Anul XLVI 2001

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

1

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

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

AN EFFICIENT WAY TO MODEL P SYSTEMS BY X-MACHINE SYSTEMS

HORIA GEORGESCU

Abstract. Starting from the powerful concept of stream X-machine, the author proposed in some previous papers an original way to integrate more stream X-machines with ε-transitions into a system. The communication between the components is assured by means of a communication matrix, used as a common memory. It was proved that by introducing an additional component, it is possible to achieve communication in a structured way, namely by channels, with **select** constructs appearing in each communicating state. This model was used for proving that P systems may be modelled by communicating stream X-machine systems. Even if the proof is a constructive one, it is not satisfactory due to its low level of concurrency. In this paper a much more efficient construction, from the concurrency's point of view, is presented.

Keywords: *P systems, Communicating X-machine systems, concurrent processes, communication using channels.*

1. Introduction

The concept of *X-machines* was introduced by Eilenberg in [4] and used by Holcombe [7] as a possible specification language. Afterwards, a lot of research has been done in this field.

The new features which differentiate an X-machine from a finite-state one are: a set of processing functions Φ, an input and an output tape and a set X which characterizes the internal memory of the machine. The transitions between the states are performed according to these functions. The X-machine evolves from one state to another according to the current state, the content of the input tape and internal memory and the function chosen to be applied. A new item may be added to the output tape after a transition takes place. In this way both the system data and the control structure are modelled, while allowing them to be separated.

2000 *Mathematics Subject Classification.* 68N19, 94A40.

1998 *CR Categories and Descriptors.* B.4.4 [**Hardware**]: Input/Output and Data Communication – *Performance Analysis and Design Aids*; D.1.3 [**Software**]: Programming Techniques – *Concurrent Programming.*

The research performed during the last years concerned mainly their generative power and their possible use for testing. Very little attention has been paid to the possible communication between these machines and consequently to their use for specification of concurrent processes.

In [2] cooperating distributed grammar systems are used for modelling their concurrent behaviour.

In [5] another approach was proposed, using a communication matrix for the communication between the X-machines. Using this model, in [1] it was proved that communicating stream X-machines systems are equivalent (from the generative power point of view) with a single X-machine.

A different approach from [5] is proposed in [3], where more powerful tools are added. The new mode allows the use of channels as the mechanism for passing messages between the X-machines. It also allows implementation of specific constructs for channels as **select** with an optional **terminate** clause. The main idea was to introduce a new X-machine, called *Server*, for controlling the communication between the components of the system; the last column (the one corresponding to the X-machine *Server*) of the communication matrix is used too. Following this model, an automatic scheme for writing concurrent programs (in Pascal-FC or Ada like style) was proposed.

Any non-trivial biological system is a hierarchical construct, made up of several *organs* which are well defined and delimited from their neighbouring organs. Each organ evolves internally, but also cooperates with neighbour organs in order to keep alive the system as a whole; cooperation consists in a flow of materials, energy and information, necessary for the functioning of the system.

A membrane structure is composed of *regions* delimited by *membranes*. A region is a space enclosed by membranes. neighbour regions communicate through the membranes separating them. The space outside the skin membrane is called the *outer region*.

P systems have been studied mainly from the point of view of their computational power and it was shown that their generative power is that of the Turing machines. The main results can be found in [8] and [9].

P systems were used also for solving NP-complete problems in polinomial time. For this purpose, one approach is to allow the number of membranes to grow dinamically (see [11]), while another approach is to count only the changes of configuration (see [10]). Unfortunately, in the first approach the number of membranes grows exponentially, while in the second approach the local time complexity is exponential too.

In [6] it was shown that P systems may be modelled by communicating stream X-machine systems.

In Section 2 and Section 3, the basic results concerning communicating stream X-machine systems (CSXMS) and P systems are reviewed. In section 4 it is presented a new construction for modelling P systems by CSXMS; this construction assures a high degree of concurrency.

2. Communicating Stream X-machine Systems

Definition 1. *A stream X-machine with ε-transitions is a tuple:*

$$X = (\Sigma, \Gamma, Q, M, \Phi, F, I, T, m^0),$$

where:

- Σ *and* Γ *are finite sets called the input and output alphabets, respectively;*
- Q *is the finite set of states;*
- M *is a (possibly infinite) set called memory;*
- Φ *is a finite set of partial functions of the form:*

$$f : M \times \Sigma^{\varepsilon} \to \Gamma^{\varepsilon} \times M;$$

- F *is the next state function* $F : Q \times \Phi \to 2^Q$*;*
- I *and* T *are the sets of initial and final states;*
- m^0 *is the initial memory value*

together with an input tape and an output tape. F must be a total function.

We define a *configuration* of the X-machine by (m, q, s, g), where $m \in M, q \in Q, s \in \Sigma^\star, g \in \Gamma^\star$. A machine computation starts from an initial configuration $(m^0, q^0, s^0, \varepsilon)$, where $q^0 \in I$ and $s^0 \in \Sigma^\star$ is the input sequence.

