sqrt{10}}{30}$	$\frac{35-\sqrt{70}}{45}$; $\frac{35+\sqrt{70}}{45}$	$\frac{40-\sqrt{130}}{45}$
	α		0.87868	0.76932; 0.98168	0.79719
	$\lambda\alpha$		0.46432	0.79396; 0.50845	0.72625
	A_1		0.11929	0.20301; 0.07657	0.20122
	A_2		0.47809	0.11185; 0.66501	0.20769
k=5	α^2				$\frac{10-\sqrt{2}}{15}$; $\frac{10+\sqrt{2}}{15}$
	α				0.75656; 0.87232
	$\lambda\alpha$				0.98850; 0.47210
	A_1				0.12403; 0.07018
	A_2				0.02327; 0.44736

TABLE 1. Elements of quadrature formulas $Q_{n,k}^{[3]}$, $n = \overline{2,9}$.

n	k=1	k=2	k=3	k=4	k=5
2	$\left(\frac{7}{15};1\right)$				
3	$\left(\frac{19}{30};1\right)$	$\left(\frac{10+\sqrt{5}}{15};1\right)$			
4	$\left(\frac{4}{5};1\right)$	$\left(\frac{5-\sqrt{5}}{5};1\right)$			
5	$\left(\frac{29}{30};1\right)$	$\left(\frac{20-\sqrt{110}}{15};1\right)$	$\left(\frac{30+\sqrt{30}}{45};1\right)$		
6		$\left(\frac{5-2\sqrt{2}}{3};1\right)$	$\left(\frac{15-\sqrt{21}}{18};1\right)$		
7		$\left(\frac{10-\sqrt{35}}{5};1\right)$	$\left(\frac{15-\sqrt{30}}{15};1\right)$	$\left(\frac{20+\sqrt{10}}{30};1\right)$	
8		$\left(\frac{35-\sqrt{455}}{15};1\right)$	$\left(\frac{105-\sqrt{1785}}{90};1\right)$	$\left(\frac{7}{13};\frac{35-\sqrt{70}}{45}\right)$ $\cup\left(\frac{35+\sqrt{70}}{45};1\right)$	
9			$\left(\frac{60-2\sqrt{165}}{45};1\right)$	$\left(\frac{40-\sqrt{130}}{45};1\right)$	$\left(\frac{4}{7};\frac{10-\sqrt{2}}{15}\right)$ $\cup\left(\frac{10+\sqrt{2}}{15};1\right)$

TABLE 2. Admissible values of parameter α^2, $n=\overline{2,9}$.

	$Q_{2,1}^{[1]}$	$Q_{2,1}^{[2]}$	$Q_{2,1}^{[3]}$	$Q_{2,1}^{[4]}$
α^2	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{7}{15}$	$\frac{2}{3}$
λ^2	$\frac{5}{2}$	1	$\frac{5}{3}$	$\frac{4}{5}$
A_0	$-\frac{8}{9}$	$\frac{64}{81}$	0	$\frac{24}{25}$
A_1	$\frac{10}{9}$	$\frac{40}{81}$	$\frac{40}{49}$	$\frac{2}{5}$
A_2	$\frac{1}{9}$	$\frac{25}{81}$	$\frac{9}{49}$	$\frac{9}{25}$

TABLE 3. Elements of $Q_{2,1}$ quadrature formulas

	$Q_{3,1}^{[1]}$	$Q_{3,1}^{[2]}$	$Q_{3,1}^{[3]}$	$Q_{3,1}^{[4]}$
α^2	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{19}{30}$	$\frac{2}{3}$
λ^2	$\frac{5}{2}$	1	$\frac{10}{11}$	$\frac{5}{6}$
A_0	$-\frac{56}{9}$	$-\frac{32}{81}$	0	$\frac{8}{25}$
A_1	$\frac{20}{9}$	$\frac{80}{81}$	$\frac{320}{361}$	$\frac{4}{5}$
A_2	$\frac{1}{9}$	$\frac{25}{81}$	$\frac{121}{361}$	$\frac{9}{25}$

TABLE 4. Elements of $Q_{3,1}$ quadrature formulas

	$Q_{3,2}^{[1]}$	$Q_{3,2}^{[2]}$	$Q_{3,2}^{[3]}$	$Q_{3,2}^{[4]}$
α^2	$\frac{4}{7}$	$\frac{3}{5}$	$\frac{10+\sqrt{5}}{15}$	$\frac{2}{3}$
λ^2	$\frac{7}{4}$	1	$\sqrt{5}-2$	$\frac{1}{2}$
A_0	$\frac{58}{45}$	$\frac{128}{81}$	0	$\frac{8}{5}$
A_1	$\frac{49}{90}$	$\frac{40}{81}$	$\frac{8\left(21-4\sqrt{5}\right)}{361}$	$\frac{2}{5}$
A_2	$\frac{1}{45}$	$\frac{5}{81}$	$\frac{109+48\sqrt{5}}{361}$	$\frac{1}{5}$

TABLE 5. Elements of $Q_{3,2}$ quadrature formulas

$f(x,y)$	$\frac{1}{(3+x+y)^2}$	e^{xy}	$\sqrt{2+x+y}$	$\frac{1}{\sqrt{3+x+y}}$
$I_2[f]$	$5.87787E{-}01$	$4.22889E{+}00$	$5.51634E{+}00$	$2.38405E{+}00$
$Q_{2,1}^{[1]}[f]$	$6.06351E{-}01$	$4.24137E{+}00$	$5.48365E{+}00$	$2.38611E{+}00$
$Err_{2,1}^{[1]}$	$1.86E{-}02$	$1.25E{-}02$	$3.27E{-}02$	$2.06E{-}03$
$Q_{2,1}^{[2]}[f]$	$5.86676E{-}01$	$4.22897E{+}00$	$5.51752E{+}00$	$2.38394E{+}00$
$Err_{2,1}^{[2]}$	$1.11E{-}03$	$8.05E{-}05$	$1.18E{-}03$	$1.08E{-}04$
$Q_{2,1}^{[3]}[f]$	$5.93612E{-}01$	$4.23365E{+}00$	$5.51298E{+}00$	$2.38477E{+}00$
$Err_{2,1}^{[3]}$	$5.83E{-}03$	$4.76E{-}03$	$3.36E{-}03$	$7.24E{-}04$
$Q_{2,1}^{[4]}[f]$	$5.85275E{-}01$	$4.22800E{+}00$	$5.51830E{+}00$	$2.38376E{+}00$
$Err_{2,1}^{[4]}$	$2.51E{-}03$	$8.92E{-}04$	$1.96E{-}03$	$2.87E{-}04$

TABLE 6. Numerical results ($n = 2$ and $k = 1$)

$f(x,y,z)$	$\frac{1}{(4+x+y+z)^3}$	e^{xyz}	$\sqrt{3+x+y+z}$	$\frac{1}{\sqrt{4+x+y+z}}$
$I_3[f]$	$2.06717E{-}01$	$8.15085E{+}00$	$1.36405E{+}01$	$4.10778E{+}00$
$Q_{3,2}^{[1]}[f]$	$2.70857E{-}01$	$8.48274E{+}00$	$1.35969E{+}01$	$4.11385E{+}00$
$Err_{3,2}^{[1]}$	$6.41E{-}02$	$3.32E{-}01$	$4.35E{-}02$	$6.07E{-}03$
$Q_{3,2}^{[2]}[f]$	$2.12208E{-}01$	$8.27150E{+}00$	$1.36385E{+}01$	$4.10871E{+}00$
$Err_{3,2}^{[2]}$	$5.49E{-}03$	$1.21E{-}01$	$1.96E{-}03$	$9.34E{-}04$
$Q_{3,2}^{[3]}[f]$	$2.10618E{-}01$	$8.25999E{+}00$	$1.36390E{+}01$	$4.10850E{+}00$
$Err_{3,2}^{[3]}$	$3.90E{-}03$	$1.09E{-}01$	$1.41E{-}03$	$7.20E{-}04$
$Q_{3,2}^{[4]}[f]$	$2.09377E{-}01$	$8.25046E{+}00$	$1.36395E{+}01$	$4.10833E{+}00$
$Err_{3,2}^{[4]}$	$2.66E{-}03$	$9.96E{-}02$	$9.78E{-}04$	$5.47E{-}04$

TABLE 7. Numerical results ($n = 3$ and $k = 1$)

$f(x,y,z)$	$\frac{1}{(4+x+y+z)^3}$	e^{xyz}	$\sqrt{3+x+y+z}$	$\frac{1}{\sqrt{4+x+y+z}}$
$I_3[f]$	$2.06717E{-}01$	$8.15085E{+}00$	$1.36405E{+}01$	$4.10778E{+}00$
$Q_{3,2}^{[1]}[f]$	$2.12259E{-}01$	$8.09655E{+}00$	$1.36344E{+}01$	$4.10788E{+}00$
$Err_{3,2}^{[1]}$	$5.54E{-}03$	$5.43E{-}02$	$6.07E{-}03$	$9.88E{-}05$
$Q_{3,2}^{[2]}[f]$	$2.00868E{-}01$	$8.05430E{+}00$	$1.36426E{+}01$	$4.10692E{+}00$
$Err_{3,2}^{[2]}$	$5.85E{-}03$	$9.65E{-}02$	$2.11E{-}03$	$8.60E{-}04$
$Q_{3,2}^{[3]}[f]$	$2.00429E{-}01$	$8.01713E{+}00$	$1.36427E{+}01$	$4.10692E{+}00$
$Err_{3,2}^{[3]}$	$6.29E{-}03$	$1.34E{-}01$	$2.22E{-}03$	$8.64E{-}04$
$Q_{3,2}^{[4]}[f]$	$1.99127E{-}01$	$8.02972E{+}00$	$1.36432E{+}01$	$4.10668E{+}00$
$Err_{3,2}^{[4]}$	$7.59E{-}03$	$1.21E{-}01$	$2.71E{-}03$	$1.10E{-}03$

TABLE 8. Numerical results ($n = 3$ and $k = 2$)

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

COMPLEX VALUE BOUNDARY ELEMENTS METHODS (CVBEM) FOR SOME MIXED BVP

TITUS PETRILA

Abstract. This paper presents a method for setting up a CVBEM for some mixed boundary value problem joined to the Laplace equation, in plane simply conected domains. The considered boundary value problems are met in modelling of different fluid flows as well as in microelectronics, design of electrical machines, magnetohydrodynamics, etc.

The existence and the uniqueness of the classical solutions of the boundary value problems envisaged in this paper have been studied long time ago. Since then such studies have stopped at the analytical stage without emphasizing efficient convergent calculation algorithms, these theoretical results remained rather unusable to applied mathematics . That is why a convergent BEM which is the main purpose of our work, seems to be an useful tool for filling in the above mentioned gap. In the sequel we will foccuss on the Robin problem, an important modified Volterra problem following as an particular case. Some extension as well as some effective numerical approches of certain particular problems will be considered in the future.

1. Let us consider the Robin(mixed, Dirichlet-Neumann) problem for the Laplace operator in a disk Ω (centred at O, of radius R) with the boundary C. With the form of complex functions, the problem leads to the determination of a function $f = u + iv$ holomorphic in Ω , continuous with its derivative in $\Omega \cup C$ such that its real part u satisfies the boundary condition

$$\alpha u + \beta \frac{du}{dn_i}|_C = l, \tag{1}$$

where α , $\beta \in R$, while l is supposed to be a given continuous function on C and $\frac{du}{dn_i}$ is the derivative along the unit inward normal direction $\vec{n}$. Of course for $\beta = 0$ or $\alpha = 0$ we get respectively the Drichlet and Neumann problems. In the last case the compatibility condition requires that $\int\limits_{|z|=R} \frac{l}{\beta} df = 0$. We remark that

in the case of a disk, the condition (1) can be replaced by

$$\alpha Ref - \frac{\beta}{R} Re(z\frac{df}{dz})|_{|z|=R} = l,$$

with a Dirichlet condition $ReF(z)|_{|z|=R} = l$ for the holomorphic function

$$F(z) = \alpha f(z) - \frac{\beta}{R} z\frac{df}{dz}$$

Using now a technique already built up by us [3, 4] $f(z)$ is looked under the form of $F(z) = \frac{1}{2\pi i}\int\limits_C \frac{F(t)}{t-z}dt, \text{for} z \in \Omega$. But this Cauchy′s integral for our solution is, in fact, the integral representation required by a BEM which, together with an appropiate boundary integral equation provide the main tools for setting up a BEM.

In our case we will manage to avoid the construction and the solution of any boundary integral equation which, by elliminating of some involved approximations of both contour and itegrals simplifies essentialy our BEM.

Namely, considering a set of nodal points, z_0, z_1, ..., z_n ($z_0 = z_n$) on C, disposed contour clockwise and separating the curve C into boundary elements $C_j = (z_{j-1}z_j)$, $j \in \{1, \ldots, n\}$, the envisaged approximation $\tilde{F}(t)$ for the unknow function $F(t)$ is

$$\tilde{F}(t) = \sum_{j=1}^{n} F_j L_j(t) \text{where} F_j = F(z_j),$$

while$L_j(t)$ are the interpolating Lagrange functions, constructed on each arc respectively, i.e.

$$L_j(t) = \begin{cases} \frac{t-z_{j-1}}{z_j - z_{j-1}} & \text{for } t \in C_j \ . \\ \frac{t-z_{j+1}}{z_j - z_{j+1}} & \text{for } t \in C_{j+1} \\ 0 & \text{otherwise.} \end{cases}$$

Accordingly, the approximation $F^*(z)$ of our unknown function $F(z)$ in an interior point $z \in \Omega$ will be defined by

$$F^*(z) = \sum_{i=1}^{n} F_j \tilde{L}_j(z)$$

where,

$$\tilde{L}_j(z) = \frac{1}{2\pi i}\int_C^{'} \frac{L_j(t)}{t-z}dt = \frac{1}{2\pi i}(\frac{z-z_{j-1}}{z_j - z_{j-1}} \ln\frac{z-z_j}{z-z_{j-1}} + \frac{z-z_j}{z_j - z_{j+1}} \ln\frac{z-z_{j+1}}{z_j - z_{j+1}})$$

and where one choses the main (principal) determination of the complex logarithm. The continuity of the approximation $F^*(z)$ allows us to write that

$$U_k + iV_k = F_k \approx F^*(z_k) = \sum_{j=1}^{n} F_j \tilde{L}_j(z_k), k \in \{1, \ldots, n\}$$

i.e. we are led to the following real linear system of $2n$ equations and $2n$ unknowns

$$U_k = \sum_{i=1}^{n} M_{kj} U_j - \sum_{j=1}^{n} N_{kj} V_j V_k = \sum_{i=1}^{n} M_{kj} V_j - \sum_{j=1}^{n} N_{kj} U_j$$

By solving this system within the data of the Dirichlet boundary problem, we get the looked approximation $\tilde{F}(t)$ of the function $F(t)$ and, implicitely, via Cauchy′s formula, the solution of the proposed boundary problem in all the points of the domain D. Concerning the coefficients L_{kj}, for $k \neq j$, they could be directly calculated from the expresion of $\tilde{L}_j(z)$ using the equality

$$\lim_{z \to z_p} (z - z_p) \ln \frac{z_j - z_{j+1}}{z_j - z_{j-1}} = 0$$

in the case $k = j - 1$ or $k = j + 1$. For $k = j$ we get

$$L_{jj} = \frac{1}{2\pi i} \ln \frac{z_j - z_{j+1}}{z_j - z_{j-1}},$$

the same principal determination of the logarithm being considered.
A division $d := (z_0, z_1, \ldots, z_n), z_0 = z_n$ of the curve C will be called "acceptable" if for each $t \in C_j, j \in \{1, 2, \ldots, n\}$ the following condition is fulfilled

$$max\{|t - z_j|, |t - z_{j-1}|\} < |z_j - z_{j-1}|.$$

By introducing the concept of "acceptable" division

$$d := (z_0, z_1, \ldots, z_n)$$

of the norm $||d||$ and using a convergence theorem given by us [3, 4], we state that

$$\lim_{n \to \infty} F^*(z) = F(z)$$

for an acceptable division $||d|| \to 0$. At the same time the solution $f^*(z)$ of the following differential equation

$$\alpha f^*(z) - \frac{\beta}{R} z \frac{df^*(z)}{dz} = F^*$$

i.e.

$$f^*(z) = K - \frac{R}{\beta} \int \frac{F^*(z)}{z} e^{-\frac{R\alpha}{\beta} - 4z} dz] e^{\frac{R\alpha}{\beta} \ln z}$$

where K is an determined constant, which will be fixed by the conditions associated to the problem, and $\ln z$ means the principal (main) determination of the complex logarithm, will converge to the exact solution $f(z)$, due to the continuity of all the functions involved. Summarizing the CVBEM just built up is convergent, i.e.

$$\lim_{n\to\infty} f^*(z) = f(z),$$

for any acceptable division $d = (z_0, \ldots, z_n)$ of norm $||d||$, which $||d|| \to 0$. Obviously the above procedure can be also extended to the case when α and β are real continuous functions. But despite that, the solution of the attached differential equation will be not the same as before, the convergence of the corresponding CVBEM is still valid. We remark that the initial Robin problem can be transformed into a Dirichlet problem, using that $\frac{du}{dn_i}|_C = \frac{dv}{ds}|_C$ obvious extension of Cauchy-Riemann relations (of course the validity of these relations on the boundary is connected with the analycity of $f(z)$ on C, which ensures that the image $F(C)$ is also an analytical curve). Hence

$$\frac{dv}{ds}|_C = -\frac{\alpha}{\beta}u - \frac{l}{\beta},$$

i.e.

$$v|_C = -\frac{\alpha}{\beta}2\pi R u(0) - \int_C \frac{l}{\beta}ds + k,$$

where k is an arbitrary constant while $u(0)$ is the value of the function u at $z = 0$ (the Gauss theorem). The equivalence of the mixed (Robin) problem with a Dirichlet one can be established even in the case when Ω is an arbitrary, plane, simply connected (bounded) domain, whose boundary C is an analytical curve. Keeping the same requirements on the unknown function $f(z)$, which also ensures even the validity of Cauchy-Riemann relations on C, the Robin condition, with α and β real continuous functions, can be replaced by

$$\alpha u + \beta\frac{dv}{ds}|_C = l$$

At the same time, supposing that $\beta \neq 0$ and using the harmonicity of u, the initial Robin condition leads

$$\int_C \frac{\alpha}{\beta}uds = \int_C \frac{l}{\beta}ds, \quad \int_C \frac{du}{dn_i}ds = 0,$$

for any harmonical function u. Immediately we can write that $v|_C = K$, where K is an undetermined constant, i.e. the mentioned equivalence has been proved.

2. Let now $\Omega \subset R^2$ be a simply connected domain whose analytical boundary C is divided into C_u and C_v so that $C_u \cap C_v = \emptyset$ and $C_u \cup C_v = C$. We intend to determine the function $u \in C^2(\Omega) \cap C'(\Omega \cup C)$ which satisfies the conditions:

$$\begin{aligned} \triangle u &= 0 \quad \text{in } \Omega \\ u|_{C_u} &= l_1 \quad \text{on } C_u \\ \frac{du}{dn_i}|_{C_v} &= l_2 \quad \text{on } C_v, \end{aligned}$$

l_1 and l_2 being two given real continuous functions, and $\frac{d}{dn_i}$ the derivative in the direction of the inward normal n_i. Of course this problem, which is considered to be a "modififed Volterra" problem [2], could be rewritten under the previous (Robin) form if α and β are a couple of real continuous functions so that $\alpha\beta|_C = 0, \alpha|_{C_u} \neq 0$ and $\beta|_{C_v} \neq 0$, l being now

$$l = \alpha\frac{l_1}{\alpha} + \beta\frac{l_2}{\beta}.$$

It is obvious that this problem is also equivalent with a clssical Volterra problem. Precisely the condition $\frac{du}{dn_i}|_C = l_2$ can be replaced by $v = \int l_2 ds + k$, k being an integration constant whose determination is accomplished within the frame of the proposed problem. But this modified Volterra problem leads, using the same technique as above, to a Dirichlet problem whose data on boundary can have a finite number of singularities of first type. But the existence and the uniqueness of the solution of this Dirichlet problem has been proved too (Fatou), so that, the whole procedure for appropiate CVBEM envisaged before, can be developped again.

References

[1] Caius Iacob, *Introduction mathèmatique à la mécanique des fluides*, Editions Gauthier-Villars et L'Academie Roumaine, 1959

[2] Dorel Homentcovschi, *On the mixed boundary-value problems for harmonic functions in plane domains*, Journal of Applied Mathematics and Physics (ZAMP), **31** (1980), p. 351–366

[3] Titus Petrila, *On certain mathematical problems connected with the use of the complex variable boundary element method to the problems of plane hydrodynamics. Gauss variant of the procedure*, The mathematical heritage of C. F. Gauss, World Scientific,Publishing Company, Singapore, 1991, p. 585–604

[4] Titus Petrila, *An improved CVBEM for plane hydrodynamics*, Revue d'analyse numerique et de la theorie de l'approximation, vol. XVI, fasc. **2** (1987), p. 149–157

"Babes-Bolyai" University, Faculty of Mathematics and Computer Science, RO-3400 Cluj-Napoca, Romania

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

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

BLIND SIGNATURE AND BLIND MULTISIGNATURE SCHEMES USING ELLIPTIC CURVES

CONSTANTIN POPESCU

Abstract. Blind signature schemes and blind multisignature schemes are useful in protocols that guarantee the anonymity of the participants. In this paper we propose an elliptic curve blind signature scheme and an elliptic curve blind multisignature scheme. The proposed schemes are described in the group of points on an elliptic curve because it offer equivalent security as the other groups but with smaller key size and faster computation times.

1. Introduction

The concept of blind signature schemes was introduced by Chaum in 1982 [2]. A blind signature scheme allows to realize secure electronic payment systems protecting customer's privacy [1], [3], [5]. Recent anonymous prepaid electronic payment systems, based on the blind signature technique, emulate physical cash. In these systems, the users withdraw electronic coins which consist of numbers, generated by users, and blindly signed by an electronic money issuer. Each signature represents a given amount. These coins are then spent in shops which can authenticate them by using the public signature key of the bank. The users retain anonymity in any transaction since the coins they use have been blindly signed.

In a blind multisignature scheme [8], [10] we have one owner Alice, who wants to obtain a digital signature from several signers, so that each signer doesn't know a relationship between the blinded and unblinded message and signature parameters. This means they cannot recognize the signature later, even if they all collude. A blind multisignature scheme can be used as a building block in cryptographic applications, e.g. in electronic voting schemes [4].

In this paper we propose an elliptic curve blind signature scheme and an elliptic curve blind multisignature scheme. The schemes proposed are described in the group of points on an elliptic curve defined over a finite field. Elliptic curve groups are advantageous because they offer equivalent security as the other groups but with smaller key size and faster computation times.