A *change of configuration*, denoted by $\vdash : (m, q, s, g) \vdash (m', q', s', g')$ is possible if:

- $s = \sigma s'$, $\sigma \in \Sigma^{\varepsilon}$;
- there is a function $f \in \Phi$ with $F(q, f) \neq \emptyset$ and $q' \in F(q, f)$ (in which case we say that *f may be applied, f emerges from q and reaches q'*), so that $f(m, \sigma) = (\gamma, m')$ and $g' = g\gamma$, $\gamma \in \Gamma^{\varepsilon}$.

The output corresponding to an input sequence $s \in \Sigma^\star$, is defined as:

$$X(s) = \{g \in \Gamma^\star | \exists m \in M, q^0 \in I, q \in T, \text{so that } (m^0, q^0, s, \varepsilon) \overset{\star}{\vdash} (m, q, \varepsilon, g)\}$$

where $\overset{\star}{\vdash}$ denotes the reflexive and transitive closure of $\vdash$.

Definition 2. *A Communicating Stream X-machine System (CSXMS for short) is a system*

$$S_n = ((X_i)_{i=1,\ldots,n}, \mathcal{CM}_n, C^0),$$

where:

- $X_i = (\Sigma_i, \Gamma_i, Q_i, M_i \times \mathcal{CM}_n, \Phi_i, F_i, I_i, T_i, m_i^0)$ *are X-machines;*

- *$M = M_1 \cup \ldots \cup M_n$ is the set of memory values and $\widetilde{M} = M \cup SS$ is the set of (general) values, where SS is a set of special strings of symbols different from those in M.*
- *$\mathcal{CM}_n$ is the set of all matrices of order $n \times n$ with elements in $\widetilde{M}$. This set defines the possible values of the global memory of the system, which is used for communication between the component X-machines; therefore it is referred to as the communication matrix;*
- *C^0 is the initial communication matrix;*
- *for any i and $f \in \Phi_i$, $f : M_i \times \mathcal{CM}_n \times \Sigma_i^\varepsilon \to \Gamma_i^\varepsilon \times M_i \times \mathcal{CM}_n$.*

Let $\Gamma = \Gamma_1 \cup \ldots \cup \Gamma_n$. A common output tape O is used by all components. It is initially void and afterwards it contains sequences $g \in \Gamma^\star$.

We mention that each X-machine X_i can access (read from or write to) only $+_i$, where $+_i$ denotes the set of all elements in the ith line and ith column of the communication matrix. An element C_{ij} may receive the value @, meaning that the connection from X_i to X_j is disabled. A disabled connection can not be enabled later.

In each X-machine X_i there are two kinds of states: $Q_i = Q_i' \cup Q_i''$, $Q_i' \cap Q_i'' = \emptyset$, where Q_i' contains *processing states* and Q_i'' contains *communicating states*. The final states are processing states; there is no function emerging from them.

The functions emerging from a processing state depend only on the local memory and on the local input tape and are meant to (partially) change the local memory and possibly add some information to the output tape O.

The functions emerging from a communicating state depend on the local memory and on $+_i$ and are meant to move a value from the internal memory to the communication matrix and viceversa, as well to assign a special value to the communication matrix.

When an X-machine X_i moves to a final state, all elements in $+_i$ have to change their values into @.

A *configuration* of a CSXMS system S_n has the form: $z = (z_1, \ldots, z_n, C)$, where:

- $z_i = (m_i, q_i, s_i, g_i)$, $i = 1, \ldots, n$;
- m_i is the current value of the memory M_i of X_i;
- q_i is the current state of X_i;
- $s_i \in \Sigma_i^\star$ is the current input sequence of X_i;
- $g_i \in \Gamma_i^\star$ is the current output sequence of X_i;
- C is the current value of the communication matrix of the system.

The system starts with all X-machines in their initial states, $C = C^0$ and $M_i = m_i^0$ for all $i \in \{1, \ldots, n\}$. The *initial configuration* of the system is $z^0 = (z_1^0, \ldots, z_n^0, C^0)$, where $z_i^0 = (m_i^0, q_i^0, s_i^0, \varepsilon)$ with $q_i^0 \in I_i$.

We can think about a change of configuration $z \models z'$ as follows: let t be the time when the system reached the configuration z and t' the closest following moment

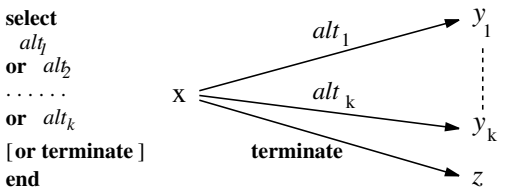


FIGURE 1. The **select** construct for communicating states

of time at which a component terminates the execution of a function; then z' is the configuration of the system at time t'.