1991 *Mathematics Subject Classification.* 94A60.

1991 *CR Categories and Descriptors.* D.4.6 [**Operating Systems**]: Security and Protection – *Authentication Cryptographic controls.*

2. Elliptic Curves over Finite Fields

Many researchers have examined elliptic curve cryptosystems, which were firstly proposed by Miller [15] and Koblitz [12]. The elliptic curve cryptosystems which are based on the elliptic curve logarithm over a finite field have some advantages over other systems: the key size can be much smaller over the other schemes since only exponential-time attacks have been known so far if the curve is carefully chosen [13], and the elliptic curve discrete logarithms might be still intractable even if factoring and the multiplicative group discrete logarithm are broken.

Elliptic Curves over $GF(2^n)$: A non-supersingular elliptic curve E over $GF(2^n)$ can be written into the following standard form

$$E: \quad y^2 + xy = x^3 + ax^2 + b, \quad b \neq 0,\ a, b \in GF(2^n).$$

The points $P = (x, y)$, $x, y \in GF(2^n)$ that satisfy this equation, together with a "point at infinity" denoted O form an abelian group $(E, +, O)$ whose identity element is O.

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two different points on E and both P and Q are not equal to the infinity point. Addition law for E non-supersingular is as follow: For $2P = P + P = (x_3, y_3)$, if $x_1 \neq 0$

$$\begin{aligned} x_3 &= \delta^2 + \delta + a \\ y_3 &= (x_1 + x_3)\delta + x_3 + y_1, \quad \textit{where } \delta = x_1 + y_1/x_1. \end{aligned}$$

If $x_1 = 0$, $2P = O$. For $P + Q = (x_3, y_3)$, if $x_1 = x_2$, then $P + Q = O$. Otherwise,

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1, \quad \textit{where } \lambda = (y_1 + y_2)/(x_1 + x_2). \end{aligned}$$

Elliptic Curves over $GF(p^n)$: A non-supersingular elliptic curve E over $GF(p^n)$, $p > 2$ can be written into the following standard form

$$E: \quad y^2 = x^3 + ax + b, \quad 4a^3 + 27b^2 \neq 0,\ a, b \in GF(p^n).$$

For the addition law, for the elliptic curve E over $GF(p^n)$, see more details in [15].

3. Elliptic Curve Blind Signature Scheme

In this section we describe the elliptic curve blind version of the Harn's signature scheme [7]. We will use the same setup as suggested in IEEE P1363 standard form [11].

3.1. Key Generation.

Firstly, we choose elliptic curve domain parameters:

(1) Choose p a prime and n an integer. Let $f(x)$ be an irreducible polynomial over $GF(p)$ of degree n, generating finite field $GF(p^n)$ and assume that α is a root of $f(x)$ in $GF(p^n)$.

(2) Two field elements $a, b \in GF(p^n)$, which define the equation of the elliptic curve E over $GF(p^n)$ (i.e., $y^2 = x^3 + ax + b$ in the case $p > 3$), where $4a^3 + 27b^2 \neq 0$.
(3) Two field elements x_p and y_p in $GF(p^n)$, which define a finite point $P = (x_p, y_p)$ of prime order in $E(GF(p^n))$ ($P \neq O$, where O denotes the point at infinity).
(4) The order q of the point P.
(5) The converting function $c(x) : GF(p^n) \to Z_{p^n}$ which is given by

$$c(x) = \sum_{i=0}^{n-1} c_k p^i \in Z_{p^n}, \quad x = \sum_{i=0}^{n-1} c_k \alpha^i \in GF(p^n),\ 0 \leq c_i < p.$$

The operation of the key generation is as follows:

(1) Select a private key d, a random integer, from the interval $[1, q-1]$.
(2) Compute the public key Q, which is a point on E, such that $Q = dP$.

3.2. **Blind Signature Protocol.** The following protocol is a blind version of the Harn's elliptic curve signature scheme.

(1) Alice generates a one-time key pair $(\overline{k}, \overline{R})$ in the following way: randomly chooses $\overline{k} \in [1, q-1]$ and compute $\overline{R} = \overline{k}P = (x_{\overline{k}}, y_{\overline{k}})$. She computes $\overline{r}$ such that

$$\overline{r} = c(x_{\overline{k}}) = \sum_{i=0}^{n-1} c_{i\overline{k}} p^i, \quad where\ x_{\overline{k}} = \sum_{i=0}^{n-1} c_{i\overline{k}} \alpha^i,\ 0 \leq c_{i\overline{k}} < p.$$

and sends $\overline{r}$ and $\overline{R}$ to Bob.
(2) Bob chooses blind factors $a, b \in [1, q-1]$, computes the point R on E such that $R = a\overline{R} + bP = (x_k, y_k)$ and computes $r = c(x_k)$. He also computes $\overline{m} = (H(m) + r)a^{-1} - \overline{r}$, where $H(\cdot)$ is a hash function, and sends $\overline{m}$ to Alice.
(3) Alice computes $\overline{s} = d(\overline{m} + \overline{r}) + \overline{k}\ (mod\ q)$ and sends $\overline{s}$ to Bob.
(4) Bob computes $s = a\overline{s} + b$.

The pair (r, s) is an elliptic curve signature of the message m.

Theorem 3.1. *The pair (r, s) is a Harn elliptic curve signature of the message m and the above protocol is an elliptic curve blind signature scheme.*

Proof: The validity of the signature (r, s) of the message m follows from the next steps:

(1) Compute a point on E such that $sP - (H(m) + r)Q = (x_e, y_e)$.
(2) Use the converting function to compute the integer $c(x_e)$ and check if $r = c(x_e)(mod\ q)$. If this equation is true, then (r, s) is accepted as a valid signature of the message m. It is easy to verify that $sP - (H(m) + r)Q = R$.

To prove that the above protocol is blind we show that for every possible signer's view there exists a unique pair (a, b) of blind factors, with $a, b \in [1, q-1]$. Given any view consisting of $\overline{R}, \overline{k}, \overline{r}, \overline{m}, \overline{s}$ and any valid elliptic curve signature (r, s) of a message m, we consider

$$\begin{aligned} a &= (H(m)+r)(\overline{m}+\overline{r})^{-1}(mod\ q) \\ b &= s - a\overline{s}(mod\ q). \end{aligned}$$

We have to show that $R = a\overline{R} + bP$. We have $a\overline{R} + bP = a\overline{k}P + sP - a\overline{s}P = a\overline{k}P + sP - aP(d\overline{m} + d\overline{r} + \overline{k}) = sP - adP((H(m)+r)a^{-1} - \overline{r}) - ad\overline{r}P = sP - dH(m)P - drP = sP - (H(m)+r)Q = R$. □

4. Elliptic Curve Blind Multisignature Scheme

In this section we describe the elliptic curve blind version of the Harn's multisignature scheme [8].

4.1. **Key Generation.** The elliptic curve domain parameters are the same as in Section 3. We assume there are t signers $U_i,\ i = 1, ..., t$. The operation of the key generation is as follows:

(1) Each signer U_i randomly selects his private key d_i, an integer, from the interval $[1, q-1]$.
(2) The public key of the signer U_i is the point
$$Q_i = d_iP = (x_{d_i}, y_{d_i}),\ i = 1, ..., t.$$
(3) The public key for all signers is
$$Q = Q_1 + ... + Q_t = dP = (x_d, y_d),$$
where $d = d_1 + ... + d_t(mod\ q)$.

4.2. **Blind Multisignature Protocol.** The following protocol is a blind version of the Harn's elliptic curve multisignature scheme.

(1) The user U_i generates a one-time key pair $(\overline{k}_i, \overline{R}_i)$ in the following way: randomly chooses $\overline{k}_i \in [1, q-1]$ and computes $\overline{R}_i = \overline{k}_iP = (x_{\overline{k}_i}, y_{\overline{k}_i})$. The user U_i computes $\overline{r}_i,\ i = 1, ..., t$, such that $\overline{r}_i = c(x_{\overline{k}_i})$ and sends $\overline{r}_i$ and $\overline{R}_i$ to the clerk.
(2) The clerk chooses the blind factors $a, b \in [1, q-1]$, computes the point R on E such that $R = a\overline{R} + bQ = (x_k, y_k)$, where $\overline{R} = \overline{R}_1 + ... + \overline{R}_t$ and $Q = Q_1 + ... + Q_t$. The clerk computes $r = c(x_k)(mod\ q)$ and $\overline{m} = (H(m) + r + b)a^{-1} - \overline{r}$, where $H(\cdot)$ is a hash function, and sends $\overline{m}$ and $\overline{r}$ to each signer U_i.
(3) The user U_i computes the signature $\overline{s}_i = d_i(\overline{m} + \overline{r}) + \overline{k}_i\ (mod\ q),\ i = 1, ..., t$ and sends $\overline{s}_i$ to the clerk.

(4) The clerk computes $\overline{s}_iP-(\overline{m}+\overline{r})Q_i=(x_{e_i},y_{e_i})$ and check $\overline{r}_i=c(x_{e_i})$ $(mod\ q)$, $i=1,...,t$. The elliptic curve blind multisignature of the message m can be generated as (r,s), where $\overline{s}=\overline{s}_1+...+\overline{s}_t(mod\ q)$ and $s=\overline{s}a(mod\ q)$.

The pair (r,s) is a elliptic curve multisignature of the message m.

Theorem 4.1. *The pair (r,s) is a Harn elliptic curve multisignature of the message m and the above protocol is an elliptic curve blind multisignature scheme.*

Proof: The validity of the elliptic curve multisignature (r,s) of the message m follows from the next steps:

(1) Compute a point on E such that $sP-(H(m)+r)Q=(x_e,y_e)$.
(2) Use the converting function to compute the integer $c(x_e)$ and check if $r=c(x_e)(mod\ q)$. If this equality is true, then (r,s) is accepted as a valid elliptic curve multisignature of the message m. It is easy to verify that $sP-(H(m)+r)Q=R$.

To prove that the above protocol is blind we show that for every possible signer's view there exists a unique pair (a,b) of blind factors, with $a,b\in[1,q-1]$. Given any view consisting of $\overline{R}_i,\overline{k}_i,\overline{r}_i,\overline{s}_i,\overline{m},\overline{r}$ and any valid elliptic curve multisignature (r,s) of a message m, we consider

$$\begin{aligned} a &= s\overline{s}^{-1}(mod\ q)\\ b &= (\overline{m}+\overline{r})a-H(m)-r(mod\ q).\end{aligned}$$

We have to show that $R=a\overline{R}+bQ$. We have $a\overline{R}+bQ=a(\overline{R}_1+...+\overline{R}_t)+((\overline{m}+\overline{r})a-(H(m)+r))Q=a(\sum_{i=1}^{t}\overline{s}_iP-\sum_{i=1}^{t}(\overline{m}+\overline{r})Q_i))+a(\overline{m}+\overline{r})Q-(H(m)+r)Q=a\overline{s}P-(H(m)+r)Q=sP-(H(m)+r)Q=R$. □

5. Security Considerations

Our elliptic curve blind signature and multisignature schemes are as secure as the Harn schemes [7], [8]. But, our schemes is more efficient than Harn schemes because the group of points on an elliptic curve offer smaller key size and faster computation times. The signature schemes in [9] can provide similar elliptic curve blind signature schemes and elliptic curve blind multisignature schemes. In order to avoid the Pollard-rho [19] and Pohling-Hellman [18] algorithms for the elliptic curve discrete logarithm problem, it is necessary that the number of F_q-rational points on E, denoted $\#E(F_q)$, be divisible by a sufficiently large prime n. It is commonly recommended that $n>2^{160}$. To avoid the reduction algorithms of Menezes, Okamoto and Vanstone [14] and Frey and Ruck [6], the curve should be non-supersingular. To avoid the attack of Semaev [20] on F_q-anomalous curves, the curve should not be F_q-anomalous (i.e., $\#E(F_q)\neq q$).

A prudent way to guard against these attacks, and similar attacks against special classes of curves that may be discovered in the future, is to select the elliptic curve E at random subject to the condition that $\#E(F_q)$ is divisible by a large

prime - the probability that a random curve succumbs to these special purpose attacks is negligible. A curve can be selected verifiable at random by choosing the coefficients of the defining elliptic curve equation as the outputs of a one-way function such as SHA-1 according to some pre-specified procedure.

6. Conclusion

In this paper we proposed an elliptic curve blind signature scheme and an elliptic curve blind multisignature scheme. The proposed schemes are described in the setting of the group of points on an elliptic curve because it offer equivalent security as the other groups but with smaller key size and faster computation times. Our elliptic curve blind signature and multisignature schemes are as secure as the Harn schemes. These schemes are practical, requiring just a few exponentiations or integer multiplications over a group.

References

[1] S. Brands, Electronic Cash Systems Based on the Representation Problem in Groups of Prime Order, Advances in Cryptology-CRYPTO'93, Lecture Notes in Computer Sciences, Springer-Verlag, 1993.
[2] D. Chaum, Blind signature systems, Advances in Cryptology-CRYPTO'83, Plenum Press, 1984, pp. 153.
[3] D. Chaum, Privacy Protected Payment, SMART CARD 2000, Elsevier Science Publishers B.V., 1989, pp. 69-93.
[4] L. Chen, M. Burmester, A practical voting scheme with allows voters to abstain, Proceedings of Chinacrypt'94, 1994, pp. 100-107.
[5] N. Ferguson, Single Term Off-line Coins, Advances in Cryptology-EUROCRYPT'93, Lecture Notes in Computer Sciences, 765, Springer-Verlag, 1993, pp. 318-328.
[6] G. Frey, H. Ruck, A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves, Mathematics of Computation, 67, 1998, pp. 353-356.
[7] L. Harn, A New Digital Signature Based on the Discrete Logarithm, Electronics Letters, Vol. 30, No.5, 1994, pp. 193-195.
[8] L. Harn, Group-oriented (t, n) Threshold Signature and Multisignature, IEE Proceedings Computers and Digital Techniques, Vol. 141, No.5, 1994, pp. 307-313.
[9] L. Harn, On the design of generalized ElGamal type digital signature schemes based on the discrete logarithm, Electronics Letters, 1994.
[10] P. Horster, M. Michels, H. Petersen, Blind multisignature schemes and their relevance to electronic voting, Proc. 11th Annual Computer Security Applications Conference, New Orleans, IEEE Press, 1995, pp. 149 - 155.
[11] IEEE P1363, Standard Specifications for Public Key Cryptography, The Institute of Electrical and Electronics Engineers, 1998.
[12] N. Koblitz, Elliptic curve cryptosystems, Mathematics of Computation, 48, 1987, pp. 203-209.
[13] N. Koblitz, CM-Curves with Good Cryptographic Properties, Proceedings of Crypto'91, 1992.
[14] A. Menezes, T. Okamoto, S. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, IEEE Transactions on Information Theory, 39, 1993, pp. 1639-1646.
[15] A. Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, 1993.

[16] V. Miller, Uses of elliptic curves in cryptography, Advances in Cryptology, Proceedings of Crypto'85, Lecture Notes in Computer Sciences, 218, Springer-Verlag, 1986, pp. 417-426.
[17] T. Okamoto, K. Ohta, Universal Electronic Cash, Advances in Cryptology-CRYPTO'91, Lecture Notes in Computer Sciences, 576, Springer-Verlag, 1991, pp. 324-337.
[18] S. Pohling, M. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, IEEE Transactions on Information Theory, 24, 1978, pp. 106-110.
[19] J. Pollard, Monte Carlo methods for index computation *mod* p, Mathematics of Computation, 32, 1978, pp. 918-924.
[20] I. Semaev, Evaluation of discrete logarithms in a group of p -torsion points of an elliptic curve in characteristic p, Mathematics of Computation, 67, 1998, pp. 353-356.

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

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

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

A COMPONENT BASED APPROACH FOR SCIENTIFIC VISUALIZATION OF EXPERIMENTAL DATA

DUMITRU RADOIU AND ADRIAN ROMAN

Abstract. The paper addresses the issue of component-based scientific visualization systems as a solution to most of the standard/commercial visualization system problems. The visualization process is requested to meet the process validation criteria. The visualization system is requested to observe a reference model. The benefits are discussed on a detailed component based visualization system.

Keywords: scientific visualization, software components, visualization reference model, and visualization process validation

1. Introduction

The commercial visualization systems, the so-called "turn key systems, are easy to use but they have some disadvantages [1]:

- they are expensive;
- they run on pretentious platforms;
- they are "rigid, meaning that one can not use more than one instance of a visualization module. Several instances of the same module allow the simultaneous execution of several sets of input data;
- their modules can not be used to build other systems;
- the implemented algorithms are not always the algorithms desired by the user;
- the functionalities of a turn key system are "fixed, they can not be changed by the user.

This article proposes a new approach of the visualization field. Three goals are to be reached:

a.: Description of a **reference model** for the visualization process;

b.: Formulation of some criteria for the **validation of the scientific visualization process** ;

c.: Description of a **software component** model used to build personalized visualization systems. Such a model allows the rapid construction of visualization systems using modules that implement the desired algorithms.

The **reference model** represents an abstract view of the visualization process. It uses the concept of **level**. A level is seen as a distinct stage of the process that accepts data and services of a certain format as input. The result is a new set of data and services offered as input to the next level.

The reference model also serves to standardize the terminology, as well as to compare the visualization systems and to identify the constraints imposed by the process.

The **validation of the scientific visualization process** determines whether the process results in a *scientific* visualization.

The model of the **component-based visualization systems** is supposed to suggest some formalization capable to offer proper answers to the following questions: How does a software component behave? How are the interfaces between two components described? Which are the conditions that allow the composition of two or more components?

The idea of the component-based visualization systems is to use independent components to construct personalized visualization systems in order to obtain optimum and flexible systems that include a large variety of functionalities. A very important advantage introduced by this model is that the visualization systems can include user-made components, implementing the desired algorithms. The construction of components implies a large agreement upon the different data models, upon a formal model for time-sometimes the time being a critical variable–, upon a user model, etc.

2. Towards a reference model

In order to describe the reference model for the visualization systems a "decomposition" of the visualization process is performed. The identification of the "levels" used for the data processing allows a better understanding of the conditions that must be imposed on the visualization function. The proposed model [2] contains three levels: **modeling level, logical visualization level** and **physical visualization level.**

The **modeling level** only "extracts" from the data set the information of interest, allowing a global processing of the data. This level performs operations such as extraction of the data geometry, sub-sampling or re-sampling, extraction of the data set characteristics, etc.

The data objects are passed to the next level, the **logical visualization level,** where they are "factorized" into primitive data and mapped to glyphs and graphic primitives. A set of functionalities is implemented at this level, e.g. the choice of the visualization primitives, of the optical characteristics of objects, the computation of the optical properties of scenes, the light settings etc. Interaction elements are also present, performing rotations, translations, zooming.

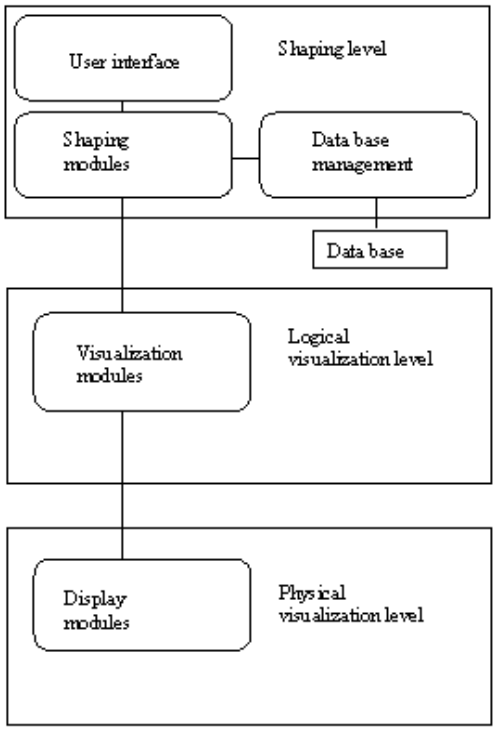


Fig. 1 The architecture of a visualization system for experimental data.

The **physical visualization level** includes the choice of visualization medium and classical tools are available to perform the hidden surface removal, shading, lightning, etc.

Figure 1 proposes the architecture of a visualization system for experimental data following the above-described model.

3. Validation of the scientific visualization process

Scientific visualization is a computational process that transforms scientific data in visual objects [3]. Not all visualizations are scientific ones. A scientific visualization guaranties a certain degree of accuracy. In order to state conditions to be fulfilled by a scientific visualization, some mathematical structures over data sets are introduced.

3.1. **Basic concepts and definitions.**

The idea of using the mathematical structures defined over data sets to find conditions imposed on the visualization function has been promoted by many authors [4], [5], [6], [7], [8].

Scientific data can be obtained in many different ways, e.g. by running a simulation or through a DAQ process. Usually, scientific data objects are finite representations of complex mathematical objects. We note by $\mathbf{O}$ the set of such objects, $o \in \mathbf{O}$. During the visualization process, initial data objects, o, are processed through different transformation functions $Mat(o) = o'$, into a new set $o' \in \mathbf{O}'$. Objects o' are then mapped $Map(o') = g$ into a set of ideal geometrical objects $g \in \mathbf{G}$, through a set of graphical primitives. Objects g are usually n-dimensional (nD), animated (t) and interactive. A group of g objects is usually called the **logical visualization** of a scene.

Ideal geometrical objects g, nD, animated (t) and interactive are usually represented $Rep(g) = g', g' \in \mathbf{G}'$, on real $2D$ screens. A group of g' objects is usually called a **physical visualization** of a scene. Functions $Rep(g) = g'$ implement classical graphical operations such as composition of the scene, volume generation, isosurface generation, simulation of transparency, reflectivity and lighting conditions, $nD \rightarrow 2D$ projection, clipping, hidden surface removal, shading, animation (t), setting user interactivity (zoom, rotate, translate, pan, etc), etc.

By **interactivity** we understand the attributes of visual objects (logical and/or physical) whose setting allows $nD \rightarrow 2D$ projection (zoom, rotate, translate, pan, etc), animation control (t), control of the objects composing the scene and control of the scene as a composite object.

The scientific visualization process is described by the $Vis(o) = g'$, $Vis(o) = Rep(Map(Mat(o))) = g'$ function.

The above concepts and notations are synthesized in the table 1 (see Appendix).

3.2. **Fundamental conditions of scientific visualization.** There are many requirements concerning a certain process of scientific visualization. Here are the three fundamental ones.

The first one is the **distinctiveness condition.** This condition (although very weak) enables users to distinguish between different data objects based on their display. The condition is necessary as one can imagine many visualization functions that generate images with no use, which reveal none of the data objects characteristics/attributes.

The second condition is the so called the **expressiveness condition.** It assures that the attributes of the visual object represent the attributes of the input data set.

The third one is the **precision condition.** This condition insures that the order among data objects is preserved among visual objects.

The distinctiveness condition. em Different input data (different mathematical objects) have to be mapped into different visual objects.

This can be stated as: $o_1 \neq o_2 \Leftrightarrow Vis(o_1) \neq Vis(o_2) \Leftrightarrow Rep(Map(Mat(o_1))) \neq Rep(Map(Mat(o_2))) \Leftrightarrow g'_1 \neq g'_2$ for any $o_1, o_2 \in O, g'_1, g'_2 \in G'$

The interpretation of this condition is that $Vis()$, $Mat()$, $Map()$ and $Rep()$ functions are injective.

The expressiveness condition. *The visual objects express all and only the characteristics of input data.*

It results that the visualization function should be bijective/one to one.

The two conditions are necessary but not sufficient. Another condition is needed to establish an order relation, seen as a precision relation.

The precision condition. *For any objects o_n, $o_m \in O$ such that o_n is "more precise" than o_m we have $Vis(o_n)$ "more precise" than $Vis(o_m)$,with $Vis(o_n)$, $Vis(o_m) \in G'$.*

The precision relation adds something new. If the visualization function is well defined and the input data objects are strictly ordered, the visual objects can be ordered by precision.

The first two conditions introduce criteria of validation and control of the visualization process. The visualization function $Vis()$ fulfilling these criteria results in a scientific visualization. The third condition allows further developments by defining mathematical operations on the given ordering.

The mathematical structures and the other notions presented above are later used to support the main ideas of this paper.

4. Component based visualization systems

The alternative to the acquisition of commercial visualization systems and their parameterization to fit to the user's problem is the use of components. These are pre-made complex functional entities, that can be (re)used to construct the desired architecture of a visualization system. A component performs a specific visualization task, which can be isolated inside the system.

A component can be a class or a collection of classes. Unlike classes, a component can be implemented using a technology that is completely different from the object oriented one, e.g. an assembling language. Classes are some how similar to components, but the object-oriented technology has not imposed or hasn't been able to impose components. One of the reasons is that the definition of the objects is a purely technical one. An object represents the encapsulation of a state, of a behavior, polymorphism and inheritance. The definition does not include the notions of independence and forward composition. Components as well as objects are re-usable. Components' description has to be more general in order to be capable to assure their independence and reusability. Finally, the interaction between components has to lead to a functional system.

Considering the reference model introduced in the section 2, we conclude that a component-based visualization system resembles figure 2, where each rectangle represents a component.

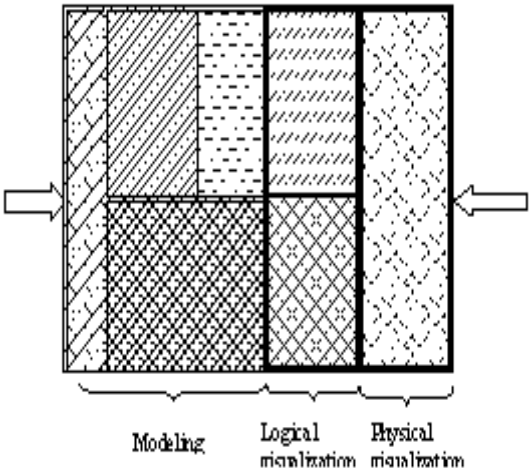


Fig. 2 The architecture of a component based visualization system

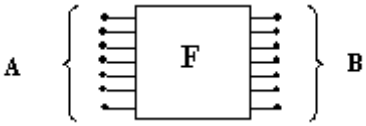


Fig. 3 The graphical description of a software component

4.1. **Software Components.**

4.1.1. *Basic Notions and Definitions.* A *software component* is a binary program that represents the physical encapsulation of related services according to a published specification. [9]

A software component can be seen as an interactive system that communicates asynchronously through channels. The services implemented can be accessed through a consistent and published interface that includes an interaction standard. A component has a *black-box* view captured by the published specification, and a *white-box* view showing implementation details.

The interface defines **a set of channels C**, divided in **a subset** $A = A_1, A_2, ..., A_n$ **of the input channels** and **a subset** $B = B_1, B_2, ..., B_m$ **of the output channels,** meaning that $\mathbf{C} = \mathbf{A} \bigcup \mathbf{B}$. A component can be described graphically as in figure 3.

Any component implements a function F :

$$F : X^n \to Y^m, \;\; (x^1, x^2, ..., x^n) \to (y^1, y^2, ..., y^m)$$

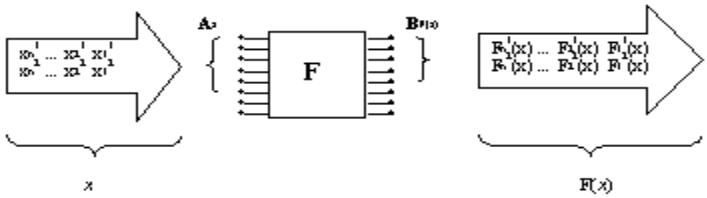


Fig. 4 The way a software component works

which assigns an output on m channels to an input on n channels. In other words it assigns the output data $(y^1, y^2, ..., y^m)$ to an input data $(x^1, x^2, ..., x^n)$. The inputs are timed streams of messages. The component sends new outputs every time the inputs change. This is why a software component can be seen as a system that communicates asynchronously (the component waits for a message; when it is received the component processes it and the result is sent to the output; then, the component waits for the next message). A message can be a string of characters, a binary number, a decimal number etc. Each channel has a stream of messages attached, representing all the messages received (sent) through that channel.

A component needs a time t in order to process a message. We consider that the inputs are introduced at the given moments: $t_0 = 0$, $t_1 = \tau$, $t_2 = 2\tau, ..., t_n = n\tau$. That means that the n-th message is accepted by the component only at the t_n moment.

Consider the input stream of messages x. The following notations are introduced:

- $x(n)$ - the string of the first n messages (until the moment t_n) from x;
- x_n^i - the input message on the channel$i$ at the moment t_n;
- $F(x)$ - the *stream* of messages assigned to the output channels for the input x;
- $F(x)(n)$ - the first n messages from $F(x)$;
- $F_n^i(x)$ - the output message on the channel i at the moment t_n;
- F_x- the *set* of messages assigned to the output channels for the input x;
- A_x - the subset of the input channels for which the input is different from zero, assigned to the input $x(A_x \subseteq A)$;
- $B_{F(x)}$- the subset of the output channels for which the output is different from zero, assigned to the output $F(x)(B_{F(x)} \subseteq B)$.

The way a component works is described in Fig. 4, or simplified in Fig. 5.
The following definitions are introduced:

(a): *The function F is* **consistent** *if for$x = z \Rightarrow F(x) = F(z)$, for any streams of messages x and z.*

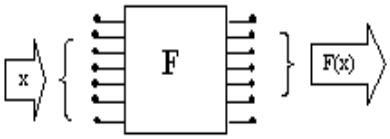


Fig. 5 A simplified scheme of the way a software component works

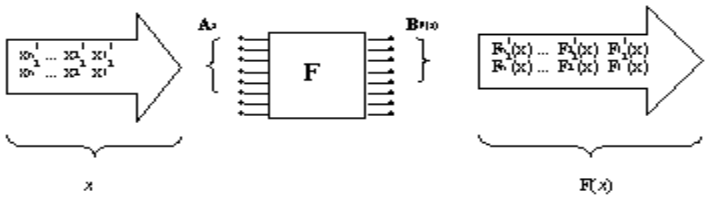


Fig. 6 A simplified scheme for a deterministic behavior of F

The function F is consistent if for identical inputs, identical outputs are obtained.

(b): *The function F is* **causal** *if for any n natural the following condition is fulfilled:* $x(n) = z(n) \Rightarrow F(x)(n) = F(z)(n)$.[9]

If the function F is causal then it is also consistent. The processing time for a causal function is $\tau = 0$. Such a component can not be constructed. We call it *ideal*.

(c): *The function F is* **strictly causal** *if for any n natural the following condition is fulfilled:* $x(n) = z(n) \Rightarrow F(x)(n+1) = F(z)(n+1)$.[9]

We consider $F(x)(0) = F(z)(0)$.

A real component is always strictly causal, meaning that the output at the moment t_{n+1} corresponds to the input at the moment t_n. This component can be implemented because the processing time is $\tau > 0$.

(d): *The function F is* **realizable** *if there is a function strictly* $f : X^n \to Y^m$ *such that for any input x we have*$f_x \in F_x$ *(the output determined by f belongs to the set of the outputs determined by F).[9]*

We denoted by f_x the set of messages assigned by the function f to the output channels, for the input x.

A function f with $f_x \in F_x$ for any input leads to a deterministic behavior of the function F (Figure 6).

(e): The function F is **fully realizable** if it is realizable and for any input x there are **p** functions strictly causal $f_x^i : X^n \to Y^m$, $i = 1, ..., p$ such that [9] $F_x = \cup_{i=1}^{p} f_x^i$

This property guarantees that for every output there is a strategy (a deterministic behavior) that produces this output (Figure 7).

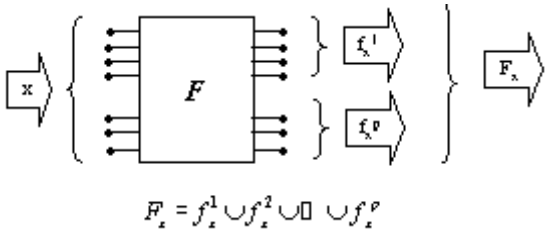


Fig. 7 Graphical description of a fully realizable function F

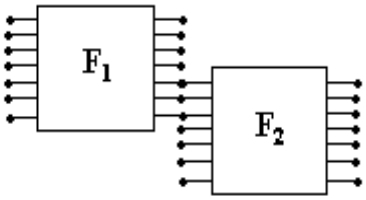


Fig. 8 Two composed components

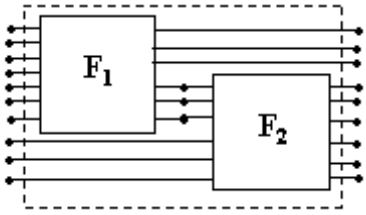


Fig. 9 Two composed components seen as one

(f): *The functionF is* **time-independent** *if the timing of the messages in the input streams does not influence the messages in the output streams but only their timing. [9]*

4.1.2. *Composition Rules.* Channels are assumed to have a type from a given set of types T assigned. A type is a name for a set of data elements. We consider the function *type* that assigns a corresponding type from the set T to each channel from the set C (type: $C \to T$).

Definition: *Two components that implement the functions F_1 and F_2, are said to be* **composed** *if output channels of the first component are used as input for the second one (See Figure 8).*

Remarks:

- Finally the two components can be seen as one (Figure 9).

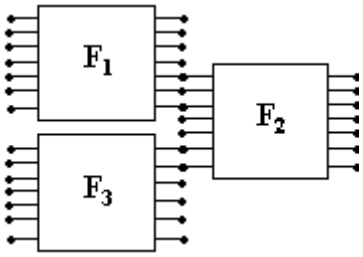


Fig. 10 Three composed components

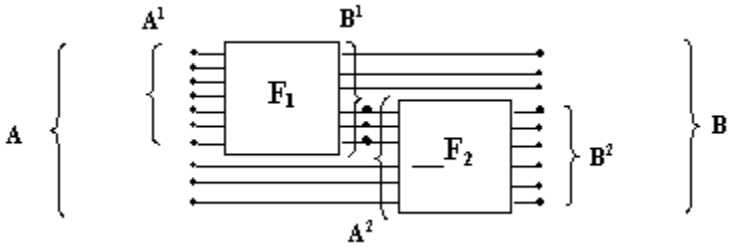


Fig. 11 A detailed view of two composed components

- The definition can be restated for more than two components (Figure 10).

We can say that the composition of two components is reduced to the existence of common elements in the sets B^1 and A^2, where B^1 represents the set of output channels of the first component A^2 represents the set of input channels of the second component. We can state that two components are composed if $B^1 \bigcap A^2 \neq \Phi$.

Condition. Two channels, one output B_n^1 and one input A_m^2, can be used for the composition of the components if and only if $type\ (B_n^1) = type(A_m^2)$.

Consider two composed components (Figure 11).

The following properties can be stated:

a) $A = A^1 \bigcup (A^1 B^1)$, $B = B^2 \bigcup (B^1 A^2)$, $C = A \bigcup B$.

b) Considering that the processing time for the first component is τ_1 and for the second one τ_2, the processing time of the composed component is, $\tau = \tau_1 + \tau_2$, for the worst case

$$F = \begin{cases} F_1, & if\ \ A_x \subseteq AA^1\ \ and\ \ B_{F(x)} \subseteq (BB^2); \\ F_2, & if\ \ A_x \subseteq (AA^1)\ \ and\ \ B_{F(x)} \subseteq B^2; \\ F_1 \circ F_2, & if\ \ AAA - x \subseteq A^1\ \ and\ \ B_{F(x)} \subseteq B^2. \end{cases}$$

c) Using the notations from the first part of the article we describe the function that implements the composed component as follows:

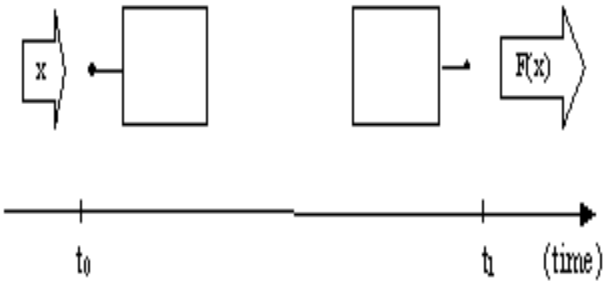


Fig. 12 A one input-one component

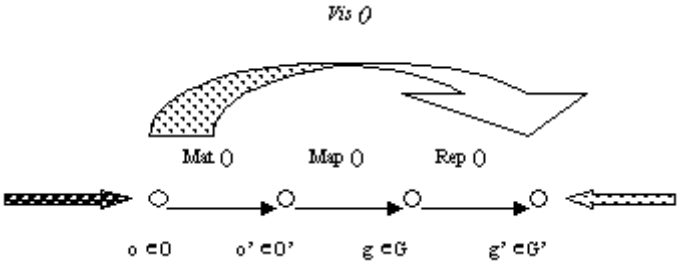


Fig. 13 Synthesis of the proposed visualization model

Remarks:

- $F_1^{\circ}F_2$ means that for an input x we get: $x \to F_1(x) \to F_2(F_1(x)) \equiv F(x)$
- The function described above does not include all cases. For example, the function is not defined for $A_x = A$ and $B_{F(x)} = B$. These cases are solved by decomposition, in order to fit into the given description.

4.2. **The Visualization Pipeline Using Components.** Consider the case of one input -one output component. The function F describing the component is considered to be time-independent *(definition (f))*. Two working steps can be revealed: at moment t_0 a message is introduced into the input channel and at moment t_1 the result is read from the output channel (Figure 12)

We considered that the implemented services are **consistent** *(definition (a))*. The component is **strictly causal** *(definition (c))* and **fully realizable** *(definition (e))* because the strictly causal function needed is F.

The reference model introduced to describe the visualization process contains three distinct steps (Figure 13).

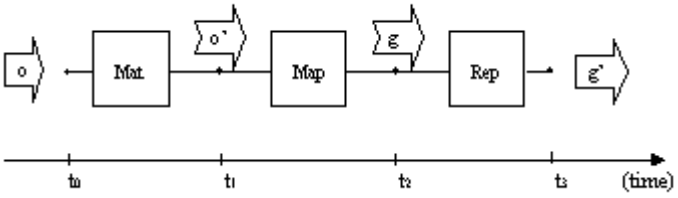


Fig. 14 A component-based visualization system

Suppose that each step/function is performed by an independent component. Then the visualization pipeline becomes the one in figure 14. The components should take into account the scientific visualization criteria, meaning that the *Mat(),Map(),Rep()* functions implemented by the components should be injective and *Vis()* should be bijective.

A further decomposition can be performed. Each component can be obtained from the composition of two or more components depending on the visualization process. A personalized visualization system is obtained at this level.

4.3. **Example: Visualization of natural gas reservoir using a component based visualization system.** In order to present a detailed component based visualization system we consider the example of a natural gas reservoir. A nonuniform data set obtained using drills is visualized. The data set is processed using two paths (Figure 15).

The filter component transforms the nonuniform input data into a uniform data set using the geological characteristics of the underground and the methane concentration. The filter also performs the sub-sampling of the data set. One of the paths includes the intersection of the data set with a plane. The generation of the isosurfaces represents another step of the modeling level.

The colors, the light, the position of the camera and other properties are set at the logical visualization level. A human-computer interface (HCI) allows the user to change the properties without interfering with the data flow.

Finally, two objects that can be rotated, translated and zoomed are obtained (Figure 16): one used for the dimensional exploration of the deposit and another for the visualization of the level lines. (The presented images were obtained using the VTK library, and the components were implemented using Tcl/Tk)

5. Conclusion

The paper addresses the issue of component-based scientific visualization systems. The visualization process validation criteria are presented in detail and a reference model for the visualization systems is proposed. The paper supports the idea that a component- based architecture solves most of the standard/commercial

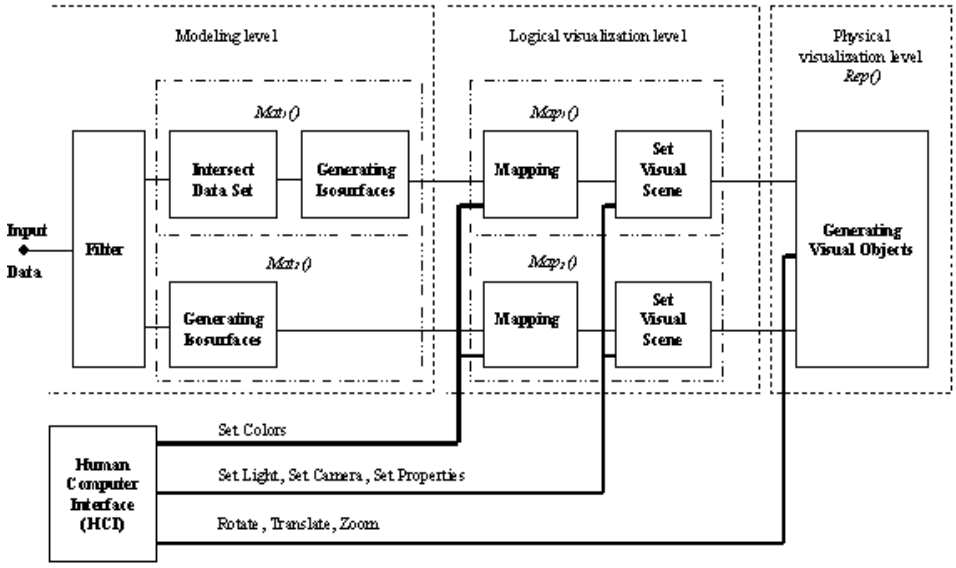


Fig. 15 The architecture of a custom made system used to visualize a natural gas reservoir

visualization system problems. It should be noted that the introduction of the component-based technology does not solve lead automatically to the wide spread of customizable scientific visualization systems unless a critical mass is reached. The theory is supported by a detailed example of a visualization application of a natural gas reservoir.

References

[1] Upson C., Faulhaber, Jr. T., Kamins D., Laidlau D., Schelgel D., Vroom J., Gurwitz R., van Dam A., *The Application Visualization System: A Computational Environment for Scientific Visualization*, Computer Graphics and Applications, vol. 9, nr.4, 1989.

[2] Radoiu D., Roman A., *Modelarea procesului de vizualizare, in Tehnologii Avansate Aplicatii in educatie*, Editura Universitatii Petru Maior, 1999, p.86-101

[3] Kaufman Arie, Nielson G., Rosenblum L. J., *The Visualization Revolution*, IEEE Computer Graphics, July 1993, p. 16-17.

[4] Hibbard W., Dyer C., Paul B., *A lattice Model for Data Display*, Proceedings of IEEE Visualization '94, 1994, pp. 310 - 317.

[5] Hibbard Williams L., Dyer Charles R., Brian E. Paul, *Towards a Systematic Analysis for Designing Visualizations*, Scientific Visualization, IEEE Computer Society, 1997, pp. 229 - 251.

[6] Radoiu D., *Scientific Visualization of Experimental Data*, Ph.D.Thesis, Universitatea Babes Bolyai, 1999.

Fig. 16 The visualization of the natural gas reservoir

[7] Radoiu D., *On Scientific Visualization Systems Design*, Studia, 1998.
[8] MacKinlay, *Automating the Design of Graphical Presentations of Relational Information*, ACM Transactions on Graphics, Vol.5, Nr.2 1986, pp.110-141.
[9] M.Broy, *Software Concept and Tools*, Springer-Verlag, 1998, 57-59.
[10] D. Radoiu, *Vizualizarea Stiintifica a Datelor Experimentale*, Editura Universitatii Petru Maior, 2000.

Petru Maior University of Tirgu Mures
E-mail address: dradoiu@uttgm.ro

Polytechnic University of Bucharest

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

EXTENDING STATECHARTS FOR CONCURRENT OBJECTS MODELING

DAN MIRCEA SUCIU

Abstract. Object-oriented concurrent programming is a methodology that seems to satisfy nowadays requirements for complex applications development. The fundamental abstractions used in this methodology are concurrent (or active) objects and protocols for passing messages between them. Statecharts seem to be one of the most appropriate ways of modeling the behavior of concurrent objects. Based on statecharts we defined an executable formalism, called *scalable statechart*, for effective modeling of object-oriented concurrent applications with respecting of *homogeneous object model*.

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

1. Introduction

Object-oriented concurrent programming is based on object-oriented programming methodology, which is known at this moment as a top methodology for developing reusable applications. This methodology is conceptually simple and wide applicable and is based on two fundamental concepts:*objects* (that identify knowledge) and *message passing* (that is an unified protocol for communicating between objects).

The idea of building programming languages that can integrate the object- oriented programming mechanisms with concurrency mechanisms is very attractive. To achieve an optimal integration of these mechanisms is very useful to identify objects as activity units and to associate synchronizing code at the message passing level. These objects are often called *concurrent* objects, *active* objects or*actors*. The result of this unification is the integration of all the object-oriented programming and concurrency concepts and it allow the programmer not to be explicitly involved in establishing the synchronization discipline.

In section 2 we will describe and detail the key concepts of the object-oriented concurrent programming. Section 3 contains the structural description of a general concurrent (active) object that belongs to *homogeneous* object model, based on studies realized on over 100 object-oriented concurrent languages [SUC98].

Statecharts represent a visual formalism for describing states and transitions in a modular fashion, enabling nesting, orthogonality and refinement [HAR87],

[OMG99]. Statecharts are used for specifying objects behavior in designing complex systems. In section 4 we propose an extension of statecharts for modeling the behavior of concurrent (active) objects. The formal description of our statechart, called *scalable* statechart, proves its consistency and executability.

2. The context of object-oriented concurrent programming

In object-oriented concurrent programming a system is viewed as a physical simulation model of real or conceptual world behavior. This physical model is defined with a particular programming language and is materialized through an application.