A change of configuration:

$$z = (z_1, \ldots, z_n, C) \models z' = (z'_1, \ldots, z'_n, C') \tag{1}$$

with $z_i = (m_i, q_i, s_i, g_i)$, $z'_i = (m'_i, q'_i, s'_i, g'_i)$, $s_i = \sigma_i s'_i$, $\sigma_i \in \Sigma_i^{\varepsilon}$, $g'_i = g_i\gamma_i$, $\gamma_i \in \Gamma_i^{\varepsilon}$ for any i, may be described as follows. Let $C_0 = C$. For i taking the values $1, 2, \ldots, n$ in this order, there are two possibilities:

- either $z_i = z'_i$, or
- there exists a function $f \in \Phi_i$ emerging from q_i and reaching $q'_i \in F_i(q_i, f)$ and $C_i \in \mathcal{CM}_n$ so that $f(m_i, C_{i-1}, \sigma_i) = (\gamma_i, m'_i, C_i)$.

The X-machines act simultaneously. The system stops successfully when all X-machines reach final states (i.e. all values in C are @).

Let $\overset{\star}{\models}$ be the reflexive and transitive closure of $\models$. Then the *output computed by a CSXMS S_n corresponding to an input sequence s* can be defined as follows: $X(s) = \{g = (g_1, \ldots, g_n) \in \Gamma_1^{\star} \times \ldots \times \Gamma_n^{\star} \,|\exists z^0$ an initial configuration and z a final one, $z^0 \overset{\star}{\models} z$, with $z = (z_1, \ldots, z_n, C)$, $C \in \mathcal{CM}_n$ and $z_i = (m_i, q_i, \varepsilon, g_i)$ for any $i = 1, \ldots, n\}$.

The mechanism introduced above assures only a low level of synchronization. Therefore channels were intoduced as a higher level of synchronisation.

For this aim, in each communicating state of each X-machine X_i the classical **select** construct with guarded alternatives and **terminate** clause was introduced, as presented in Fig.1. The alternatives alt_s, $s \in \{1, \ldots, k\}$, should have the following forms:

1) [**when** $cond_k$ =>] $j\,!\,val$
2) [**when** $cond_k$ =>] $j\,?\,v$

with the following meanings:

1) if $cond_k$ is fulfilled, then val has to be sent to the X-machine X_j (via C_{ij});

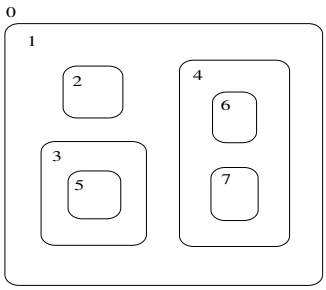


FIGURE 2. A membrane structure

2) if $cond_k$ is fulfilled, then v of M_i has to receive a value from the X-machine X_j (via C_{ji}).

The alternatives are *macrofunctions*; *val* is a memory value, $cond_s$ depends only on the local memory M_i and on the local input tape, and v is a variable of M_i. As usual, the square brackets show that the information they include is optional.

The **terminate** clause acts as follows: if the other alternatives in the **select** construct are false and will be false forever, then the X-machine stops. In other words, if present, the **terminate** clause applies when all X-machines X_j to/from which X_i tries to send/receive messages have stopped. Executing **terminate** implies moving to a final state.

In the following, the form of functions emerging from a communicating state can be only as in Fig. 1.

In [3] an implemetation of the **select** constructs was proposed and it was proved that this implementation is a correct one.

The above results are important due to the well known power of communication using channels. A first result appears in [3]: the authors present an automatic scheme for generating a concurrent program, written in an Pascal-FC or Ada like style, starting from an arbitrary CSXMS that uses in communicating states only **select** constructs. Another result appears in Section 4.

3. P Systems - Membrane Computing

Let us consider the membrane structure in Fig. 2, where the outer region has the label 0. Due to its intrinsic hierarchical type, a membrane structure may be represented in a paranthesized form. For the example in Fig. 2, the corresponding representation is: $[_1[_2]_2[_3[_5]_5]_3[_4[_6]_6[_7]_7]_4]_1$.

Membrane structures may be represented also as a special kind of trees, which we will call *P-trees*. A P-tree associated to a membrane structure is defined as follows:

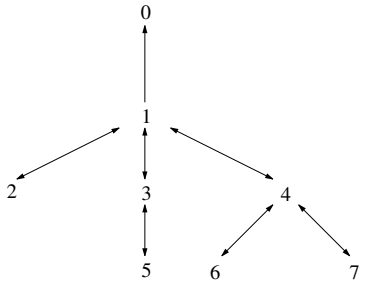


FIGURE 3. The tree associated to the membrane structure in Fig. 2

- each of the $n+1$ nodes in the tree corresponds to a region in the membrane structure; the nodes are labeled with $0, 1, ..., n$, where 0 is always the label of the outer region, and 1 is the label of the region just inside the skin membrane;
- for each node $i \in \{1, ..., n\}$, edges diverge towards its father and its sons, corresponding to the membranes which separate the respective region from the neighbour regions;
- no edge diverges from node 0.