Objects are key elements of object-oriented concurrent programming and they represent real or abstract entities with clear defined role into a system. An object has *identity, state* and *behavior*. All that an object knows *(state)* and can do *(behavior)* is expressed by sets of *properties* (or*attributes)* and *operations* (or *methods).* The state of an object is given by the values of its properties. The operations implement the object's behavior and they are procedures or functions that eventually modify the value of properties.

The object-oriented concurrent applications are composed by a set of objects that interact and communicate between them through *messages*. A *message* is a request for execution of an object's operation and it is composed of three elements: the identity of the receiver object, the name of the requested operation and a list of parameters. The mechanism of message passing allows objects to comunicate between them even if they are in different processes, contexts and/or computers. Because the entire activity of an object is revealed by its operations, the mechanism of message passing can express all the possible interactions between objects.

The process of identification of sets of objects with common properties and behaviors is called *classification.* The *class* is another key concept of object-oriented concurrent programming and represents the abstraction of the common elements (properties or operations) shared by a set of objects and describes their implementation.

The objects are concrete representations of classes and the process of building a particular object based on its class definition is called instantiation. The *concurrent* feature of a programming language represents the capacity of that language to express a potential parallelism. The object-oriented concurrent languages allow building applications where two or more operations are parallely executed, in distinct threads.

Based on the nature of relation between objects and threads, the concurrent object models can be classified in three categories: *orthogonal, homogeneous* and *heterogeneous* [PAP89].

In *orthogonal* approach the objects and threads are viewed as independent concepts. The objects are not implicitly protected by concurrent operation calls. Thus

the protection of the internal state is explicitly realized through low-level synchronization mechanisms, like semaphores or conditional critical regions [PHI95].

The *homogeneous* approach introduces the concept of *concurrent* (or *active) object*. An active object is an object that *controls* and *schedules* the execution of its operations. Active objects own threads, which in most homogeneous object models are implicitly created when a message is received. These objects may or may not be implicitly protected by external concurrent calls and contain specific mechanism for explicit protection of their internal state *(method guards, behavior abstractions, enable sets* etc [PHI95]). The logical conditions used in synchronizing the concurrent operations of active objects are called *synchronization constraints.*

In *heterogeneous* approach there are two kinds of objects: *active* and *passive*. Passive objects not own threads, are not protected (implicitly or explicitly) by external concurrent calls and their operations are executed in threads owned by caller objects.

3. The homogeneous concurrent object model

As we stated in section 2, the active objects that belong to homogeneous concurrent object models can control and schedule the receiving messages to protect their internal state by concurrent operation calls. The protection is implicit, when mechanisms with external control are used *(monitor*-like mechanisms) or explicit, in case of mechanisms with mixed or reflective control *(method guards, enable sets* etc) [SUC98].

In figure 1 is presented the structure of a general active object. The *interface manager* is a special entity located at each active object level. This entity controls and schedules the received messages and is materialized into a particular programming language by a distinct thread, a locking mechanism or a special object encapsulated in the kernel of active objects.

The interface manager controls the messages handling through asynchronous executions of associated operations based on object's state, synchronization constraints values and/or current executed operations. The messages scheduling is achieved through a special structure called *messages queue*, which retains all received and not handled messages. The interface manager, the messages queue, the properties and the synchronization constraints are not externally visible. Furthermore, just a subset of operations is visible and this subset represents the interface of the active object.

4. Scalable statecharts

In this section we define an extension of statecharts formalism for modeling the behavior of active objects corresponding to homogeneous object model. We define the scalable statecharts in an incremental way, starting from finite state machines (that we will call *level 0 scalable statecharts)* and adding elements to handle

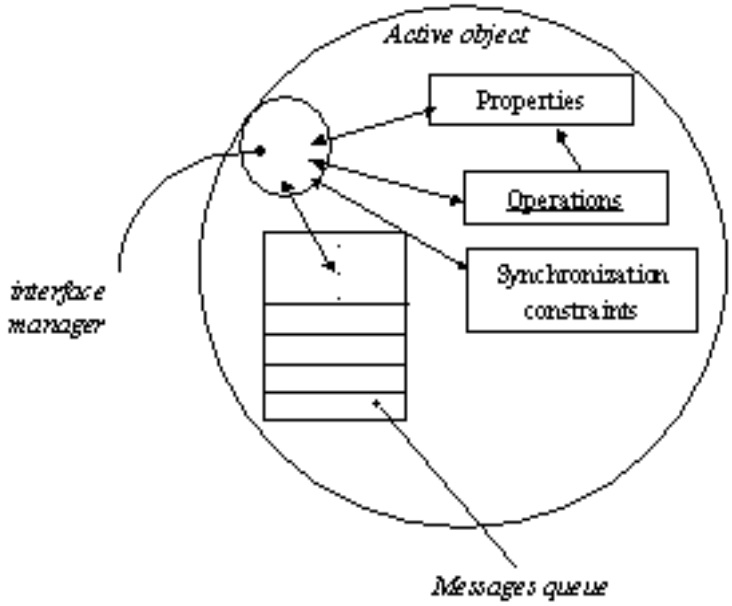


Figure 1. The structure of a general active object

the depth, orthogonality etc. For each intermediary statechart, the *configuration* and *execution* notions will be defined. The approach of definitions is a compositional one, where the execution of a statechart can be expressed by the executions of its components. The scalable statecharts represent an original way of modeling active objects behavior. They are closely related with particular concurrent object-oriented concepts, like *messages, operations, properties* or*synchronization constraints.* Therefore, the scalable statecharts are specialized versions of statecharts used in reactive systems modeling and they extend UML statecharts with specific elements of homogeneous object model presented in section 3.

4.1. **Level** 0 **scalable statecharts** (SS^0)**. Definition 1.** A *level 0 scalable statechart* of a class K is a tuple: $SS^0_K = (M, S, s_0, S_F, T; s_aC)$, where:

- M - is a finite set of messages. We will consider, without affecting the generality, that the messages signature do not contain parameters. The generalization of statecharts considering messages parameters is immediately and does not affect the semantics of statecharts execution. We will use $\perp$ to symbolize the empty message.
- S - is a finite, non-empty set of states,
- $s_0 \in S$ - is the initial state,

- SF *is a finite set of final states.* When S_F is empty, the modeled objects can not destroy themselves.
- $T \subseteq S \times M \times (S \cup S_F)$ is a finite set of transitions. A transition $(s', m, s'') \in T$ means that if an object is in state s' and receives the message m then, after handling of message m (in fact, after terminating of execution of the attached operation), the object will be in state s''.
- $s_a \in S$ - the active state of the statechart in a given moment,
- $C \in M^*$ is a finite sequence of messages, and models the messages queue of an active object. We will figure with $C = m_0 \widehat{} C_r$, where m_0 is the first message of the sequence and C_r represents the rest of sequence (the symbol $\widehat{}$ denotes the operation of concatenation).

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

The first five elements of SS^0 form the static component and they describe the *structure* of the statechart. These five elements are shared by all objects of class K for which SS^0_K is defined and they are not modified by objects execution. The dynamic component consists of active state s_a and messages queue C and not describes the objects behavior. We will use the dynamic component for the *execution* of statechart.

Definition 2. The *configuration* of a SS^0 statechart is a tuple: $(s,_a\, m \widehat{}_0 C_r) \in S \times M^*$, where s_a is the active state and m_0 is the first message from queue C at a given moment. The *initial configuration* of a SS^0 statechart is given by tuple $(s_0, \perp)$.

The *execution* of an active object modeled with a SS^0 statechart lay in sequential *interpretation* of messages from the queue C. The interpretation of a message implies the modifying of statechart configuration or the returning of that message in the queue C.

Definition 3. The *interpretation* of a SS^0 statechart configuration is a function: $\delta^0 : S \times M^* \to (S \cup S_F) \times M^*$,

$$\delta^0(s_a, m_0 \widehat{} C_r) = \begin{cases} (s', C'_r), & \text{if } \exists (s_a, m_0, s') \in T \\ (s_a, C'_r), & \text{if there is no } (s_1, s_2 \in S : (s_1, m_0, s_2) \in T \\ (s_a, C'_r \widehat{} m_0), & \text{otherwise} \end{cases}$$

In parallel with the interpretation of a configuration, the queue C may suffer some modifications. In the above definition of function δ^0 it is possible that $C_r \neq C'_r$ (in general, $C'_r = C_r^R$, where $R \in M^*$ represents the sequence of messages received by an object in the time of interpretation).

The interpretation of a SS^0 configuration models the functionality of the interface manager of active objects. A message will be accepted and its attached operation is executed if the message labels a transition that leaves the active state or if it does not appear in any transition label. Otherwise, the message is returned

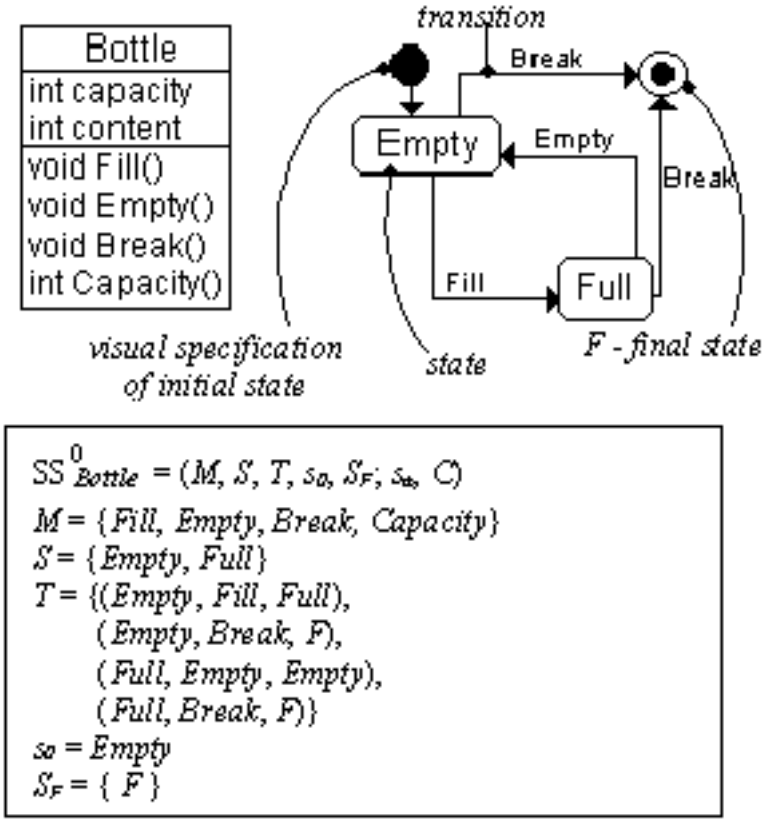


Figure 2. Graphical representation of SS^0statecharts

in the message queue. When it exists more than one transition labeled with the same message which leaves the active state, the selection of the interpreted transition is non- deterministic. To avoid the non-determinism from statecharts is possible to attach priorities to transitions.

Definition 4. The *execution* of a SS^0 statechart is a finite or infinite sequence of configuration interpretations, starting from the initial configuration, and it is denoted by:

$$(s_o, \perp) \xrightarrow{\delta^0} (s_1, m_1 \widehat{\ } C_{r1}) \xrightarrow{\delta^0} ... \xrightarrow{\delta^0} (s_k, m_k \widehat{\ } C_{rk}) \xrightarrow{\delta^0} ...,$$

where $s_0, s_1, , s_k, .. \in S$, $m_1, ..., m_k, ... \in M$ and $C_{r1}, ..., C_{rk}, ... \in M^*$. The execution is finite if a final state becomes an active state.

A possible execution of an object of class *Bottle* defined in figure 2 is:

$(Empty, \perp) \xrightarrow{\delta^0} (Empty, < Fill >< Fill >< Capacity >) \xrightarrow{\delta^0}$
$(Full, < Fill >< Capacity >< Break >) \xrightarrow{\delta^0}$
$(Full, < Capacity >< Break >< Fill >) \xrightarrow{\delta^0}$
$(Full, < Break >< Fill >) \xrightarrow{\delta^0} (F, < Fill >).$

The message queue C described in definition 1 models a particular mechanism for choosing the next handled message. This mechanism was selected to simplify the description of *interpretation* and *execution* notions. In general, the synchronization mechanisms used in object-oriented concurrent languages are more complex, and allow attaching priorities to messages or have particular policies of selecting of the right message. These mechanisms can be modeled by replacing the queue C with the pair (C', pol), where $C' \in M^*$ and *pol* is a function $pol : M^* \rightarrow M$ that describes the choosing policy of a message from the set of received messages, modeled by C'. In this case, in all previous expressions the message m_0 will be replaced with $pol(C')$.

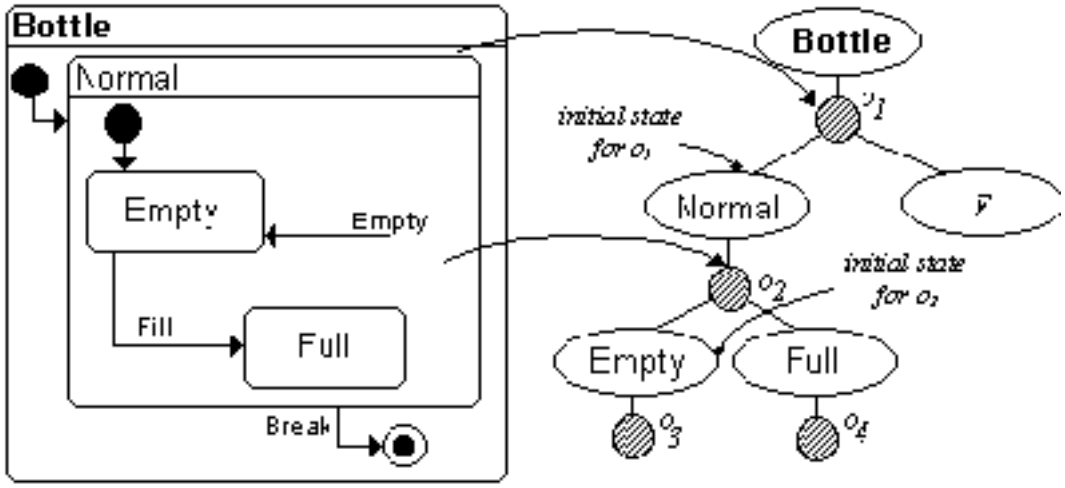
4.2. **Level 1 scalable statecharts** (SS^1)**.** We will attach to SS^0 statecharts the notions of *depth* and *orthogonality* introduced in [HAR87]. Because these extensions allow objects to be in more than one state, in distinct orthogonal components, we will extent the definitions of configuration, interpretation and execution of statecharts.

The notions of depth and orthogonality are modeled in SS^1 through a heterogeneous tree. The root of the tree and the intermediary nodes from even levels are states and the nodes from odd levels are orthogonal components (figure 3). We consider that all states have at least one orthogonal component and each orthogonal component can have zero or more states. The three types of states introduced in [HAR87] can be unitary modeled in this way:

- the *simple states* are states that have only one empty orthogonal component,
- the *composed states (OR-states)* are states that have only one non- empty orthogonal component,
- the *orthogonal states (AND-states)* are states that have more than one non-empty orthogonal components.

Definition 5. *A level 1 scalable statechart* of a class K is a tuple: $SS^1_K = (M, S, O, s_R, S_F, (stSucc, stInit, ortSucc), T; S_a, C)$, where:

- M - is a finite set of messages,
- S - is a finite, non-empty set of states,
- O - is a finite, non-empty set of orthogonal components,
- $s_R \in S$ - is the *root* of the states hierarchy,
- S_F is a finite set of final states. To preserve the consistency of our model we will presume that all the final states will be successors of



$$SS^{Bottle} = (M, S, O, s_R, S_F, (stSucc, stInit, ortSucc), T, C)$$

$M = \{Fill, Empty, Break, Capacity\}$
$S = \{Bottle, Normal, Empty, Full\}$
$O = \{o_1, o_2, o_3, o_4\}$
$s_R = Bottle$
$S_F = \{F\}$

$stSucc : \{o_1, o_2, o_3, o_4\} \to \wp(\{Bottle, Normal, Empty, Full, F\})$,
$stSucc(o_1) = \{Normal, F\}$, $stSucc(o_2) = \{Empty, Full\}$,
$stSucc(o_3) = \varnothing$, $stSucc(o_4) = \varnothing$,
$stInit : \{o_1, o_2\} \to \{Bottle, Normal, Empty, Full\}$,
$stInit(o_1) = Normal$, $stInit(o_2) = Empty$,
$ortSucc : \{Bottle, Normal, Empty, Full\} \to \wp(\{o_1, o_2, o_3, o_4\}) \setminus \{\varnothing\}$,
$ortSucc(Bottle) = \{o_1\}$, $ortSucc(Normal) = \{o_2\}$,
$ortSucc(Empty) = \{o_3\}$, $ortSucc(Full) = \{o_4\}$,

$T = \{(\{Empty\}, Fill, \{Full\}), (\{Full\}, Empty, \{Empty\}),$
$(\{Normal\}, Break, \{F\})\}$

Figure 3. Graphical representation of SS^1 statechart

orthogonal components from the root state s_R. Thus we will eliminate the termination transitions proposed in UML [OMG99] without affecting the modeling power of the statecharts.

- functions that defines the states hierarchy:
  - $stSucc : O \rightarrow \wp(S \cup S_F)$, where $stSucc(o) = s_1, s_2, ..., s_n$ is the set of sub-states of the orthogonal component o, with the restriction that $\forall o_1, o_2 \in O$ if we have $stSucc(o_1) \cap stSucc(o_2) = \phi$;
  - $stInit : O \backslash \{o : stSucc(o) = \phi\} \rightarrow S, stInit(o) = s_0 \in stSucc(o)$, the initial sub-state of the orthogonal component o ($stSucc$ is defined only for non-empty orthogonal components);
  - $ortSucc : S \rightarrow \wp(O) \backslash \{\phi\}$, where $ortSucc(s) = o_1, o_2, ..., o_m$ is the set of the orthogonal components owned by the state s, with the restriction that $\forall s_1, s_2 \in S$ we have $ortSucc(s_1) \cap ortSucc(s_2) = \phi$ (a state has at least one orthogonal component);
- $T \subseteq \wp(S \backslash \{s_R\}) \times M \times \wp(S \backslash \{s_R\})$ is a finite set of transitions. A transition $(s'_1, ..., s'_i), m, s''_1, ..., s''_j) \in T$ means that if an object is in *source* states $s'_1, ..., s'_i \in S \backslash \{s_R\}$ (each source state is located in distinct orthogonal components of a state from S) and receives a message m then, after executing the operation associated with m, the object will enter in *destination* states $s''_1, ..., s''_j \in S \backslash \{s_R\}$. The root state can't be source nor destination for a transition and the sets of source states and destination states do not contain states that include each other.
- $S_a \subseteq S \cup S_F$ - is the set of *active states* of the statechart in a given moment with the restriction that $\forall s_a \in S_a, ortSucc(s_a) = \phi$,
- $C \in M^*$ is a finite sequence of messages, and models the messages queue of an active object.

If an instance of class *Bottle* (modeled in figure 3) is in state *Full* then, corresponding to the states hierarchy, the instance it is in states *Normal* and *Bottle* too. To define in a unique way the *configuration* of a SS^1 statechart we extended the notion of *active* state. In definition 5 an active state of a SS^1 statechart has the property that is a simple state (has only one empty orthogonal component). Corresponding to definition 5, for the example from figure 3 the only states that can become active are *Empty, Full* and F.

Definition 6. A *pseudo-active* state is a composed state that contains an active sub-state. We denote with $S_{pa} \subseteq S$ the finite set of pseudo-active states of a SS^1 statechart in a given moment.

A SS^1 statechart can have more than one active state, each located in distinct orthogonal components of a pseudo-active state. It is obviously that the root state s_R is always pseudo-active.

If a composed state is a destination state of a transition and the transition is triggered, then its initial sub-states will be activated. We will define a recursive

function that will be used to determine the active states when a statechart enters in a composed state.

Definition 7. The *activation function* is a function that associates to each state $s \in S$ its simple or final sub-states that will be implicitly activated when an SS^1 statechart enters in state s. We will denote this function: $activ : S \cup S_F \rightarrow \wp(S \cup S_F)$,

$$activ(s) = \begin{cases} s, \text{ if } (s \in S, ortSucc(s) = 0, stSucc(0) = \Phi)0rs \in S_F \\ \bigcup_{0 \in ortSucc(s)} activ(stInit(0)), \text{ otherwise} \end{cases}$$

The *global activation function*, denoted by:

$$Activ : \wp(S \bigcup S_F) \rightarrow \wp(S \bigcup S_F), Activ(S') = \bigcup_{s \in S} activ(s),$$

associates to a set of states their implicitly activated sub-states.

Definition 8. A *configuration* of a SS^1 statechart is a tuple $(S_a, m_0 \widehat{} C_r)$, where $S_a \subseteq S$ is the finite set of active states and $m_0 \widehat{} C_r \in M^*$ represents the content of the messages queue C in a given moment. The *initial configuration* of a SS^1 statechart if given by $(active(s_R), \perp)$.

Definition 9. The interpretation of a SS^1 statechart configuration is a function: $\delta^1 : \wp(S) \times M^* \; rightarrow \wp(S \bigcup S_F) \times M^*$,

$$\delta^1(S_a, m_0 \widehat{} C_r) = \begin{cases} (Activ(S"), C'_r), if (S', m_0, S") \in T si S' \subseteq S_a \cup S_{pa} \\ (S_a, C'_r), if \exists S_1, S_2 \subseteq S : (S_1, m_0, S_2) \in T \\ (S_a, C'_r \widehat{} m_0), otherwise \end{cases}$$

Definition 10. The execution of a SS^1 statechart is a finite or infinite sequence of configuration interpretations, starting from the initial configuration, and is denoted:

$$(active(s_R), \perp) \overset{\delta^1}{\rightarrow} (S_1, m_1 \widehat{} C_{r1}) \overset{\delta^1}{\rightarrow} ...(S_k, m_k \widehat{} C_{rk}) \overset{\delta^1}{\rightarrow} ...,$$

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

A possible execution of the statechart for an object of *Bottle* class (figure 3) is:

$(Empty, \perp) \overset{\delta^1}{\rightarrow} (Empty, < Empty >< Fill >< Capacity >) \overset{\delta^1}{\rightarrow}$

$(Empty, < Fill >< Capacity >< Empty >) \overset{\delta^1}{\rightarrow}$

$(Full, < Capacity >< Empty >< Break >) \overset{\delta^1}{\rightarrow}$

$(Full, < Empty >< Break >) \overset{\delta^1}{\rightarrow} (Empty, < Break >) \overset{\delta^1}{\rightarrow} (F, \perp).$

5. Conclusions

In the third section we ascertained a general structure of active objects. The implementation of this model is retrieved in most concurrent object-oriented languages that use synchronization mechanisms that belong to homogeneous concurrent object models. In section four we propose a formalism for modeling of active

objects behavior, called *scalable statechart*, that is based on statecharts visual formalism described by Harel in [HAR87]. The executability of scalable statecharts is a fundamental feature for automatization of active objects implementation. Moreover, the executability allows testing, simulating and debugging of active objects at the same level of abstraction as their behavioral model. In this way the conceptual gap between the formal models of active objects behavior and debugging at their source code level is avoided. Because the scalable statechart was defined having in mind a general model of active objects, it can be used for code generation in any concurrent object-oriented programming language that contains communication and synchronization mechanism belonging to homogeneous approach. This property gives more flexibility in translation of behavioral models in source code.

References

[BAR98] F. Barbier, H. Briand, B. Dano, S. Rideau, *The Executability of Object-Oriented Finite State Machines*, Journal of Object-Oriented Programming, SIGS Publications, 4(11), pp. 16-24, jul/aug 1998.

[BEE94] Michael von der Beeck, *A Comparison of Statecharts Variants*, Formal Techniques in Real-Time and Fault-Tolerant Systems,, L. de Roever & J. Vytopil (eds.), Lecture Notes in Computer Science, vol. 863, pp. 128-148, Springer-Verlag, New York, 1994.

[COO94] S. Cook, J. Daniels, *Designing Object Systems - Object-Oriented Modelling with Syntropy*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[DOR96] Dov Dori, *Unifying System Structure and Behavior Through Object-Process Analysis*, Journal of Object-Oriented Programming, SIGS Publications, 4(9), pp. 66-73, jul/aug 1996.

[DOU99] Bruce Powel Douglas, *UML Statecharts*, Embedded Systems Programming, jan. 1999, available at http://www.ilogix.com/fs_prod.htm.

[GAN93] D. Gangopadhyay, S. Mitra, *ObjChart: Tangible Specification of Reactive Object Behavior*, Proceedings of ECOOP'93, Oscar M. Nierstrasz (ed.), Lecture Notes in Computer Science, vol 707, pp. 432-457, Springer-Verlag, 1993.

[HAR87] David Harel, *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming, vol.8, no. 3, pp. 231-274, June 1987.

[HAR96] D. Harel, A. Naamad, *The STATEMATE Semantics of Statecharts*, ACM Transactions on Software Engineering and Methodology, 5(4), pp. 293-333, 1996.

[HAR97] D. Harel, E. Gery, *Executable Object Modeling with Statecharts*, IEEE Computer, 30(7):31-42, Jul. 1997.

[MAN74] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, 1974.

[OMG99] Object Management Group, *OMG Unified Modeling Language Specification*, ver. 1.3, June 1999 available on Internet at http://www.rational.com/.

[PAP89] Michael Papathomas, *Concurrency Issues in Object-Oriented Programming Languages*, in D. Tsichritzis, editor, Object Oriented Development, pg. 207-245, University of Geneva, Switzerland, 1989.

[PHI95] Michael Phillipsen, *Imperative Concurrent Object-Oriented Languages*, Technical Report TR-95- 049, International Computer Science Institute, Berkeley, aug. 1995.

[SCU97] Marian Scuturici, Dan Mircea Suciu, Mihaela Scuturici, Iulian Ober, *Specification of active objects behavior using statecharts*, Studia Universitatis "Babes Bolyai", Informatica, Vol. XLII, no. 1, pp.19-30, 1997.

[SUC98] Dan Mircea Suciu, *Reuse Anomaly in Object-Oriented Concurrent Programming*, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLII, no. 2, pp. 74-89, 1997.

"Babes-Bolyai" University, Faculty of Mathematics and Computer Science, RO-3400 Cluj-Napoca, Romania

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

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

HALF SYNCHRONIZED TRANSITION SYSTEMS

FLORIAN MIRCEA BOIAN AND CORINA FERDEAN

Abstract. In a distributed system, defined as a collection of interconnected nodes, the underlying role is taken by the communication subsystem. It functions using common protocols which addresses the problem of heterogeneity and homogeneity of the participants nodes, and contributes to the performance of the whole system. No matter what type: *synchronous* or *asynchronous*, the communication software has to be flexible, reusable and adaptable. The formal methods used to model these protocols should reflect these qualities. In this paper, we extend the classical, synchronous and asynchronous message passing models, proposing an intermediary class of models, starting from some limitations, in time and space, imposed upon the entities involved. We named the new model, *half-synchronized transition systems.*

Keywords: distributed systems, distributed algorithms, transition systems, synchronous, asynchronous message passing

1. Introduction

In a distributed system, defined as a collection of interconnected nodes, the underlying role is taken by the communication subsystem. We cannot have distributed computing if there is no communication. Also, the communication protocols addresses the problem of heterogeneity and homogeneity of the participants nodes, and together with the communication media, have a profound influence on the performance of the whole system. These observations lead to the conclusion that the communication software, used to design distributed applications, must have important qualities, among which we mention: flexibility, efficiency, reusability, adaptability.

Also, the communication medium and applications domain properties have a major influence on designing the communications protocols. These protocols can be selected as a compromise between some competing properties. In order to achieve its main purposes, as the underlying system for collaborating between distributed applications, the communication protocols take many forms and they can be modelled in different ways, using abstract languages.

Besides the introduction, this paper is structured in two main parts. The first part presents the two major ways *synchronous* and *asynchronous-,* of modeling communication, between distributed processes, using transition systems. In the second part, the paper extends the models mentioned above and proposes an

intermediary class of models, starting from some limitations, in time and space, imposed upon the entities involved. We named the new model, *half-synchronized transition systems*, and we will describe it both intuitively and formally.

We begin the presentation with the definitions of some background terms, that we will use in describing the models.

Definition 1.1. *A* distributed system *is an interconnected collection of autonomous nodes, which can be computers, processes or processors [7]. The nodes must at least be equipped with their own private control and they are capable of exchanging information. A more restrictive definition [6] considers a system to be distributed only if the existence of autonomous nodes is* transparent *to users of the system.*

We consider a sequence of processes $P = (p_1, p_2, ..., p_N), S_{p_i}$ the set of the possible states associated with the process p_i, I_{p_i} the initial states of the process $p_i (\forall i \in 1, ..., N)$ and a set of messages M, which can be transmitted (sent and received) between the processes.

In the following, the relations " $>^I$ ", " $>^S$ ", " $>^R$ " associated with the *internal, send*, respectively *receive* events, will be presented.

Definition 1.2. *The binary relation* " $>^I$ " *defined on* $S_{p_i} \times S_{p_i}$, *contains the following pairs of states of the process* p_i*:* (s_{p_i}, u_{p_i}), *which means that an internal event of the process* p_i *is produced, the process* p_i *changes its state from* s_{p_i} *to* u_{p_i}, *but the communication subsystem remains unchanged.*

Definition 1.3. *The ternary relation* " $>^S$ " *defined on* $S_{p_i} \times M \times S_{p_i}$, *contains the following triples:* (s_{p_i}, m, u_{p_i}), *where the process* p_i *executes a* send *event with the message* m *and changes its state from* s_{p_i} *to* u_{p_i}.

Definition 1.4. *The ternary relation* " $>^R$ " *defined on* $S_{p_i} \times M \times S_{p_i}$, *contains the following triples:* (s_{p_i}, m, u_{p_i}), *where the process* p_i *executes a* receive *event with the message* $m \in M$ *and changes its state from* s_{p_i} *to* u_{p_i} .

Definition 1.5. *The* local algorithm *of a process* p_i *is a quintuple* $(S_{p_i}, I_{p_i}, >^I, >^S, >^R)$, *where all the 5 elements were defined above .*

Definition 1.6. *A* distributed algorithm *is defined as a collection of local algorithms, for the sequence of processes* $P = (p_1, p_2, ..., p_N)$ *and the set of messages* M. *These messages can be exchanged between processes, using pairs of* send-receive *events with the same message* $m \in M$.

Definition 1.7. *A* configuration *is an* $N + 1$*-uple of the following form:* $C = (s_{p0}, ..., s_{p_i}, ..., s_{p_N}, M')$, *where* $\forall i \in \{1, ..., N\}$ s_{p_i} *is a state from* S_{p_i}, *and* $M' \subseteq M$ *is a queue associated with the messages* sent, *but not* received *yet (messages which are in transit [1]).*

Definition 1.8. *An* initial configuration *is a configuration of the following form:* $I = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M)$, *where* $s_{p_i} \in I_{p_i} \forall i \in \{1, ..., N\}$ *and the messages queue* M *is empty.*

In the following, the *transition relations* "$\rightarrow_{p_i}$" and "$\rightarrow_{p_ip_j}$" on $C \times C$, will be defined:

Definition 1.9. *The transition relation "$\rightarrow_{p_i}$" is defined as the set of the following pairs of configurations:* $((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M_1)$, $(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M_2))$, *where one of the following conditions take place:*

- $(s_{p_i}, u_{p_i}) \in >^I$ *and* $M_1 = M_2$,
- $\exists m \in M$ *such that* $(s_{p_i}, m, u_{p_i}) \in >^S$ *and* $M_2 - m = M_1$.
- $\exists m \in M$ *such that* $(s_{p_i}, m, u_{p_i}) \in >^R$ *and* $M_1 - m = M_2$.

Definition 1.10. *The transition relation "$\rightarrow_{p_ip_j}$", is defined as the set of the following pairs of configurations:* $((s_{p_0}, ..., s_{p_i}, ..., s_{p_j}, ...)$, $(s_{p_0}, ..., u_{p_i}, ..., u_{p_j}, ...))$, *where* $\exists m \in M$, $(s_{p_i}, m, u_{p_i}) \in >^S \wedge (s_{p_j}, m, u_{p_j}) \in >^R$. *In this case, there is an exchange of messages from the process* p_i *to the process* p_j. *The process* p_i *produces the message* m, *initiating a* send *event, and the process* p_j *consumes the message, which leads to a* receive *event.*

2. Synchronous and asynchronous transitions systems

Definition 2.1. *The behavior of a distributed algorithm can be described using a* transition system, *defined as a triple* $T = (C, \rightarrow, I)$, *where:*

- C *is the set of the possible configurations.*
- *"$\rightarrow$" is a binary relation from* $C \times C$, *called* transition relation, *and it is defined as a reunion of the transition relations "$\rightarrow_{pi}$", associated with the* N *processes.*
- I *is the set of initial configurations.*

Definition 2.2. *We define a* terminal configuration, *a configuration* $c \in C$ *which doesn't have a successor configuration (there is no configuration* $c_1 \in C$ *which verify* $c \rightarrow c_1$).

Definition 2.3. *An* execution of T *is defined as a maximal sequence* $E = (c_0, c_1, c_2, ...)$, *where* $c_0 \in I$ *and* $\forall i, c_i \rightarrow c_{i+1}$.

Putting it into words, the execution of a transition system can be defined as a sequence of transitions between related configurations, sequence which can be infinite or ended with a terminal configuration. The transitions are the events associated with processes, events that don't affect only the states of the processes, but also they influence and can be influenced by the queue of messages.

The transitions systems allow modeling the transmission of messages between distributed processes, in two possible ways: synchronous and asynchronous transmissions. We will describe each of the two models textually and formally.

Transition systems with asynchronous message passing. In the asynchronous messages transmission, each *send* event and its corresponding *receive* event are independent. It means that a process initiates a *send* event, without waiting

for a process to receive the produced message. On the other hand, a process initiates a *receive* event, without knowing which process produced the message (see [1,3,7]).

We consider a collection of processes $P = (p_1, p_2, ..., p_N)$ which collaborate using a distributed algorithm A.

Definition 2.4. *The* transition system *induced by the distributed algorithm, using a communication model based on* asynchronous message passing, *is the system* $T = (C, \rightarrow, I)$*, constructed according to the following steps:*

(0) *The local algorithms are:* $p_i = (S_{p_i}, I_{p_i}, >^I_{p_i}, >^S_{p_i}, >^R_{p_i})$
(1) *The set of configurations are N-tuples of the following form:*

$$C = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M')$$

, where $\forall i \in \{1, ..., N\}, s_{p_i} \in S_{p_i}, M' \subseteq M$
(2) *The transition relation "* $\rightarrow$ *" is defined as in the general case of transition systems.*
(3) *The set of initial configurations I is composed of* $N + 1$*-tuples of the form:* $I = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, M')$*, where* $s_{p_I} \in I_{p_i} \forall i \in \{1, ..., N\}$*. The message queue* M' *is empty.*

In this model, only the process which initiates the *send* event, respectively the process which initiated the *receive* event changes its state.

Transition systems with synchronous message passing. In the synchronous message passing, each *send* event and its corresponding *receive* event are coordinated so as they form a single transaction of the system. In this scenario a process can transmit a message only if the destination process is ready to accept the message (see [4,5,7]).

This transition systems model also uses a collection of processes $P = (p_1, p_2, ..., p_N)$ which collaborate using a distributed algorithm A.

Definition 2.5. *The transition system induced by the distributed algorithm, using a communication model based on synchronous message passing, is the system* $T = (C, \rightarrow, I)$*, constructed according to the following steps:*

(0) *The local algorithms are:* $p_i = (S_{p_i}, I_{p_i}, >^I_{p_i}, >^S_{p_i}, >^R_{p_i})$
(1) *The set of configurations are N-tuples of the following form:*

$$C = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N}),$$

where $\forall i \in \{1, ..., N\}, s_{p_i} \in S_{p_i}$
(2) *The transition relation "* $\rightarrow$ *" is defined as a reunion of the transition relations "* $\rightarrow_{p_i}$ *" and "* $\rightarrow_{p_i,p_j}$ *",* $\forall i, j \in \{1, ..., N\}$*.*
(3) *The set of initial configurations I is composed of N-tuples of the form:* $I = (s_{p_0}, ..., s_{p_i}, ..., s_{p_N})$*, where* $s_{p_i} \in I_{p_i} \forall i \in \{1, ..., N\}$

In the configuration obtained after processing the transition of the system, both the process which initiated the *send* event and the process which got the *receive* event, change their states.

3. Transition systems with half-synchronous message passing

3.1. **Overview.** This paper proposes a new communication model, which represents an intermediary class between the synchronous and asynchronous communication models. We call this new model *half-synchronized transition system* and we will describe it using both a formal and a textual definition.

This new communication protocol also uses a transition system as the underlying modeling instrument, but the model imposes some constraints on the transition system's entities.

Thus, the message queue M is considered to be limited in space and time. The limitation in space means that the queue has a limited size, which coincides with the maximum number of messages that can be queued.

On the other hand, the limitation in time imposes that a message can be kept in the queue only for a certain period of time.

We consider that the process, which initiated the *send* event, establishes this period of time, because, as a "creator" of the message, it knows how important the message is. If the message is not consumed, during the given period of time, by another process, the message is automatically destroyed by a manager process, associated with the queue.

We represent the messages queue as a 4-tuple: (M, ds, dm, mq), where dm is the maximum size of the queue, ds is the current size of the queue ($dm \geq ds$), and M contains the messages: $M = \{m_1, m_2, ..., m_{ds}\}$. Each message is characterized by a "type" id, a "content" ct and a maximum period of life-time t, $\forall i \in \{1, ..., N\}$, $m_i = (id_i, ct_i, t_i)$. The receiving processes can retrieve the expected messages, using the type information, which could abstract different characteristics of a message, as well as its source or its destinations.

If the first three components of the queue are passive entities, the fourth element, noted with qm (queue manager) is an active process, which controls events like sending or receiving messages, as well as the life- time of the messages (destroying them, when their life-time period expires).

The queue manager have 3 possible states:

- $\mathbf{qm_w}$ waiting for a connection of client process (a sending or receiving process)
- $\mathbf{qm_r}$ waken-up by a request of a client process
- $\mathbf{qm_e}$ processing a request of a process or destroying expired messages.

The main difference between the half-synchronized and the previous two models, is that the functionality of the new model comprises the interaction between

the distributed processes and the queue manager and the activity of the queue manager.

Another important difference of this new model, compared with the preceding two, concerns the events of the communication subsystem. Thus, the *send* and *receive* events presume new scenarios and new events like *connect, acknowledge, wait, store, retrieve* and *destroy* are added. These events concern either the interaction between the distributed processes and the queue manager or the activity of the queue manager. They are also associated with binary relations between related configurations of the system.

3.2. **Events describing the interaction between the distributed processes and the queue manager.**

Definition 3.1. *The relation "$\rightarrow^{C}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_w)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r))),$$

where the process p_i is executing a connect *event to the message queue: $\exists m \in M$ of type connect-send or connect-receive, $(s_{p_i}, m, u_{p_i}) \in >^S$ and $(qm_w, qm_r) \in >^I$.*

Definition 3.2. *The relation "$\rightarrow^{A}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e))),$$

where the queue manager qm acknowledges the process p_i that it can initiate a send *or* receive *event: $\exists m \in M$ of type acknowledge, $(qm_r, m, qm_e) \in >^S$ and $(s_{p_i}, m, u_{p_i}) \in >^R$.*

Definition 3.3. *The relation "$\rightarrow^{W} p_i$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_r)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_w))),$$

where the queue manager qm execute a wait event, announcing the process p_i that the queue is full: $\exists m \in M$ of type wait, $(qm_r, m, qm_w) \in >^S$ and $(sp_i, m, up_i) \in >^R$.

Definition 3.4. *The relation "$\rightarrow^{S}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm'_e))),$$

where the process p_i initiates a send *event with the message m and qm starts processing it: $\exists m \in M$, $(s_{p_i}, m, u_{p_i}) \in >^S$, $(qm_p, m, qm'_e) \in >^R$.*

Definition 3.5. *The relation "$\rightarrow^{R}_{p_i}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)), (s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_2, ds_2, dm, qm'_e))),$$

where the process p_i initiates a receive *event with the message m and qm starts processing it: $\exists m \in M_1$, $(qm_p, m, qm'_e) \in >^S$ and $(s_{p_i}, m, u_{p_i}) \in >^R$.*

3.3. **Events describing the activity of the queue manager.**

Definition 3.6. *The relation "$\rightarrow^{S}_{qm}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0},...,s_{p_i},...,s_{p_N},(M_1,ds,dm,qm_e)),(s_{p_0},...,s_{p_i},...,s_{p_N},(M_2,ds{+}1,dm,qm_w))),$$

where qm store the received message m in the queue: $M_2 - \{m\} = M_1$ and $(qm_e, qm_w) \in >^I$.

Definition 3.7. *The relation "$\rightarrow^{R}_{qm}$" is defined as the set of the following pairs of configurations:*

$$((s_{p_0},...,s_{p_i},...,s_{p_N},(M_1,ds,dm,qm_e)),(s_{p_0},...,s_{p_i},...,s_{p_N},(M_2,ds{-}1,dm,qm_w))),$$

where qm retrieves the expected message m from the queue: $M_1 - \{m\} = M_2$ and $(qm_e, qm_w) \in >^I$.

Definition 3.8. *The relation "$\rightarrow^{D}_{qm}$ is defined as the set of the following pairs of configurations:*

$$((s_{p_0},...,s_{p_i},...,s_{p_N},(M_1,ds,dm,qm_e)),(s_{p_0},...,s_{p_i},...,s_{p_N},(M_2,ds{-}1,dm,qm_w))),$$

where qm destroys a message $m \in M_1$, whose life-time expired ($m = (id, ct, t)$ and $t = 0$): $M_1 - \{m\} = M_2 and (qm_e, qm_w) \in >^I$.

Now, we are ready to define the half-synchronized transition systems.

3.4. **Definition of the half-synchronized transition system.** As in the previous models, we also consider a collection of processes $P = (p_1, p_2, ..., p_N)$ which collaborate using a distributed algorithm A.

Definition 3.9. *The transition system induced by the distributed algorithm A, using a communication model based on half-synchronous message passing, called* half-synchronous transition system *is the system $S = (C, \rightarrow, I)$, constructed according to the following steps:*

(0) *The local algorithms are: $p_i = (S_{p_i}, I_{p_i}, >^I_{p_i}, >^S_{p_i}, >^R_{p_i})$*

(1) *The set of configurations are N-tuples of the following form:*

$$C = (s_{p_0},...,s_{p_i},...,s_{p_N},(M',ds,dm)),$$

where $\forall i \in \{1,...,N\}$, $s_{p_i} \in S_{p_i}$, $M' \subseteq M$, where the messages queue was defined above

(2) *The transition relation "$\rightarrow$" is defined as a reunion of a transition relations "$\rightarrow^{C}_{p_i}$", "$\rightarrow^{A}_{p_i}$", "$\rightarrow^{W}_{p_i}$" "$\rightarrow^{S}_{P_i}$", "$\rightarrow^{R}_{p_i}$", "$\rightarrow^{S}_{qm}$", "$\rightarrow^{R}_{qm}$", "$\rightarrow^{D}_{qm}$", associated to the corresponding events, which comprise the functionality of the system.*

3.5. **Space and time limitations scenarios.** The following scenarios describe the half-communication model defined above:

Time limitation.

1.

$(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_0}, ..., v_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1 \cup \{m = (id, ct, t)\}, ds + 1, dm, qm_w))$
and
$(s_{p_0}, ..., s_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm_w)) \rightarrow^{C}_{p_j}$
$(s_{p_0}, ..., u_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_0}, ..., v_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm_e)) \rightarrow^{R}_{p_i}$
$(s_{p_0}, ..., v_{p_j}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t')\}, ds_2, dm, qm'_e)) \rightarrow^{R}_{qm}$
$(s_{p_0}, ..., z_{p_j}, ..., s_{p_N}, (M_2, ds_2 - 1, dm, qm_w))$ and $t' < t$,
or

2.

$(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(s_{p_0}, ..., u_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_0}, ..., v_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(s_{p_0}, ..., z_{p_i}, ..., s_{p_N}, (M_1 \cup \{m = (id, ct, t)\}, ds + 1, dm, qm_w))$
and
$(s_{p_0}, ..., s_{p_N}, (M_2 \cup \{m = (id, ct, t'), t' = 0\}, ds_2, dm, qm_e)) \rightarrow^{D}_{qm}$
$(s_{p_0}, ..., s_{p_N}, (M_2, ds_2 - 1, dm, qm_w))$.

The first scenario refers to the case where a *send* event has a corresponding *receive* event with the same message $m \in M$.

In the following, we will enumerate the sequence of transitions, described formally above. The process p_i initiates a *connect* event to the queue, the queue manager returns an *acknowledge* that the queue can store new messages, the process p_i *sends* its message m (whose life-time is t) and continues its job and the queue manager *stores* the received message in the queue. Later ($t' < t$), a process p_j initiates a *connect* event, and after being *acknowledged* by the queue manager, it initiates a *receive* event with the message m, and the queue manager retrieves this message from the queue.

The second scenario differs from the first, in that the message m is not retrieved from the queue for a receiving process, hence the queue manger destroys the message, when its life-time period expires.