The edges show the way in which information may be sent from one node (region) to another one.

The P-tree T associated to the membrane structure in Fig. 2 is shown in Fig. 3. To each node (region in the membrane structure) i we associate a finite languages M_i over an alphabet V. The elements of these languages are called *objects*. Since the sets of objects are changing while the system evolves, we denote by M_i^0 their initial content and by M_i their current content: $(M_0, ..., M_n)$ is the *current configuration* of the system.

Generally, a membrane structure is *dynamic*: a region may be *divided* into several regions or may be *dissolved*. In this paper we will consider only *static structures*, in which their forms are never changed, i.e. no region can be divided or dissolved.

Let R_i be the set of *evolution* (*developmental*, *replicated rewriting*) *rules* for each region $i = 1, 2, ..., n$. They have the form:
$a \to (\alpha_1, t_1) \parallel \ \dots \ \parallel (\alpha_k, t_k)$
for some k, where $a \in V$ and $\alpha_1, ..., \alpha_k \in V^*$, while t_j, $j = 1, ..., k$ indicate the *targets* of the rule and can be only i, the father of i or one of its sons.

Such a rule can be applied to an object in M_i if the object has the form $o = \beta a \gamma$; if the rule is applied, o is erased and the new objects $\beta \alpha_j \gamma$ are created and added to M_{t_j} respectively.

A *computation* starts from the initial configuration $(M_0^0, M_1^0, ...M_n^0)$: these sets are initially associated to the nodes of the tree T.

At each step of the computation, the rules are applied to the objects in parallel; for each object one of the rules (if any) that can be applied to it is chosen randomly and the resulting objects are sent to their specified targets. All objects which do not evolve are passed unchanged to the next step.

A computation is *complete* if it halts: no rule can be used in the current configuration. In this case, the *output of the system* for this computation is M_0 (the language associated to the outer region).

We can now summarize the above disscussion concerning the (static) membrane structure as follows:

Definition 3. *A P system (membrane structure) is a construct*

$$\Pi = (V, \mu, (M_0^0, M_1^0, ..., M_n^0), R)$$

where:

- μ *is the membrane structure. Let us consider that it is represented as a P-tree with the nodes* $0, 1, ..., n$*, where* $n + 1$ *is the number of the nodes in the tree. The root of the tree is* 0 *and its son is* 1*;*
- V *is an alphabet;*
- $M_0^0, M_1^0, ..., M_n^0$ *are the sets of objects initially associated to the nodes* $0, 1, ..., n$*;*
- R *is the finite set of evolution (replicated rewriting) rules.*

A change of configuration $(M_0, M_1, ..., M_n) \mapsto (M_0', M_1', ...M_n')$ *is performed by applying rules to the objects in* $M_0, M_1, ..., M_n$ *in parallel, as described above, and by sending the resulting objects to the specified targets.*

The output of the system is the union L *of all the languages* M_0 *in final configurations.*

Remark 1. *Many variants of the above definitions may be considered. Some of them are presented below:*

- a P system (membrane structure) may be dynamic, not only static. In dynamic systems, the membranes may be divided, dissolved or thickened (the communication through them is inhibited);
- priorities may be attached to the evolution rules;
- the membranes may have electric charges (+, - or 0); in this case the evolution rules involve these electrical charges, too;
- the evolution rules may have more specific forms;
- the ouput can be related to an arbitrary node, not necessarily to the root of the tree. Moreover, the output may be considered to be $L \cap O^*$, where O is a given output alphabet;
- for each node i, a specific set of evolution rules may be considered;

- the sets M_i may be replaced by multisets, i.e. an object may appear more than once in M_i.

4. Implementing P systems using $CSXMS$

In [6] it was shown that a communicating X-machine system can be associated to any P system, which simulates the behaviour of the P system:

Theorem 1. *Any P system may be simulated by a $CSXMS$.*

The proof was a constructive one. Starting from a given P system, the corresponding CSXMS was built as follows. An X-machine P_i was associated to each node $i \in \{0, \dots n\}$. An additional X-machine P_{n+1} was considered too. In order to use communication through channels, the X-machine *Server* mentioned in Section 2 was added to the system.

The idea of this construct was quite simple. The process (X-machine) P_{n+1} receives first from each of $P_1, ..., P_n$ the new created objects that are to be sent to its neighbours together with their targets, as well as the number of the new generated objects (including those which remain local) and afterwards sends to $P_0, P_1, ..., P_n$ the objects for which they are targets. These two steps are executed repeatedly until the information which P_{n+1} receives at the first step is void.

The above construction has a simple form and solves the problem. However it is not satisfactory in real implementations, due to its very low degree of parallelism: P_{n+1} waits for *all* of the components $P_1, ..., P_n$ to send their information and only afterwards executes the second step. Therefore we will present a second and much more efficient way to associate a CSXMS to a P system.