Space limitation.

$(s_{p_0}, ..., s_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(s_{p_0}, ..., u_{p_i}, s_{p_j}, ..., s_{p_N}, (M_1, dm, dm, qm_r)) \rightarrow^{W}_{p_i}$
$(s_{p_0}, ..., v_{p_i}, ..., s_{p_N}, (M_1, dm, dm, qm_w))$

The space limitation scenario is similar to the producer-consumer problem. A process p_i tries to *connect* to the queue in order to receive the message m, but the queue manager initiates a *wait* event, telling the process, that the queue is full. So, the process waits until some free space will be available in the queue.

3.6. **An example.** We give an example of a real problem, which can be solved using the half-synchronous model. The problem concerns the activity -supplying-storing-selling merchandise- of a store of perishable nutriments.

In order to give a formal model to this problem, we consider only two processes: $P = (p_1, p_2)$, where p_1 is a "supplier" and p_2 is a "buyer". The messages set M contains the products $prod_1$, $prod_2$, ... (for simplicity, we consider that $prod_i$ includes a given quantity of the product i) and a subset M' of internal, protocol messages. Hence, $M = \{prod_1, prod_2, ...\} \cup M'$. The configurations, used in this particular model, take the following form: $(s_{p_1}, s_{p_2}, (M_1, dc, dm, qm))$, where s_{p_1} and s_{p_2} are the current states of the "supplier", respectively "buyer" processes, M_1 contains the products from the store, dc is the number of the products and dm is the maximum capacity of the store and qm is the manager & seller of the store.

Now, let's detail the two scenarios starting from the general case, for the "store model":

Time limitation

1. $(s_{p_1}, s_{p_2}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(u_{p_1}, s_{p_2}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(v_{p_1}, s_{p_2}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(z_{p_1}, s_{p_2}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(z_{p_1}, s_{p_2}(M_1 \cup \{m = (id, ct, 20d)\}, ds + 1, dm, qm_w))$
and $(s_{p_1}, s_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm_w)) \rightarrow^{C}_{p_j}$
$(s_{p_1}, u_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(s_{p_1}, v_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm_e)) \rightarrow^{R}_{p_i}$
$(s_{p_1}, v_{p_2}, (M_2 \cup \{m = (id, ct, 10d)\}, ds_2, dm, qm'_e)) \rightarrow^{R}_{qm}$
$(s_{p_1}, z_{p_2}, (M_2, ds_2 - 1, dm, qm_w))$ and $t' < t$,
or
2. $(s_{p_1}, s_{p_2}, (M_1, ds, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(u_{p_1}, s_{p_2}, (M_1, ds, dm, qm_r)) \rightarrow^{A}_{p_i}$
$(v_{p_1}, s_{p_2}, (M_1, ds, dm, qm_e)) \rightarrow^{S}_{p_i}$
$(z_{p_1}, s_{p_2}, (M_1, ds, dm, qm'_e)) \rightarrow^{S}_{qm}$
$(z_{p_1}, s_{p_2}, (M_1 \cup \{m = (id, ct, 20d)\}, ds + 1, dm, qm_w))$
and
$(z_{p_1}, s_{p_2}, (M_2 \cup \{m = (id, ct, 0))\}, ds_2, dm, qm_e)) \rightarrow^{D}_{qm}$
$(z_{p_1}, s_{p_2}, (M_2, ds_2 - 1, dm, qm_w))$.

The first scenario corresponds to the situation when a provided product m is bought, before its life-time period expires. The supplier offers a product m to the store, which expires after $t = 20$ days. The communication protocol (connect-acknowledge-send-store, respectively connect-acknowledge-receive- retrieve), between the supplier p_1 or the buyer p_2 and the store manager qm, conforms to the general case, but with particular participants.

The second scenario models the situation when the availability period for the product m expires before anyone have bought it. So, the store manager throws away this product.

Space limitation

$(s_{p_1}, s_{p_2}, (M_1, dm, dm, qm_w)) \rightarrow^{C}_{p_i}$
$(u_{p_1}, s_{p_2}, (M_1, dm, dm, qm_r)) \rightarrow^{W}_{p_i}$
$(v_{p_1}, s_{p_2}, (M_1, dm, dm, qm_w))$

Finally, if there is no more free space in the store, the supplier p_1 should wait until someone buys something from the store, in order to make its own offer.

As a final remark, we point out that, this particular model can be enhanced, in order to filter the products offered by the suppliers. In this case, the store manager decides if a provided product suits its needs.

4. Conclusion

The communication subsystem constitutes the main part of any distributed system. The protocols models, used to describe the functionality of the communication software, should be flexible, reusable, adaptable. In the first part, the paper presents the two most important possibilities *synchronous* and *asynchronous*-, of modeling communication between distributed processes, using transitions systems. In the second part, the paper extends the models mentioned above and proposes an intermediary class of models, called *half-synchronized transition systems*, starting from some limitations, in time and space, imposed on the entities involved.

References

[1] Andrews G.R. *Synchronizing Resources*, ACM Transactions on Computer Systems, Number 4, October, 1981, pp. 305-330
[2] Boian F.M., *Programare distribuită în Internet*, Editura Albastra, 1998
[3] Fred B. Schneider, *Synchronization in Distributed Programs*, ACM Transactions on Computer Systems, Volume 4, Number 2, April, 1992, pp. 125-148
[4] Lamport L., *Time, clocks, and the ordering of events in a distributed system*, Communications of the ACM, 21, 7, july 1978, pp. 125-133
[5] Spirakis G.P. *Real-Time Synchronization of Interprocess Communications*, ACM Transactions on Computer Systems, Volume 6, Number 2, April, 1994 pp. 215-238
[6] Tannenbaum A.S. *Distributed Operating Systems*, Prentice Hall, 1995
[7] Tel G. *Introduction to Distributed Algorithms*, Cambridge Press, 1994

"Babeş-Bolyai" University of Cluj-Napoca, Department of Computer Science
E-mail address: {florin,cori}@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

AN EVOLUTIONARY ALGORITHM FOR THEOREM PROVING IN PROPOSITIONAL LOGIC

D. DUMITRESCU AND M. OLTEAN

Abstract. Standard theorem proving algorithms normally have exponential complexity. This drawback could be eliminated within the Evolutionary computation framework. In this paper an evolutionary theorem proving method is proposed. The method is designed for propositional logic but may be extended to first order logic.

The proposed evolutionary approach represents a new paradigm of automated theorem proving. Method complexity is polynomial. Reducing time complexity with respect to non-evolutionary methods is an important feature of the proposed approach.

A population of theorems is evoluated using two search operators: recombination and mutation. Recombination operator implements Modus Ponens inference rule. Mutation corresponds to the substitution operation.

1. Introduction

It is well known (see [3, 4, 5]) that the problem of deciding a given formula in propositional logic is (or not) a theorem in an NP problem. As we do not know if there exists a polynomial algorithm for solving NP problems they are considered as hard problems.

Many algorithms for automated theorem proving have been proposed. Robinsons resolution method ([7]) is one of the most powerful methods. However, resolution has the great disadvantage of being NP complete. This means that on a standard (sequential) computer the algorithms run a time that is bound by an exponential function of the input sequence length.

Evolutionary Algorithms (EAs) (see [1, 2]) are useful tools for solving complex optimization and search problems. EAs can also be successfully used to solve NP problems.

In this paper we propose an evolutionary algorithm, called ETP, for automated theorem proving within the propositional logic.

A model of propositional calculus consisting from three axioms and two inference rules is considered. The inference rules of the model are Modus Ponens (MP) and substitution. Our goal is to determine if a given well-formed formula R is (or it is not) a theorem. To solve this problem the ETP algorithm will be used.

A substitution operation by which every variable from an axiom is replaced by a well-formed formula is used to obtain theorems from axioms. The replacing formulae in a substitution are generally different. In what follows by a *substitution* we designate the formulae used for replacement in an axiom or in a theorem.

2. Evolutionary model for theorem proving

Within a generation t a population $S(t)$ of substitutions is used to obtain theorems from the system axioms. These substitutions may remain constant during the search process or may be evoluated by an evolutionary procedure. In the first case we have:

$$S(t) = S, t = 1, 2, ...,$$

where t is the generation index.

Each substitution population $S(t)$ generates a theorem population $T(t)$.

Theorems from $T(t)$ will be modified using variation operators. In our model considered variation operators are recombination and substitution. Recombination operator corresponds to the application of Modus Ponens rule and substitution plays the role of mutation operator.

Using MP from two parent theorems an offspring is obtained. As usual parent recombination is guided by a fitness function.

For a given theorem T all the possible mating candidates are established. A candidate partner TP of T has to have the form

$$TP = T \rightarrow A,$$

where A is a well formed formula.

Let us note that the Modus Ponens rule can be applied only for pairs of theorems having a particular form.

Tournament selection, or other selection procedures may be used to find the mating partner of T. Let T' be the tournament winner. From T and T' an offspring is obtained applying MP inference rule. Parent T' is considered to be dominant.

Within survival dominant parent is compared with its offspring. The winner will enter the new generation.

3. Evolutionary algorithm

Let R be a formula representing a target (or tentative) theorem. Our aim is to decide if R is a theorem or it is not.

Evolutionary theorem proving (ETP) algorithm starts with an arbitrary population of substitutions. At the first step we randomly make substitutions in the axioms. A *substitution* operator realizes this operation. Performing substitution operation an initial population of theorems is obtained.

In the next phase the theorem population is evolved. New theorems are obtained using MP inference rule. The obtained theorems are modified via a mutation operator (i.e. by performing substitutions in theorems).

As MP rule implies two theorems, we can assimilate MP inference rule with recombination operator. Theorems obtained by recombination are subsequently modified by the effect of substitution operator.

After a number of generations the algorithm stops with the answer "Yes" or "No". Yes means that that the target formula R exists in the last theorem population. Therefore R is a theorem. No means either that the formula is not a theorem, or our algorithm fails to determine it. The algorithm may fail to prove either due to a bad chosen population of substitutions, or due to an incomplete exploration of solution space by search operators.

If the algorithm result is Yes then the deduction chain representing the proof of theorem R may be established.

When the output of the procedure is No then we may restart the algorithm trying to prove the theorem T, where

$$T =\sim R.$$

If the new output is Yes then definitively R is not a theorem. Otherwise we can not make a decision about the truth value of the assertion R is a theorem.

4. Individual representation

Each well-formed formula of propositional logic can be represented as a tree. The non-terminal nodes contain logical connectives and the terminal ones contain the propositional variables. Each node has maximum two descendants. If the node contains the connective "$\rightarrow$" then the node has two descendents. If the node contains the negation connective "$\sim$" then it has to have just one descendent. If a node contains a propositional variable, then it is a leaf and it has no descendents at all.

In our approach axioms, substitutions and theorems will be represented as trees. Therefore the search space for our problem is the set of trees describing well-formed formulae. This representation is similar to that used in Genetic Programming (see [6]).

To limit the dimension of the search space we have to reduce the tree depth and the number of propositional variables used in substitutions. However, the number of the propositional variables in a substitution must be higher than the number of propositional variables in the target theorem R.

Denote by $h(R)$ the target theorem height.

As by recombination the height of the trees decrease, the height of a substitution tree must be no less than $h(R) - 1$. Therefore we have

$$h(s) + 2 \geq h(R),$$

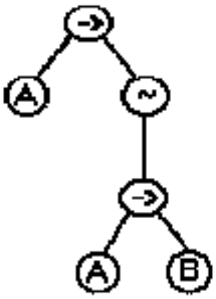


Figure 1 Tree for formula $(A \rightarrow\sim (B \rightarrow A))$

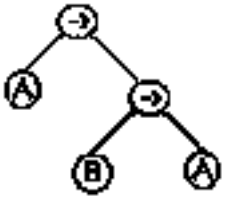


Figure 2 Tree for axiom A1 $A \rightarrow (B \rightarrow A)$

for each substitution s. (We added two because the axioms have the height one or two.)

Example. Let us consider the formula $F = (A \rightarrow\sim (B \rightarrow A))$. The tree corresponding to formula F is depicted in Figure 1.

Let us denote by A the system axioms. We may interpret each axiom as a potential solution of the problem (an individual in the search space).

The system axioms are:

A1:: $A \rightarrow (B \rightarrow A)$.
A2:: $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.
A3:: $(A \rightarrow B) \rightarrow (\sim B \rightarrow\sim A)$.

The three system axioms are represented by the trees as depicted in Figures 2-4.

We may assume the set A contains three particular candidate solutions (or individuals) corresponding to the three axioms from propositional logic. We may consider A as a static solution population.

Moreover two different evolving populations S and T will be considered where:

- S is a *substitution population.* It contains some randomly generated formulae.

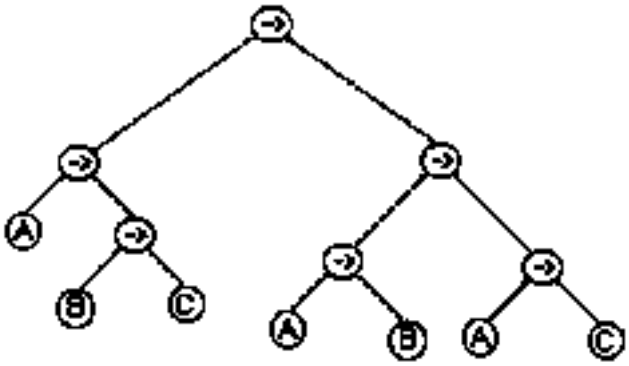


Figure 3 Tree for axiom A2 $(A \to (B \to C)) \to ((A \to B) \to (A \to C)))$

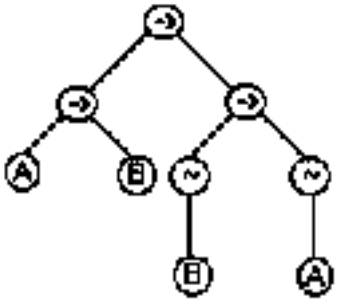


Figure 4 Tree for axiom A3 $(A \to B) \to (\sim B \to \sim A)$

- T is a *theorem population.* Each individual in this population is a theorem. Each population member will be obtained by a substitution or by a crossover operation performed on another two theorems.

Remark. By an axiom, substitution or theorem we will generally mean the corresponding trees representation.

5. Selection and Search Operators

We use two search (variation) genetic operators: MP recombination and substitution. Let us observe that the substitution operator may be interpreted as a special, problem-dependent type of mutation.

5.1. **Selection.** Each member of theorem population is selected for recombination. Each individual x in the theorem population is considered as a *recessive* parent. The corresponding *dominant* parent is chosen from the possible candidates using a tournament selection procedure.

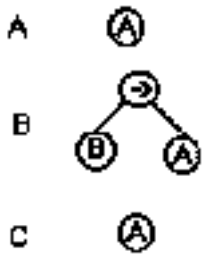


Figure 5 Substitution S1

5.2. **MP Recombination Operator.** MP recombination operator implements the Modus Ponens rule. We recall that Modus Ponens may be expressed as

$$A, A \to B \vdash B$$

Let us consider two theorems A and $A- > B$ represented as trees. Performing crossover (which corresponds to Modus Ponens rule) the offspring B is obtained. Of course B it is also represented as a tree.

From a biological point of view we may consider the first parent (theorem A) as the recessive parent. This interpretation is justified because no part of the first parent is present in the offspring. The second parent (namely $A \to B$) is the dominant one. The offspring is a part of its dominant parent.

5.3. **Substitution Operator.** Substitution (or mutation) operator implements the substitution inference rule. Substitution combines an individual from the substitution population with an individual from the axiom set or from the theorem population. The obtained offspring is a theorem and it may be added to the theorem population.

5.4. **Survival.** Several survival mechanisms may be considered (see [1]). Some survival strategies are generational and some are steady-state. In this paper we consider a steady-state survival mechanism. Each offspring is compared with its dominant parent. The best from the parent and offspring will become a member of the new population.

6. Example

Let us consider a substitution S_1 where propositional variables are replaced as follows:

- A: is replaced by the formula A.
- B: is replaced by the formula $B \to A$.
- C: is replaced by A.

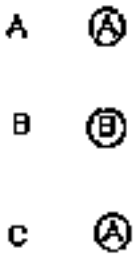


Figure 6 Substitution S2

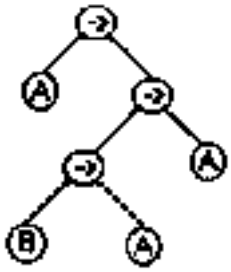


Figure 7 Tree of theorem T1

The trees describing substitution S_1 are represented in Figure 5.

Consider also a substitution S_2 specified as follows:

- A: is replaced by the formula A.
- B: is replaced by the formula B
- C: is replaced by A.

The trees describing substitution S_2 are depicted in Figure 6.

Let us consider the target theorem R is

$$R : A \to A.$$

Our aim is to prove the target theorem R using the substitution S_1 and S_2 only. In this respect S_1 and S_2 will be applied to the axioms.

By using substitution S_1 in axiom A we obtain the theorem T_1. The tree representing this theorem is depicted in Figure 7.

By using S_1 and A_2 we obtain a theorem T_2. The corresponding tree is depicted in Figure 8.

By using substitution S_2 in axiom A_1 we obtain a theorem T_3. The corresponding tree is depicted in Figure 9.

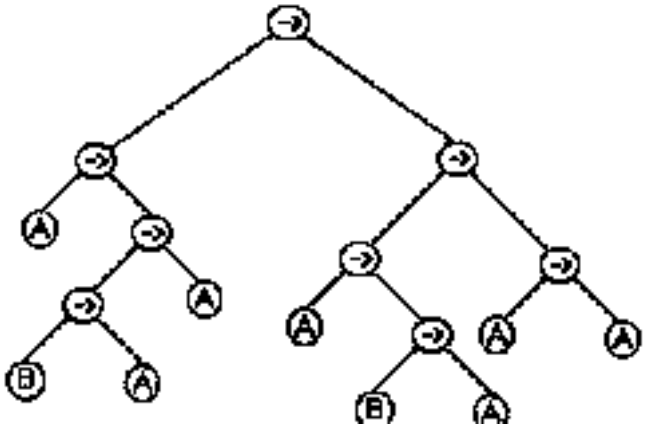


Figure 8 Tree of theorem T2

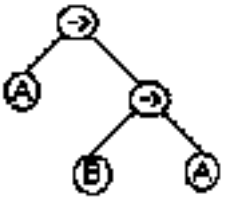


Figure 9 Tree of theorem T3

MP recombination of theorems T_2 and T_1 produces the offspring theorem T_4. The process is depicted in Figure 10.

By MP recombination of theorems T_3 and T_4 we obtain the target theorem R. The corresponding recombination process is depicted in Figure 11.

Therefore we have obtained a complete proof of theorem R.

7. Fitness Function

From the previous example we may observe that the target theorem R can be proved if and only if in the theorem population arises an individual T such that R may be reached from the root of the tree T by following the right sub-trees of T only.

Fitness of a theorem T may be defined by using the tree representing this theorem.

We may define the *handle* $H(T)$ of a theorem T by the distance from the root node (of the tree representing T) to the right sub-tree representing the target theorem R. Our aim is to minimize the theorem handle.

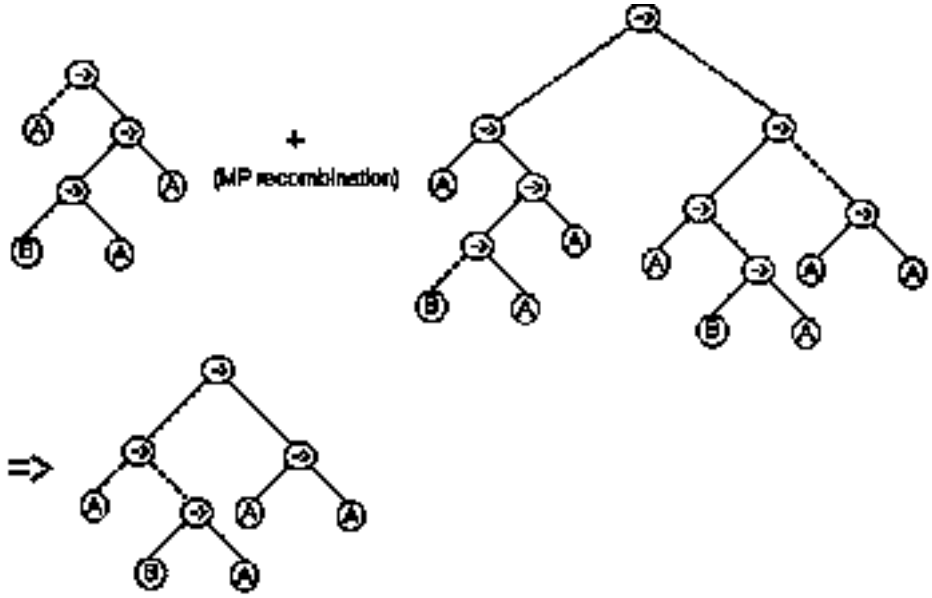


Figure 10 MP recombination of theorems T1 (recesive) and T2 (dominant)

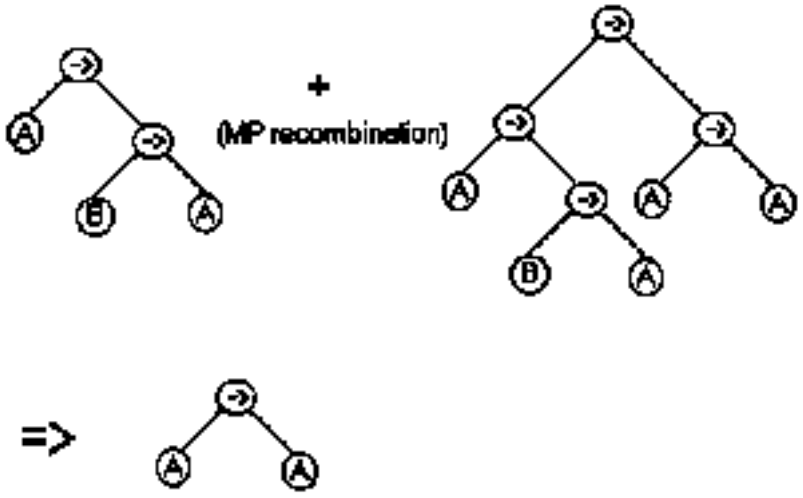


Figure 11 MP recombination of theorems T3 (recesive) and T4 (dominant)

Fitness $f(T)$ of a theorem T may be defined as

$$f(T) = \frac{1}{H(T)+1}.$$

The fitness is to be maximized.

The fitness of a theorem that does not contains the target theorem R as a right (sub)sub-tree is considered to be zero.