We will omit some cumbersome details, as those concerning the form of the local memories of the component of the system of X-machines. The stress will be put on the actions performed by the components and on the communication between them.

The components of the system are $P_0, P_1, ..., P_n$ and *Server*. A channel is associated to each edge of the P-tree. For each P_i let f_i be the father of node i and S_i the set of the sons of this node (of course, if i is a leaf, then $S_i = \emptyset$); P_i is linked through channels in both directions to P_{f_i} and P_s, for all s in S_i.

The basic idea of this new construct is that the nodes have to receive the new objects sent to them in a postorder manner, i.e. an inner membrane always receives information before its enveloping membrane does it.

For this purpose each node, after generating new objects, performs the following actions:

- sends to each of its sons the appropriate information
- receives information from its sons
- receives information from its father

- sends to its father the appropriate information, together with the number of new objects generated in the subtree for which it is the root

The new objects process P_0 receives at each step are sent to the output tape. As mentioned above, P_0 receives also the total number of the new generated objects. Only if this number is zero (no new objects were generated in the nodes of the P-tree), it starts the halting procedure.

The halting procedure is done in a preorder manner, using (of course) the **select** construct with a **terminate** clause. Process P_0 merely stops; in this way, the channel linking him to it unique son is disabled. When the process associate to any other node tries to receive information from its father and this father has already stoped, it will stop too; this is ensured by including in the **select** implementing this receive operation, the **terminate** clause. In this way, all channels having this node as sender are disabled.

The component P_0 sends to P_1 the number 0 and receives from it the new generated objects for which it is the target, as well as the number k of all new generated objects when passing from a configuration to the next one. If $k = 0$ then P_0 halts.

$nr \leftarrow -1$
repeat
  if $nr = 0$
  then select
    terminate
  end
  else -
  $nr \leftarrow 0$
  select
    1 ! 0
  end
  select
    1 ? J, k
  end
  $nr \leftarrow k$
until false

The processes $P_1, ..., P_n$ perform the actions described above:
repeat
  $nr \leftarrow 0$
  generate the new objects, update the current set M_i and collect in J_{f_i}
    and $\{J_s | s \in S_i\}$ the information to be send to the neighbours
  for all $s \in S_i$
    select
      s ! J_s

end
endfor
for all $s \in S$
select
s ? J, k
end
$M_i \leftarrow M_i \cup J$; $nr \leftarrow nr + k$
endfor
select
f_i ? J, k
or terminate
end
$M_i \leftarrow M_i \cup J$; $nr \leftarrow nr + k$
select
f_i ! J_{f_i}
end
until false

The number of the components in the system is $n + 2$. The number of channels used for communication is $4n+2$: $2n$ channels linking the nodes in the P-tree and $2(n + 1)$ channels linking these nodes to *Server*.

The communication matrix may be replaced by a matrix with 4 lines and $n+1$ columns.

The main enhancement of this new construct consists in the fact that the degree of concurrency is much higher then in the former construct and in this way corresponds to the model originally designed for P systems.

The halting of the whole CSXM system is ensured (as mentioned) in preorder, so that some nodes may still perform some actions until they are "announced" by their father that they have to halt.

5. Conclusions

In this paper we proposed a new construct for modelling P systems by CXSMS. The implementation uses communication through channels between components, as described in [3]. The CSXM associated to a P system has a high degree of concurrency, which can be exploited when a multiprocessor device is available. The new construct can be rather easily implemented in programming languages like Java, so that the evolution of a membrane computing system may also be implemented in such a language.

Further work includes the study of the different variants of P systems, as described in the third section of this paper. A Java implementation of P systems, using the tools presented in this paper, is in course of development.

References

[1] T. Bălănescu, A. J. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, C. Vertan: Communicating stream X-machines are no more than X-machines: *J.UCS*, Vol. 5, Nr. 9, September 1999, 494–507.

[2] T. Bălănescu, H. Georgescu, M. Gheorghe : Stream X-machines with underlying distributed grammars. *Informatica* (submitted).

[3] A. J. Cowling, H. Georgescu, C. Vertan : A structured way to use channels for communication in X-machine systems, *FACS*, 12 (2000), 485–500.

[4] S. Eilenberg : *Automata, languages and machines*, Academic Press, 1974.

[5] H. Georgescu, C. Vertan: A new approach to communicating X-machines systems, *J.UCS*, 2000, Vol. 6, issue 5, 490–502.

[6] H. Georgescu : Modelling P systems by communicating X-machine systems. *Annals of Bucharest University, Mathematics-Informatics Series*, L (1), 3–12, 2001.

[7] M. Holcombe: X-machines as basis for dynamic system specification, *Software Engineering Journal* 3 (1988), 69–76.

[8] Gh. Păun: Computing with membranes. An Introduction, *Bulletin of EATCS*, 67 (2), 139–152, 1999.

[9] Gh. Păun: Computing with membranes, *J. of Computer and System Sciences*, 61, 1 (2000), 108–143.

[10] S. N. Krishna, R. Rama: P systems with replicated rewriting, *J. Automata, Languages, and Combinatorics*, (to appear).

[11] Gh. Păun: P Systems with active membranes: Attacking NP-complete problems, *J. Automata, Languages and Combinatorics* 6, 1 (2001).

Academy of Economic Studies, Bucharest, Romania
E-mail address: hg@phobos.cs.unibuc.ro, g_horia@hotmail.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

GENETIC CHROMODYNAMICS FOR OBTAINING CONTINUOUS REPRESENTATION OF PARETO REGIONS

D. DUMITRESCU, CRINA GROŞAN, AND MIHAI OLTEAN

Abstract. In [5] an evolutionary algorithm for detecting continuous Pareto optimal sets has been proposed. In this paper we propose a new evolutionary elitist approach combing a non-standard solution representation and an evolutionary optimization technique. The proposed method permits detection of continuous decision regions. In our approach an individual (a solution) is either a closed interval or a point. The individuals in the final population give a realistic representation of Pareto optimal set. Each solution in this population corresponds to a decision region of Pareto optimal set. Proposed technique is an elitist one. It uses a unique population. Current population contains non- dominated solutions already founded.

Keywords: evolutionary multiobjective optimization, Pareto set, Pareto frontier, Pareto interval

1. Introduction

Several evolutionary algorithms for solving multiobjective optimization problems have been proposed ([2], [5]–[10], [12], [13]; see also the reviews [1], [11] and [14]).

Usually Pareto evolutionary algorithms aim to give a discrete picture of the Pareto optimal set (and of the corresponding Pareto frontier). But generally Pareto optimal set is a continuous region in the search space. Therefore a continuous region is represented by a discrete set. When continuous decision regions are represented by discrete solutions there is loss of information. Moreover reconstructing continuous Pareto set from a discrete picture is not an easy task [11].

In [5] an evolutionary algorithm for detecting continuous Pareto optimal sets has been proposed. The method proposed in [5] uses Genetic chromodynamics evolutionary technique [4] to maintain population diversity.

2000 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* I.2.8 [**Computing Methodologies**]: Artificial Intelligence – *Problem Solving, Control Methods, and Search.*

In this paper a new evolutionary approach, combing a non-standard solution representation and a Genetic Chromodynamics optimization technique, is considered. Within the proposed approach continuous decision regions may be detected. A solution (individual) is either a closed interval or a point (considered as a degenerated interval). Mutation is the unique search operator considered.

The mutation operator idea is to expand each individual toward left and toward right. In this respect both interval extremities are mutated. The left extremity is mutated towards left and the right extremity is mutated towards right.

To reduce population size and to obtain the correct number of solutions within the final population the *merging* operator introduced in context of Genetic Chromodynamics is used.

The solutions in the final population supply a realistic representation of Pareto optimal set. Each solution in this population corresponds to a decision region (a subset of Pareto set). A decision region will also be called a *Pareto region*.

The solutions are detected in two stages. In the first stage a Genetic Chromodynamics technique is used to detect all (local and global) Pareto solutions. In the second stage the solutions are refined. During the fine tunning the sub-optimal regions are removed.

The evolutionary multiobjective technique proposed in this paper is called *Continuous Pareto Set* (CPS) algorithm.

2. Problem statement

Let Ω be the search space. Consider n objective functions f_1, f_2, ..., f_n ,

$$f_i : \Omega \to \mathcal{R},$$

where $\Omega \subset \mathcal{R}$.

Consider the multiobjective optimization problem:

$$\begin{cases} \text{optimize } f(x) = (f_1(x), \ldots, f_n(x)) \\ \text{subject to } x \in \Omega \end{cases}$$

The key concept in determining solutions of multiobjective optimization problems is that of Pareto optimality. In what follows we recall some basic definitions.

Definition1. *(Pareto dominance)* Consider a maximization problem. Let x, y be two decision vectors (solutions) from Ω. Solution x *dominate* y (also written as $x \succ y$) if and only if the following conditions are fulfilled:

(i) $f_i(x) \geq f_i(y), \forall i = 1, 2, \ldots, n$,
(ii) $\exists j \in \{1, 2, \ldots, n\} : f_j(x) > f_j(y)$.

Definition 2. Let $S \subseteq \Omega$. All solutions, which are not dominated by any vector of S, are called *nondominated* with respect to S.

Definition 3. Solutions that are nondominated with respect to S, $S \subset \Omega$, are called *local Pareto* solutions or *local Pareto regions*.

Definition 4. Solutions that are nondominated with respect to the entire search space Ω are called *Pareto optimal* solutions.