Examples. For the theorem T_1 represented in Figure 10 has the handle

$$H(T_1) = 2.$$

The fitness of T_1 is

$$f(T_1) = 1/3.$$

For the theorem T_4 (the offspring depicted in Figure 10) has the handle

$$H(T_4) = 1.$$

The fitness of T_4 is

$$f(T_4) = 1/2.$$

For the target theorem R has the handle

$$H(R) = 0.$$

The fitness of $R$is

$$f(R) = 1.$$

For the theorems T_2 (Figure 8) and T_3 (Figure 9) has the handle

$$H(T_2) = H(T_3) = \infty.$$

The fitness of T_2 and T_3 is

$$f(T_2) = f(T_3) = 0.$$

8. ETP procedure

In this section an evolutionary theorem proving method based on previous considerations rule is presented. The method uses a function ETP_F that evolves a population of candidate theorems. If the function output is Yes i.e. R is a theorem then the ETP algorithm stops. Otherwise we try to prove the formula $\sim R$ is a theorem. For this respect function ETP_F is used again.

Evolutionary theorem proving (ETP) algorithm may be outlined as below:

ETP ALGORITHM

```
begin
    if ETP_F(R) = Yes    {R is a theorem}
    then print deduction chain;
    else if ETP_F(~ R) = Yes   {R is not a theorem}
        then print R is not a theorem
        else   {we can not say if R is or is not a theorem}
            print we can not make a decision about R
        endif
    endif
end
```

Function ETP_F is outlined as follows:

```
function ETP_F(R theorem);    {returns Yes if R is theorem otherwise return No}
begin
    Initialization:
    generate randomly a substitution population (S);
    generate a theorem population T(t) using substitutions from S;
    {apply substitutions from S}
    Evolving theorems:
    t = 0;
    while t <= MaxGen do
        for each individual c in T(t)
            find b - the best mate for c;
            z = crossover(b, c);
            if fitness(z) > fitness(c)
            then discard c from the theorem population T(t);
                add z to the theorem population T(t)
            endif
        endfor
        Mutate chromosomes from the current theorem population T(t);
        t = t + 1;
    endwhile
end
```

9. Concluding remarks and further research

A new evolutionary approach of automated theorem proving is proposed. The method is designed for propositional logic. This algorithm illustrates a new philosophy of theorem proving: According to our knowledge a similar approach does not exist in the literature.

We consider that using the proposed method some well-known drawbacks of classical method may be eliminated.

Numerical experiments have shown the effectiveness of the proposed algorithm. The method uses a heuristic fitness function.

The proposed approach will be extended for the first order logic.

References

[1] Bck, T., Fogel, D.B., Michalewicz, Z. (Eds.), (1997), Handbook of Evolutionary Computation, Institute of Physics Publishing, Bristol, and Oxford University Press, New York.

[2] Dumitrescu, D., Lazzerini, B., Jain, L.C., Dumitrescu, A., (2000) Evolutionary Computation, CRC Press, Boca Raton, FL.

[3] Fitting, M., (1990), First-Order Logic and Automated Theorem Proving, Springer-Verlag, New- York.

[4] Gallier, J.H., (1986), Logic for Computer Science, Foundation of Automatic Theorem Proving, Harper and Row.

[5] Garey, M.R., Johnson, D.S., (1978), Computers and Intractability: A Guide to NP- completeness, W.H. Freeman and Company, New York.

[6] Koza, J.R., (1992), Genetic Programming, MIT Press, Cambridge, MA.

[7] Robinson, J.A., (1965), A Machine-Oriented Logic Based on the Resolution Principle, Journal of ACM, vol 12, pp 23-41.

Acknowledgments

The authors thank the anonymous reviewer for pertinent observations.

"Babeş-Bolyai" University of Cluj-Napoca, Department of Computer Science
E-mail address: {ddumitr,moltean}@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

SEMANTIC REPRESENTATION OF THE QUANTITATIVE NATURAL LANGUAGE SENTENCES

ADRIAN ONET AND DOINA TATAR

Abstract. In this paper we propose the description of two methods of quantitative natural language sentences representation with the use of the first order logic. Both of these methods extend the classic first order logic. One of these two methods was introduced by J. Allen [4] who extends the classic logic by admitting sets as objects (method 2). The other method, more restrictive, (introduced by the authors of this paper) extends the quantifier semnification - specially the existential one - thus the quantifier would be able to express exact quantities (method 1). This method is a specialization of Allen's method.

In the second part of this paper we describe an implementation of the first method by the lambda calculus and also the transformation of these expressions in first order formulas using Prolog predicates.

1. First order predicate logic as representation language for quantitative natural language sentences

Among the most used methods in natural language representation is the classic first order predicate logic, but unfortunately not all natural language sentences can be represented using the classic first order predicate logic. This is the case of quantitative sentences. For instance, a sentence like "Three men entered the room" is quite difficult to represent by using the first order logic, because there are more quantifiers in natural language that the universal and existential quantifiers. A representation like:

$$\exists X.(men(X) \wedge (\exists Y.room(Y) \wedge enter(X,Y))$$

would not mean the same thing semantically speaking, there is no information in this formula concerning the number of persons which entered the room.

We will describe two methods of extension for the classic first order logic in order to allow the representation of the quantitative aspects in natural language. The first method is weaker (it does not allow the representation of all quantitative aspects) but its implementation is easier (one of its implementations is given in the second part of this paper), and a second method which extends the representation ontology in order to allow sets as objects.

1.1. **First method.** Let $L = (\Sigma, F, A, R)$ be a first order predicate logic [3]:

$$\Sigma = V \cup C \cup (\cup F_i) \cup (\cup P_i) \cup \{(,), \forall, \exists, \wedge, \vee, \rightarrow\}$$

where V represents a set of symbols called variables, C represents a set of symbols called constants, F_j represents a set of function symbols with the arity j, P_j represents a set of predicate symbols with the arity j, $\{\neg, (,), \wedge, \vee, \rightarrow\}$ represents logic operators and $\exists$ and $\forall$ represent the logic quantificators, F represents a set of well-formed formulas, A represents a set of axioms over L and R a set of rules over L.

We can extend this logic by introducing a pair (M, R) where M represents a set of numbers and R represents the order relation "$<$" over M. If E is the domain of the logic and $|E|$ is finite then M can be the set $\{1, ..., n\}$ where $|E| = n$. If E is infinite then we will create M such as $|M| = |E|$.

We introduce a new set of quantificators, $\{\exists_n | n \in M\}$ where $\exists_n x.(p(x))$ means that there exists exactly n x in E such that $p(x)$. If we introduce the exclusive or operator $\otimes(p \otimes q \leftrightarrow (\neg p \wedge q) \vee (p \wedge \neg q))$ we have:

$$\exists x.p(x) \leftrightarrow \otimes_{j \in M} \exists_j x.p(x),$$

and it can also be proved that $\exists_n x.p(x) \rightarrow \exists x.p(x)$ is a theorem $\forall n \in M$.

In addition, we introduce the following predicates to describe the relation between the elements of M:

$p < q \leftarrow R_{pq}$
$greater(p, q) \leftarrow \neg p < q$
$most(p) \leftarrow |E|/2 < n$
$some(n)$ iff $n \in M$

In actual discourse, the interpretation of the most will usually be relative to some previously defined context. The most predicate can be rewritten to represent the most concept in the natural language.

With this extension, a sentence like "Three men enter the room" would be interpreted like

$$\exists_3 X.(man(X) \wedge (\exists_1 Y.room(Y) \wedge enter(X, Y))),$$

where the first quantificator $\exists_3$ means that there is exactly 3 men who enter the room and the second quantificator $\exists_1$ means that the three men enter exactly one room.

With this extension of the first order logic we can also represent inconsistent knowledge expressed in natural language sentences. A sentence like "At least five men walk" has the following interpretation:

$$\exists_n X.(man(X) \wedge (walk(X)) \wedge greater(n, 3)).$$

The disadvantage of this method is that it cannot represent sentences that involve the interaction of two (or more) elements of the quantified set (for example

"Three men meet at four" – we will see that this kind of sentence can be represented using the second method).

Even with this disadvantage this method can be used successfully to represent quantitative sentences thanks to the easy implementation (see section 2).

1.2. **Second method.** This method has been introduced by J. Allen [4] and extends the ontology of the first order predicate logic to allow sets as objects. While sets in general may be finite (such as the set consisting of John and Mary) or infinite (such as the set o numbers greater then 7). In this method we will only use finite sets. We also allow constants to denote sets. Thus S_1 might be the set $S_1 = \{\text{John, Mary}\}$. The sets will be written in the form $\{y|P(y)\}$, which is the set of all object that satisfy the expression $P(y)$. The set of all men is $\{y|Man(y)\}$. In addition, we introduce the following predicates to relate sets and individuals:

$S1 \subset S2$ iff all elements of S_1 are in $S2$
$x \in S$ iff x is a member of the set S

With setlike objects in the representation, we can produce an interpretation for "Some men meet at four" as follows:

$$\exists M : (M \subset \{x|Man(x)\} \wedge Meet(M, 4PM))$$

that is, there is a subset of men M that met at three. In principle, sets are allowed in all situation where individuals have been allowed. In practice certain verbs require only sets or only individuals in certain argument positions. For example, the verb meet requires its agent to be a set with more than one element, as a single individual cannot meet. Other verbs require individuals and exclude sets, and others allow both sets and individuals as arguments.

Consider the different formulas that arise from the collective/distributive readings. There are two representation of the sentence "Some men bought a suit". The collective reading would map to:

$$\exists M1 : (M1 \subset \{z|Man(z)\} \wedge \exists_s : (Suit(s) \wedge Buy1(M1, s)))$$

that is there is a subset of the set of all men who together bought a suit. The distributive reading involves some men individually buying suits and would be represented by:

$$\exists M2 : (M2 \subset \{z|Man(z)\} \wedge \forall m : (m \in M2 \wedge \exists s : (Suit(s) \wedge Buy1(m, s))))$$

Note that with the first method described earlier we have:

$$\exists_n X.(man(X) \wedge (\exists_1 Y.(suit(Y) \wedge buy(X, Y)))),$$

which describes the distributive reading.

Note that the distributive and collective readings both involve a common core meaning involving the subset of men. The only difference is weather you use the set as a unit or quantify aver all members af the set.

The set-based representation can also be used to ensure that more that one man bought a suit. To do this we introduce a new function that returns the cardinality of set. For any given set S, let $|S|$ be the number of elements in S. Using arithmetic operators, we can now encode constraints on the size of sets. For example, the meaning of "Three men entered the room" would be as follows (with tense information omitted):

$$\exists M : ((M \subset \{y|man(y)\} \wedge |M| = 3) \wedge \forall m : (m \in M \wedge enter(m, room)))$$

by changing the restriction to $|M| \geq 3$, we get the meaning of "At least three men entered the room", and so on. More problematic quantifiers can also be given an approximate meaning using sets. For example we can use the most definition from the first method (i.e. most is true if more than half of some set has a given property), then "Most men smoke" might have the meaning:

$$\exists M : ((M \subset \{y|man(y)\} \wedge |M| \geq \frac{|\{y|man(y)\}|}{2}) \wedge \forall m : (m \in M \wedge smoke(m)))$$

In actual discourse, the interpretation of the quantified terms will usually be relative to some previously defined set. For example the sentence "Most men smoke" typically will refer to most of the men in a previously mentioned set rather than to most of the men in the world. In other words, the sentence would not claim that more than half of all men smoke, but that more than half the men in a certain context (say in a railway station) smoke.

If we compare the previous two methods, we can observe that the second method extends the first one, i.e. the first method works just at cardinality level of the sets. That is the first method specifies only those aspects which can be represented with the cardinality function in the second method, and the elements of the set doesn't interact with each other. For example the sentence "Mary eat two apples" can be represented using both methods, with the first method, its interpretation will be:

$$\exists_2 X.(apple(X) \wedge eat(mary, X),$$

using the second method the sentence will be mapped as follows:

$$\exists M : (M \subset \{y|apple(y)\} \wedge |M| = 2) \forall m : m \in M.eat(mary, m)$$

But the following sentence "Four men meet" have the following interpretation using the second method:

$$\exists M : (M \subset \{x|Men(x)\} \wedge |M| = 3) \wedge Meet(M),$$

but it can not be represented using the first method, because this method doesn't introduce the set concept. In the next section we introduce an automate process which associates semantic representation (first order predicate logic with the extension introduced by the first method) for quantitative natural language expression.

2. An automate process for associating semantic representation of quantitative natural language sentences

In this section we will use the lambda calculus as a national extension of first order logic with the extension discussed in the first method and then we will give an implementation of this lambda calculus in Prolog. In this section we use the notation given by P. Blackburn and J. Bos in [2,3].

2.1. **The Lambda Calculus.** The lambda calculus is a natural extension of the first order logic that allows us to bind variable using a new binding operator λ. Occurrences of variables bound by λ should be thought of as placeholders for missing information. An operation called β-conversion performs the required substitutions. The lambda operator marks missing information by binding variables. Here is a simple lambda expression:

$$\lambda x.man(x)$$

Here the prefix $\lambda x.$ binds the occurrence of x in $man(x)$. In this example, the binding of the free x variable in $man(x)$ explicitly indicates that man has an argument slot where we may perform substitutions.

Concatenation indicates that we wish to perform substitution. We're using a special symbol "@" to indicate concatenation. The following expresion:

$$\lambda x.man(x)@vincent,$$

yields $man(vincent)$.

The lambda expressions $\lambda x.man(x)$ and $\lambda y.man(y)$ are equivalent, all these expressions are functors which when applied to an argument, replace the bound variable by the argument. No matter which argument A we choose, the result of applying any of the two expression to A and then β-converting is $man(A)$. The process of relabeling bound variables is called α-conversion. Consider the following grammar:

$$\begin{array}{lcl} s & \rightarrow & np, vp. \\ np & \rightarrow & det, noun. \\ det & \rightarrow & 'every'. \\ noun & \rightarrow & 'men'. \\ vp & \rightarrow & 'cry'. \end{array}$$

Now we can build the semantic representation for our first sentence "Every man cries". For this purpose we assign lambda expressions to different basic syntactic categories, i.e.

$$\begin{array}{l} 'every' : \lambda P.\lambda Q.\forall x.(P@x \rightarrow Q@x) \\ 'man' : \lambda Y.man(Y) \\ 'cries' : \lambda X.cry(X) \end{array}$$

According to our grammar, a determiner and a common noun can combine to form a noun phrase. For our analysis we will associate the NP node with the

Figure 1

functional application that has the determiner representation as functor and the noun representation as argument, and then associate that with the verb phrase (VP). Using a graphic representation this will look as the Figure 1 presents.

Now let's take a look at how we represent quantitative sentences with lambda calculus.We will give three examples of DCG (which will bind four general situations) to do this transformation for particular quantitative sentences (according to the first method described in the previous section).

2.1.1. *Definite quantity sentences.* For the definite quantity sentence we will have to represent with the lambda calculus the $\exists_n$ quantifier. This quantifier is mapped by the following predicate: $exists(n, X, formula(X))$ where n represents the quantity (as it's described in the first method), X is a variable and $formula(X)$ is a formula which contains the variable X. The representation for the $\forall$ quantifier is $forall(X, formula(X))$, where X and $formula(X)$ have the same meaning as it was shown.

Lambda expressions will be represented in Prolog as follows: `lambda(L,F)`, where `L` is intended to be a Prolog variable, while `F` is a first order formula or the Prolog representation of a lambda expression. For example the lambda expression $\lambda A.men(A)$ will look in Prolog: `lambda(A,men(A))`. The concatenation will be represented in Prolog with the Prolog operator @ defined as:

```
:- op(950,yfx,@).
```

The indefinite sentences are given by the following determinants: one, two, three, four, ..., three and half, etc. Each of this determinant will be replaced by an expression $\exists_n$ where n represents the number denoted by the determinant. For example $\exists_{3.5}$ represents the determinant three and half. Let be the following sentence "Three men enter the room". In the following we describe a DCG who will take such sentences (which describe definite quantities) and will give us the

lambda expression corresponding to that sentence:

$$s(NP@VP) \rightarrow np(NP), vp(VP).$$
$$np(Det@Noun) \rightarrow det(Det), noun(Noun).$$
$$vp(NP@IV) \rightarrow iv(IV), np(NP).$$

with lexical entries

$$noun(lambda(A, man(A)) \rightarrow [men].$$
$$iv(lambda(B, lambda(C, enter(C, B))) \rightarrow [enter].$$
$$noun(lambda(C, room(C)) \rightarrow [room].$$
$$det(lambda(D, lambda(E, exists(3, X, D@X\&E@X)))) \rightarrow [three].$$
$$det(lambda(F, lambda(G, exists(1, X, D@X\&E@X)))) \rightarrow [the].$$

where $enter(A, B)$ express that A enter in B.

The construction of the first three lexical entries are obvious. Let's now have a look at the construction of the fourth clause:

$$det(lambda(D, lambda(E, exists(3, X, D@X\&E@X)))) \rightarrow [three].$$

These constructions yield that for any determinant which indicates a number, the lambda expression corresponds to $\exists_n$ quantifier where n represents that number (in our case n=3). In the last clause the determinat "the" denotes quantity equal to 1, so it is processed like its predecessor clause.

So if we have the following goal (in Prolog):

```
?-np(Sem,[three men enter the room],[]).
```

the `Sem` will be bounded to (we used bracket for a clear see):

```
Sem=(lambda(D,lambda(E,exists(3,X, D@X&E@X)))@lambda(A,man(A)) ) @
   (lambda(F,lambda(G,exists(1,X, F@X&G@X)))@lambda(C,room(C))) @
   lambda(C,lambda(B,enter(B,C)))
```

In last section we will give a method to apply the β-conversion to a lambda expression and transform to a first order formula. Next we will show this mechanism of β-conversion on the expression `Sem`.

```
1. Sem=(lambda(D,lambda(E,exists(3,X, D@X&E@X)))@lambda(A,man(A)) ) @
   ((lambda(F,lambda(G,exists(1,X, F@X&G@X)))@lambda(C,room(C))) @
   lambda(C,lambda(B,enter(B,C))))
2. Sem=(lambda(E,exists(3,X, lambda(A,man(A))@X&E@X))) @
   ((lambda(G,exists(1,X, lambda(C,room(C))@X&G@X))) @
   lambda(C,lambda(B,enter(B,C))))
3. Sem=(lambda(E,exists(3,X, man(X)&E@X))) @
   ((lambda(G,exists(1,X, room(X))&G@X))) @
   lambda(C,lambda(B,enter(B,C))))
4. Sem=(lambda(E,exists(3,X, man(X)&E@X))) @
   ((exists(1,X, room(X)& lambda(C,lambda(B,enter(B,C))@X))))
5. Sem=(lambda(E,exists(3,X, man(X)&E@X))) @
   ((exists(1,X, room(X)& lambda(B,enter(B,X))))))
   // with an ?-conversion we will transform X in Y
6. Sem=(lambda(E,exists(3,X, man(X))&E@X))) @
   ((exists(1,Y, room(Y)& lambda(B,enter(B,Y)))))
```

```
7. Sem=(exists(3,X, man(X)& (exists(1,Y, room(Y)& lambda(B,enter(B,Y))) @X)))
8. Sem=(exists(3,X, man(X)& (exists(1,Y, room(Y)& enter(X,Y)))
```

which represents the formula:

$$\exists_3 X.(man(X) \wedge \exists_1 Y.(room(Y) \wedge enter(X,Y)))$$

which is (in the Skolem normal form)

$$\exists_3 X.\exists_1 Y.(man(X) \wedge room(Y) \wedge enter(X,Y))$$

With this method we can represent sentences like "John eats one and half apple" and "The suit costs 400$". But it cannot represent sentences like "Two men look the same" and "Four kids fight with each other".

2.1.2. *Indefinite quantity sentences.* In this part we'll describe how indefinite quantity sentences can be represented using the lambda calculus. The indefinite sentences are given by the following determinants: most, number of, lot of ... For each of this determinant we must construct a special predicate like the most predicate which we defined in the previous section.

Let be the following sentence "Most people laugh". In the following we'll describe a DCG who will take such sentences (which describe indefinite quantities) and will give us the lambda expression corresponding to that sentence:

$$s(NP@VP) \rightarrow np(NP), vp(VP).$$
$$np(Det@Noun) \rightarrow det(Det), noun(Noun).$$
$$vp(V) \rightarrow v(V).$$

with the lexical entries

$$noun(lambda(A, people(A)) \rightarrow [people].$$
$$v(lambda(B, laugh(B))) \rightarrow [laugh].$$
$$det(lambda(C, lambda(D, exists(N, X, C@X\&D@X\&most(N))))) \rightarrow [most].$$

where the predicate *most* is defined in the previous section and $laugh(B)$ express that B laugh. Note that in this case (i.e. for indefinite quantity) we need a variable N instead of a constant. N will denote the number and the predicate most will tell us if that number is enough to represent the most concept.

So if we have the following goal (in Prolog):

```
?-np(Sem,[Most people laugh],[]).
```

the `Sem` variable will be bounded to:

```
Sem=( lambda(C,lambda(D,exists(N,X,C@X&D@X\&most(N))))@
                lambda(A,people(A))@lambda(B,laugh(B))
```

The β-conversion on the expression `Sem` will be:

```
1. Sem=(lambda(C,lambda(D,exists(N,X,C@X&D@X\&most(N))))@
                  lambda(A,people(A))@lambda(B,laugh(B))
2. Sem=(lambda(D,exists(N,X,people(X)&D@X&most(N))) @lambda(B,laugh(B))
3. Sem=(exists(N,X,people(X)&laugh(X)&most(N))
```

which represents the formula:

$$\exists_N X.(man(X) \wedge laugh(X) \wedge most(N))$$

With this method we can represent sentences like "A number of people reads", "A lot of apples are green", etc. But we cannot represent sentences like "Most people meet at three".

2.1.3. *Restrictive quantity sentences.* The restrictive sentences are given by the following determinants: at least, minimum, maximum, at the most. For each of this determinant we must construct a special predicate like the $least/2$ predicate which we defined as follows:

$$Least(N, M) \leftarrow N > M.$$

Let be the following sentence "At least four men cry". The DCG is:

$$s(NP@VP) \rightarrow np(NP), vp(VP).$$
$$np(Det@Noun) \rightarrow det(Det), noun(Noun).$$
$$det(NP@Numeral) \rightarrow np(NP), numeral(Numeral).$$
$$np(Prep@Noun) \rightarrow prep(Prep), noun(Noun).$$
$$vp(V) \rightarrow v(V).$$

with the lexical entries

$$noun(lambda(A, man(A)) \rightarrow [men].$$
$$v(lambda(B, cry(B))) \rightarrow [cry].$$
$$prep(lambda(C, C) \rightarrow [at]$$
$$noun(lambda(D, lambda(E, lambda(F, exists(N, X, E@X\&$$
$$F@X\&least(N, D))))))?[least]$$
$$numeral(lambda(G, G)@4) \rightarrow [four].$$

where $cry(B)$ expresses that B cry.

So if we have the following goal (in Prolog):

```
?-np(Sem,[At least four men cry],[]).
```

the `Sem` variable will be bounded to (we used bracket for a clear see):

```
Sem=(((lambda(C,C)@ lambda(D,lambda(E,lambda(F,exists(N,X,E@X&F@X&least(N,D))))))
   @ lambda(G,G)@4) @ lambda(A,man(A))) @ lambda(B,cry(B))
```

The β-conversion on the expression `Sem` will be:

```
1. Sem=((lambda(D,lambda(E,lambda(F,exists(N,X,E@X&F@X&least(N,D)))))
          @ lambda(G,G)@4) @ lambda(A,man(A))) @ lambda(B,cry(B))
2. Sem=((lambda(D,lambda(E,lambda(F,exists(N,X,E@X&F@X&least(N,D)))))
          @ 4) @ lambda(A,man(A))) @ lambda(B,cry(B))
3. Sem=(lambda(E,lambda(F,exists(N,X,E@X&F@X&least(N,4))))
           @ lambda(A,man(A))) @ lambda(B,cry(B))
4. Sem=lambda(F,exists(N,X, lambda(A,man(A))@X&F@X&least(N,4)))@
        lambda(B,cry(B))
5. Sem=lambda(F,exists(N,X, man(X)&F@X&least(N,4)))@ lambda(B,cry(B))
6. Sem=exists(N,X, man(X)& lambda(B,cry(B))@X&least(N,4))
7. Sem=exists(N,X, man(X)& cry(X))&least(N,4))
```

which represents the formula:

$$\exists_N X.(man(X) \wedge cry(X) \wedge least(N, 4))$$

With this method we can represent sentences like "At the most three men", "John eats maximum three apples", etc.

2.2. **Implementing Lambda Calculus.** Of course, now we want to reduce this complicated lambda expressions into readable first order formulas by carrying out β-conversion. The following code does this (see [1, 2]):

```
betaConvert(Var,Result):- var(Var),!, Result=Var.
betaConvert(Functor@Arg,Result):-
   compound(Functor),
   betaConvert(Functor,ConvertedFunctor),
   apply(ConvertedFunctor,Arg,BetaConverted),!,
   betaConvert(BetaConverted,Result).
betaConvert(Formula,Result):-
   compose(Formula,Functor,Formulas),
   betaConvertList(Formulas, ResultFormulas),
   compose(Result,Functor,ResultFormulas).
```

The first clause of the `betaConvert/2` simply records the fact that variable cannot be further reduced. The second clause does the most important things: it checks whether the functor is complex term and then reduces it to a lambda expression. If that succeeds, it applies the converted functor to `Arg` using `apply/3`.

The third and final clause breaks down formulas and predicates and reduces their arguments of subformulas. This is done by the help of:

```
betaConvertList([],[]).
betaConvertList([Formula|Others],[Result|ResultOthers]):-
   betaConvert(Formla,Result),
   betaConvertList(Others,ResultOthers).
```

In order to define the `apply/3` we first define the `substitute/4` predicate:

```
substitute(Term,Var,Exp,Result):-
   Exp=Var, !, Result=Term.

substitute(_Term,_Var,Exp,Result):-
   \+ compound(Exp) , !, Result=Term.
substitute(Term,Var,Exp,Result):-
   compose(Formula,Functor,[Exp,F]),
   member(Funcotr,[lambda, forall, exists]), !,
   (
   Exp=Var, !,
   Result=Formula
   ;
   substitute(Term,Var,F,R),
   compose(Result, Functor, [Expr,R])
).
substitute(Term, Var, Formula, Result):-
   Compose(Formula, Functor, ArgList),
```

```
   substituteList(Term, Var, ArgList, ResultList),
   compose(Result, Functor, ResultList).
substituteList(Term, Var, [], []).

substituteList(Term, Var, [Exp|Others], [Result| ResultOthers]):-
   substitute(Term,Var, Exp, Result),
   substituteList(Term, Var, Others, resultOthers).
```

Here is an example about the functionality of this predicate.

```
?-substitute(A,B,love(C,B), Result).

Result=love(C,A)
```

then `apply/3` will be

```
apply(lambda(X,Formula), Argument, Result):-
   substitute(Argument, X, Formula, Result).
```

To finish off, we define a driver predicate that calls the parser to analyze a sentence, reduces the resulting lambda expression into a first order formula, and directs the result to the standard output:

```
parse:-
   readLine(Sentence),
   s(LambdaExpression, Sentence,[]),
   normalform(LambdaExpression, NormalLambdaExpression),
   betaConvert(NormalLambdaExpression, Formula),
   printRepresentation(Formula).
```

Here the predicate `normalform(LambdaExpression, NormalLambdaExpression)` transform a `LambdaExpression` into a `normalform` labda expression, i.e. if we have the `LambdaExpression`: `exists(1,X,lambda(A, man(A))&room(X))@vincent` in normal form it will be `lambda(A, exists(1,X,man(A)&room(X))@vincent.`

References

[1] P. Blackburn, J. Bos, *Representation and Inference for Natural Language.* A first Course in Compuational Semantics, Volume I Working with first oreder logic. Computerlinguistik, Universitt des Saarlandes, September 1999.

[2] P. Blackburn, J. Bos, *Representation and Inference for Natural Language.* A first Course in Compuational Semantics, Volume II Working with discourse representation structures". Computerlinguistik, Universitat des Saarlandes, September 1999.

[3] D. Tatar ,*Bazele matematice ale calculatoarelor*, Curs lito. Universitatea "Babes-Bolyai" Cluj- Napoca, Facultatea de Matematica si Informatica, 1993.

[4] J. Allen, *Natural Language Understanding*, The Benjamin/ Cumings Publishing Company, Massachusetts, 1995

[5] D. Tatar, A. Onet, *Automated definite clause grammars compiling*, Presented at ROSYCS' 98 at the "Alexandru Ioan Cuza" University of Iasi, 1998.

"Babes-Bolyai" University, Faculty of Mathematics and Computer Science, RO-3400 Cluj-Napoca, Romania

E-mail address: {adrian,dtatar}@cs.ubbcluj.ro

STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume **XLIV**, Number 2, 1999

ANNIVERSARIES

PROFESSOR GRIGOR MOLDOVAN AT HIS 60TH ANNIVERSARY

E. MUNTEANU AND M. FRENTIU

Professor G. Moldovan was born on December 29, 1939 in Vadu Izei, Maramures. After finishing secondary school in 1956, he took up a teacher career in his native village. Two years later, in 1958, he arrived in Cluj and studied Mathematics at "Babeş-Bolyai" University. In 1963, after his graduation, he was named assistant at the Department of Computation, Faculty of Mathematics, "Babeş-Bolyai" University of Cluj. Since 1963 Professor G. Moldovan gave many courses and seminars on Numerical Analysis, Fundamentals of Computer Science, Computer Architecture, Data Structures, Formal Languages and Automata, Distributed Data Base Systems, etc. In 1975 he became an assistant professor, and in 1990 a full professor of this faculty.

Simultaneously with his pedagogical work, Professor G. Moldovan has developed an appreciated research work in numerical analysis and computer science. He received his Ph. D. degree in 1972, in Approximation Theory, under the supervision of Acad. T. Popoviciu and then Prof. D. D. Stancu. Since 1971, when the first students in Computer Science had begun their studies, he became interested in this new field of activity. In these topics he has written a lot of books and research papers, mentioned in the Abstract.

Two events, that changed his entire research activity, must be mentioned. The first is the one already mentioned above, the birth of Computer Science Section in our Faculty, in 1971, and the second, in 1975, is the foundation of our University Computer Centre (with a Felix C-256 computer). And Professor G. Moldovan was the head of the Computer Center since its very beginning.

Professor G. Moldovan was a tenacious and consistent director of Computer Center, a very good organiser, a hard worker man, a valuable teacher, a very good colleague, and an important PhD supervisor.

Now, on celebrating his 60-th birthday, we wish him Many Happy Returns of the Day and a long life in health and happiness.

ABSTRACT
of the Scientifical Work of Professor Grigor MOLDOVAN

(1) G. MOLDOVAN, *Asupra unui procedeu de integrare numerică a unei ecuaţii diferenţiale de ordinul întâi.* G.M. Seria A (nr. 5), 69, 1964, 161-166.

(2) D. D. STANCU, G. MOLDOVAN, *Proiect de terminologie românească pentru limbajul ALGOL-60 cuprinzând şi terminologia corespunzătoare în limbile: engleză, franceză, germană, rusă.* Litografiat, Universitatea din Cluj, 1965.

(3) G. MOLDOVAN, *Asupra aproximării funcţiilor continue prin polinoame Bernstein.* Studia Univ. "Babeş-Bolyai", Seria Math.-Physica, 11(1), 1966, 63-71.

(4) G. MOLDOVAN, I. RâP: *Asupra unei inegalităţi din teoria aproximării funcţiilor de doua variabile prin polinoamele lui Bernstein.* St. Cerc. Mat. 18 (nr. 6), 1966, 845-853.

(5) G. MOLDOVAN, *O problemă de aproximare prin polinoamele lui Bernstein.* Studia Univ. "Babeş-Bolyai", Seria Math.-Physica, 12(2), 1967, 51-57.

(6) G. MOLDOVAN, *Aproximarea funcţiilor continue prin polinoamele lui Bernstein generalizate.* Volum: Lucrările Colocviului de teoria aproximării funcţiilor, Cluj, 15-20 septembrie 1967, 139-140.

(7) G. MOLDOVAN, *Comportarea soluţiilor sitemelor de ecuaţii diferenţiale cu coeficienţi constanţi care depind de un parametru mic.* Studia Univ. "Babeş-Bolyai", Seria Math.-Physica, 1968, 21-30.

(8) G. MOLDOVAN, *Asupra unor operatori de tip Bernstein.* Studia Univ. "Babeş-Bolyai", Seria Math.-Physica, 14(2), 1969, 59-64.

(9) G. MOLDOVAN, *Asupra ordinului de aproximare al unui operator a lui V. A. Baskakov.* Studia Univ. "Babeş-Bolyai", Seria Math.-Physica,16(1), 1971, 69-71.

(10) G. MOLDOVAN, *Sur la convergence de certains operateurs convolutifs positive.* C. R. Acad. Sci. Paris, Serie A-B,272, 1971, A 1311-1313

(11) G. MOLDOVAN, *Generalizări ale polinoamelor lui S.N. Bernstein.* Teză de doctorat, Cluj, 1971

(12) G. MOLDOVAN, *Discrete convolutions and linear positive operators I.* Annales Univ. Sci. Budapesta de Rolando EOtvOs nomin., Section Math.,15, 1972, 31-44

(13) G. MOLDOVAN, *Asupra unor relaţii convolutive.* Studia Univ. "Babes-Bolyai", Seria Math.-Physica, 1972, 67-72.

(14) E. DANI, G. MOLDOVAN, A. MURESAN, *Asuprea clasificarii drumurilor in teoria grafelor.* Studia Univ. "Babeş-Bolyai", Series Oeconomica, f.1, 1972, 137-142.

(15) G. MOLDOVAN, *Convergenţa şirurilor valorilor unor operatori convolutivi in Rm.* Studia Univ. "Babeş-Bolyai", Seria Math.-Mech., 1973, 69-80.
(16) P. T. MOCANU, G. MOLDOVAN , M. O. READE, *Numerical computation of the a-convex Koebe function.* Studia Univ. "Babes-Bolyai", Seria Math.-Mech., 1974, 37-46.
(17) G. MOLDOVAN, *Convoluţii discrete relative la funcţii de mai multe variabile şi operatori liniari.* Studia Univ. "Babeş-Bolyai", Seria Math.-Mech., 1974, 51-57.
(18) G. MOLDOVAN, *Operateurs convolutifs positifs.* Seminaire d'Analyse Numerique de Grenoble, 1974, 1-31(14.02.1974).
(19) G. MOLDOVAN, *Quelque proprieté algebrique des opérateurs convolutifs positifs.* Seminaire d'Analyse Numerique de Grenoble, 1974, 1-31(14.03.1974).
(20) G. MOLDOVAN, G. MURESAN, T. TOADERE, *Asupra unui sistem informatic privind evidenţa studenţiilor.* Studia Univ. "Babes-Bolyai", Seria Math.-Mech., 1979, 46-50.
(21) G. MOLDOVAN, *L'evolution de l'erreur de l' approximation d' une fonctions continue par certaine operateurs linears positifs.* Mathematica, 22(45), nr. 1, 1980, 85-95.
(22) G. MOLDOVAN, *Proprietăţi algebrice ale unei clase de operatori convolutivi pozitivi.* Studia Univ. "Babes-Bolyai", Seria Mat -Mech., 1981, 9- 14.
(23) G. MOLDOVAN, *Modelul relaţional pentru baze de date.* Metodologii si tehnici moderne de proiectare si scriere a programelor. Bucuresti, Tipografia Universităţii Bucureşti, 1981, 348-352.
(24) G. MOLDOVAN, P. POP, R. POP DELEAN: *Pachet de programe pentru exploatarea bazei de date personal MEI la nivelul şi necesităţile Inspectoratelor Şcolare Judeţene.* Informatica în învăţământ. Nr.5, 1981, 3-6.
(25) G. MOLDOVAN, G. MURESAN, C. POPESCU, T. TOADERE, T. TOKES, *Simularea procesului de dizolvare in sonde.* Lucrările Institutului de cercetări miniere pentru substanţe nemetalifere, 1983, 347-348.
(26) T. TOKES, G. MOLDOVAN, G. MURESAN, T. TOADERE, *Elaborarea modelului matematic pentru calculul pierderilor de presiune pe reţele tehnologice.* Lucrările Institutului de cercetări miniere pentru substanţe nemetalifere, 1983, 349-351.
(27) G. MOLDOVAN, *Reorganization of a distributed data base.* Univ. of Cluj-Napoca, Fac. Math. Res. Sem.Preprint no. 5, 1984, 3-10.
(28) I. GARBACEA, G. MOLDOVAN, *Distributed data bases.* Univ. of Cluj-Napoca, Fac. Math. Res. Sem.Preprint no. 5, 1984, 70-91.

(29) G. MOLDOVAN, G. MURESAN, T. TOADERE, T. TOKES, *Two Mathematical Models for Salt Layer Working.* Univ. of Cluj-Napoca, Fac. Math. Res. Sem. 5, 1984, 117-127.

(30) G. MOLDOVAN, G. MURESAN, T. TOADERE, *Sistem informatic privind înscrierea şi evidenţa studenţilor.* Volumul cu lucrarile Simpozionului : "Informatica şi aplicaţiile sale", Zilele Academice Clujene, 3-7 decembrie 1985, p.159-162

(31) G. MOLDOVAN, *O problemă privind bazele de date distribuite.* Volum cu: Lucrarile prezentate la a VIII-a consfătuire a lucrătorilor din unităţile de informatică , Centrul de calcul coordonator al MEI, Institutul de Construcţii Bucureşti, 1985, p.102-103.

(32) G. MOLDOVAN, G. MURESAN, T. TOADERE, T. TOKES: *Model matematic şi programe privind exploatarea sării prin dizolvare.* Volumul cu lucrarile Simpozionului: "Informatica şi aplicaţiile sale", Zilele Academice Clujene, 3-7 decembrie 1985, 153 - 158.

(33) G. MOLDOVAN, *Caracterisation des fonctions convexes a l'aide des operateurs convolutifs positifs.* Studia Univ. "Babeş-Bolyai", Seria Math., 1986, 47-51.

(34) G. MOLDOVAN, *O problemă de optimizare într-un sistem de baze de date distribuite.* Volumul cu lucrăriele Simpozionului: "Informatica şi aplicaţiile sale", Zilele Academice Clujene, 20 noiembrie 1986, p.13-19.

(35) G. MOLDOVAN, S. DAMIAN, *On some generalization of on optimization problem for distributed data bases.* Studia Univ. "Babeş-Bolyai", Seria Math., 1987, 64-69.

(36) G. MOLDOVAN, *An automorfism property of a class of polinomial type positive convolutive operators.* Studia Univ. "Babeş-Bolyai", Seria Math., 1987, 70-72.

(37) G. MOLDOVAN, *Asupra unei probleme de optimizare pentru baze de date distribuite.* Volum: Lucrarile sesiunii de cercetări ştiinţifice a Centrului de calcul al Universităţii din Bucureşti, 20-21 februarie 1987, 368-373.

(38) G. MOLDOVAN, B. PâRV, *Biblioteca dBase II application for bibliographical documentation.* Univ. of Cluj-Napoca, Fac. Math. Res. Sem. 5, 1987, 36-45.

(39) G. MOLDOVAN, S. DAMIAN, *On an optimization problem for distributed data bases.* Analele Univ. Bucuresti, Math.-Info, 37, 1988, f. 2, 82-87.

(40) G. MOLDOVAN, I. RâP, *Asupra unor probleme de asteptare in retele de calculatoare.* Lucrarile conferinţei de matematică aplicată şi mecanică, Cluj-Napoca, 21-23 oct. 1988, 493-501.

(41) G. MOLDOVAN, S. DAMIAN, *O problemă de optimizare locală pentru redistribuirea bazelor de date într-o reţea de calculatoare.* INFO Iasi '89, 19-21 oct. 1989, 289-300

(42) G. MOLDOVAN, S. DAMIAN, *A local optimization problem for data base redistribution in a computer net.* Studia Univ. "Babes-Bolyai", Mathematica, 34, f.3, 1989, 3-16.

(43) G. MOLDOVAN, *Reorganizarea unei baze de date distribuite.* Univ. Cluj - Napoca, Fac. Mate-Info. Res. Sem. Preprint no.5, 1992, 116-123

(44) G. MOLDOVAN, I. RâP, *Files d'attente dans des systems distribuees ses services.* Studia Univ. "Babeş-Bolyai", Math., 37, f.3, 1992, 69-74.

(45) C. BOBOILĂ, G. MOLDOVAN,: *Modélisation des objets complexes avec identité d'objet.* An. Univ. Craiova Ser. Mat.-Inform. 21,1994, 89-98.

(46) G. MOLDOVAN, C. BOBOILA: *Un modle de représentation d' objets complexes avec identité d'objet.* Studia Univ. "Babeş-Bolyai",Series Math., 40(3), 1995, 67 - 80.

(47) G. MOLDOVAN, C. BOBOILA, *Une approche sur la modelisation d'objets complexes dans un systeme oriente-objet.* An. Univ. Craiova Ser. Mat.-Inform. 22(1995), 98-106.

(48) G. MOLDOVAN, C. BOBOILA, *L'objet actif base d'un modele de donees oriente objet.* University of Cluj-Napoca, Fac. of Math. and Computer Science, Research Seminaris, Preprint no. 2, pp. 43-52, 1995

(49) G. MOLDOVAN, C. BOBOILA, *Un modele a objets complexes avec identite d'objet.* Univ. "Babeş-Bolyai", Cluj - Napoca, Fac. of Math. and Computer Science, Research Seminaris, Preprint no. 2, pp. 53-60, 1995

(50) G. MOLDOVAN, C. BOBOILA, *Un modle de définition et représentation des objets complexes pour les bases de données.* Proceedings of ROSYCS' 96, Iaşi, mai, 1996. 229-240.

(51) G. MOLDOVAN, V. VARGA, *Theoretical Problems of Distributed Databases Fragmentation.* Univ. "Babeş-Bolyai", Cluj - Napoca, Fac. of Math. and Computer Science, Research Seminaris, Preprint no. 2, pp. 79-84, 1996

(52) G. MOLDOVAN, M. MOLDOVAN, S. MOLDOVAN, *Sisteme şi posibilităţi de modelare a lor.* Volum: Lucrările Sesiunii de Comunicări Ştiinţifice a Universităţii "Bogdan Vodă" Baia Mare, martie 1997, Ed. Risoprint, Cluj-Napoca, 1997 (ISBN- 973-9298-IS- 4) p.121-135.

(53) G.MOLDOVAN, A. VANCEA, M. VANCEA, *Data dependence testing for automatic parallelization.* Studia Univ."Babeş-Bolyai", Informatica, 2(1), 1997, pp3-18.

(54) G. MOLDOVAN, *Catalan Numbers and binary recursive defined objects.* Univ. "Babeş-Bolyai", Cluj - Napoca, Fac. of Math. and Computer Science, Research Seminars, Preprint no. 2, pp. 189-200, 1998.

BOOKS

(55) Gr. MOLDOVAN, *Scheme logice şi programe FORTRAN, Litogr.* Univ. din Cluj-Napoca, 1973, 178 pag.

(56) Gr. MOLDOVAN, *Scheme logice şi programe FORTRAN,* Ed. Didactic şi pedagogică, Bucureşti, 1976, 188 pag.

(57) Gr.MOLDOVAN, L. TAMBULEA, F. LANDA, D. OPREAN, *Scheme logice şi programe COBOL. Culegere de probleme,* Litogr.Univ."Babeş-Bolyai", Cluj- Napoca, 1979, 205 pag.

(58) Gr. MOLDOVAN, *Bazele informaticii II*, Litogr.Univ. Cluj-Napoca, 1985, 286 pag.

(59) Gr. MOLDOVAN, F. Boian şi alţii, *MATH-I. Biblioteca de programe ştiinţifice pentru calculatoarele personale româneşti*, ITCI Software,1987, 129 pag.

(60) Gr. MOLDOVAN, *Elemente de informatică,* Litogr.Univ. Cluj-Napoca, 1985, 86 pag.

(61) F. BOIAN, D. CHIOREAN, S. DAMIAN, Gr. MOLDOVAN, I. PARPUCEA, *Minicalculatoru CORAL şi sistemul de operare MIX,* Univ. Cluj-Napoca, 1986, 80 pag.

(62) Gr. MOLDOVAN, V. CIOBAN, M. LUPEA, *Probleme pentru programare în limbajul C, Litogr.* Univ. "Babeş-Bolyai" Cluj-Napoca, 1995, 151 pag.

(63) Gr.MOLDOVAN, V. CIOBAN, M. LUPEA, *Limbaje formale şi teoria automatelor. Culegere de probleme*, Litogr.Univ. "Babeş-Bolyai" Cluj-Napoca, 1996, 148 pag.

(64) Gr. MOLDOVAN, R. JOLDEŞ, *Descrierea algoritmilor. Teorie şi aplicaţii,* Univ. "1 Decembrie 1918", Alba Iulia, 1996, 139 pag.

(65) Gr. MOLDOVAN, V. CIOBAN, M. LUPEA, *Limbaje formale şi teoria automatelor. Culegere de probleme*, Ed. Mesagerul, Cluj-Napoca, 1997, 150 pag.

(66) Gr. MOLDOVAN, *Limbaje formale şi tehnici de compilare,* Univ. "Babeş- Bolyai" Cluj-Napoca, Centrul de formare continuă şi învăţământ la distanţă, Cluj-Napoca, 1999, 106 pag.

"Babeş-Bolyai" University of Cluj-Napoca, Department of Computer Science
E-mail address: mfrentiu@cs.ubbcluj.ro