Let us note that when the search space is a subset of $\mathcal{R}$, then Pareto optimal set may be represented as:

(i) a set of points;
(ii) a set of disjoint intervals;
(iii) a set of disjoint intervals and a set of points.

Remark. In each of the cases (i), (ii) and (iii) a point or an interval represents a Pareto region.

Evolutionary multiobjective optimization algorithms are intended for supplying a discrete picture of Pareto optimal set and of Pareto frontier. But Pareto set is usually a union of continuous set. When continuous decision regions are represented by discrete solutions there is some information loss. The resulting sets are but a discrete representation of their continuous counterparts.

Methods for finding Pareto optimal set and Pareto optimal front using discrete solutions are computationally very difficult. However the results may be accepted as the 'best possible? at a given computational resolution. Methods for obtaining the continuous representation using discrete outputs of evolutionary techniques are considered in Veldhuizen [11].

The evolutionary method proposed in this paper directly supply the true (i.e. possibly continuous) Pareto optimal set.

3. Solution representation and domination

In this paper we consider solutions are represented as intervals in the search space Ω.

Each interval-solution k is encoded by an interval $[x_k, y_k] \in \mathcal{R}$. Degenerated intervals are allowed. Within degenerate case $y_k = x_k$ the solution is a point.

In order to deal with proposed representation a new domination concept is needed. This domination concept is given by the next definition.

Definition 5. An interval-solution $[x, y]$ is said to be *interval-nondominated* if and only if all points of that interval $[x, y]$ are nondominated point-wise solutions. An interval-nondominated solution will be called a *Pareto-interval*. **Remarks.**

(i) If $x = y$ this concept reduced towards ordinary non-domination notion.
(ii) If no ambiguity arise we will use *nondominated* instead of *interval-nondominated*.

4. Mutation

Problem solutions are detected in two stages. In the first stage a Genetic Chromodynamics technique is used to detect all (global and local) solutions. This represent the *evolution stage*.

In the second stage (*fine tuning* or *refinement stage*) solutions that have been detected in the first stage are refined. By using the refinement procedure sub-optimal Pareto regions are removed from the final population.

Most of the multiobjective optimization techniques based on Pareto ranking use a secondary population (an archive) denoted P_{second} for storing nondominated individuals. Archive members may be used to guide the search process. As dimension of secondary population may dramatically increase several mechanisms for reducing archive size have been proposed. In [13] and [14] a population decreasing technique based on a clustering procedure is considered. We may observe that preserving only one individual from each cluster implies a loss of information.

In our approach the population size does not increase during the search process even if the number of Pareto optimal points increase. The population size is kept low due to the special representation we consider.

When a new nondominated point is found it replaces another point solution in the population or it is used for building a new interval solution with another point in the population. This does not cause any information loss concerning Pareto optimal set during the search process.

The algorithm starts with a population of degenerated intervals (i.e. a population of points). The unique variation operator is mutation. It consists of normal perturbation of interval extremities. Mutation can also be applied to point-solutions (considered as degenerated intervals). Each interval extremity is mutated. The left extremity of an interval is always mutated towards left and the right interval extremity is mutated only towards right.

For mutation two cases are to be considered.

a) Degenerated interval: The individual is mutated towards left or right with equal probability. The obtained point represents the offspring. Parent and offspring compete for survival. The best, in the sense of domination, enter the new population.

If parent and offspring are not comparable with respect to domination relation then the two points defines an interval solution. The new interval solution is included in the new generation. The point solution representing the parent is discarded.

b) Nondegenerated interval: (i) Firstly the left extremity of the interval $[u, v]$ is mutated towards left. A point-offspring u' is obtained. Consider the case when the offspring u' and the parent u do not dominate each other. In this situation a new interval solution $[u', v]$ is generated. The new solution has the offspring u' as its left extremity and v as its right extremity. If the offspring u' dominates the parent u, then the interval solution $[u, v]$ enters the new population.

(ii) A similar mutation procedure is applied to the right interval extremity of the previously obtained solution ($[u, v]$ or $[u', v]$).

5. Population model

For each generation every individual in the current population is mutated. Parents and offspring directly compete for survival. The domination relation guides this competition.

For detecting the correct number of Pareto optimal regions it is necessary to have, in the final population, only one solution per Pareto optimal region.

In this paper we consider the merging operator of Genetic Chromodynamics for implementing the population decreasing mechanism. Very close solutions are fused and population size decreases accordingly.

In the framework of this paper the merging operator is described as bellow:

(i) if two interval solutions overlap the shortest interval solution is discarded. Degenerated interval- solution included into non-degenerated interval-solutions are removed too;

(ii) if two degenerated solutions are closer than a fixed threshold, then the worst solution is discarded.

The merging operator is applied each time a new individual enters the population.

The method allows a natural termination condition. The algorithm stops when there is no improvement of the solutions for a fixed number of generations. Each solution in the last population supplies a Pareto optimal region contributing to the picture of Pareto optimal set.

6. Fine tuning

During the fine tuning stage sub-optimal solutions (regions) are removed. For this aim each continuous Pareto region is transformed into a discrete set. Discretized version is obtained considering points within each interval solution at a fixed step size.

Let us denote by ss the step size. Consider an interval solution $[x, y]$. From this solution consider the points xj fulfilling the conditions:

- $x_j = x + j \cdot ss,\ j = 0, 1, \ldots;$
- $x_j \leq y.$

These points represent the discretized version (denoted D) of the interval solution $[x, y]$.

Each point x_j within the interval solutions is checked. If a point from the discretized set D dominates the point x_j then x_j is removed from the Pareto interval $[x, y]$ together with a small neighboring region. The size of the removed region is equal with ss.

The intervals obtained after this stage are considered as the true Pareto sets.

7. Algorithm Complexity

The complexity of the proposed algorithm is low. Let m be the number of objectives and N the population size. The first stage requires

$$K_1 = O(m \cdot N \cdot IterationNumber)$$

operations.

Denote by I_{max} is the longest interval solution in the population. Consider a function

$$F : \mathcal{R} \times \mathcal{R} \rightarrow N.$$

Admit that F fulfills the following conditions:

(i) F is a linear function,
(ii) $F([a, a]) = 1$, for each $a \in (?\infty, \infty)$.

Second stage (fine tuning) requires

$$K_2 = O(N^2 \cdot F(I_{max})^2)$$

operations.

8. CPS Algorithm

Using the previous considerations we are ready to design a new multiobjective optimization algorithm.

The evolution stage of the CPS algorithm is outlined as bellow:

Algorithm **CPS**:
generates an initial population P(0) {all intervals are degenerated i.e. $x_k = y_k$}
$t = 0$;
while not (Stop_Condition) **do**

```
begin
    for each individual k in P(t) do
    begin
        p = random {generate a random number between 0 and 1}
        if p < 0.5
        then
            Left_offspr = MutateTowardsLeft(x_k);
            if dominate(Left_offspr, x_k)
            then
                if x_k = y_k
                then x_k = y_k = Left_offspr
            else
                if nondominated(Left_offspr, x_k)
                then x_k = Left_offspr;
        else
            Right_offspr = MutateTowardsRight(y_k);
            if dominate(Right_offspr, y_k)
            then
                if x_k = y_k
                then x_k = y_k = Right_offspr
            else
                if nondominated(Right_offspr, y_k)
                then y_k = Right_offspr;
        endif
    endfor
    t = t + 1
endwhile
```

Fine tuning part of CPS algorithm is obvious.

9. Numerical experiments

Several numerical experiments using CPOS algorithm have been performed. For all examples the detected solutions gave correct representations of Pareto set with an acceptable accuracy degree. Some particular examples are given below.

Example 1. Consider the functions $f_1, f_2 : [-10, 13] \to \mathcal{R}$ defined as

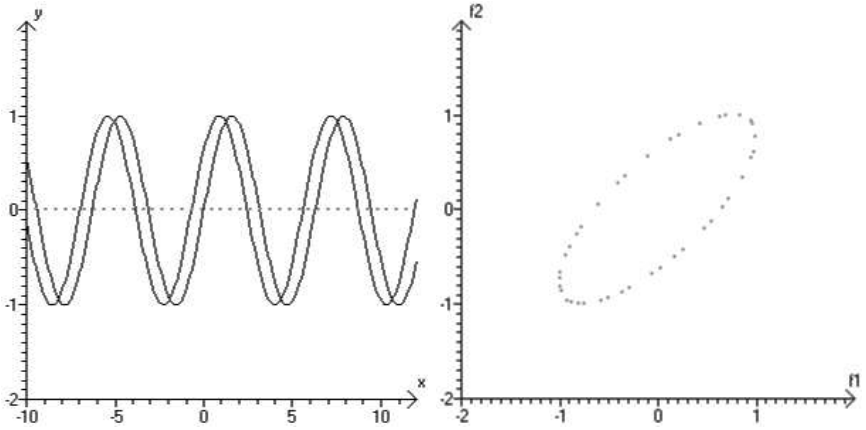
$$f_1(x) = sin(x),$$

$$f_2(x) = sin(x + 0.7).$$

Consider the multiobjective optimization problem:

$$\begin{cases} \text{optimize } f(x) = (f_1(x), f_2(x)) \\ \text{subject to } x \in [-10, 13] \end{cases}$$

The initial population is depicted in Figures 1(a) and 1(b).



(a) (b)

FIGURE 1. (a) Initial population represented within solution space; (b) Initial population represented within objective space

Consider the value

$$\sigma = 0.1$$

for the standard deviation of mutation operator. Solutions obtained after 3 generations are depicted in Figures 2(a) and 2(b).

The final population, obtained after 42 generations, is depicted in Figures 3(a) and 3(b).

The final population after the refinement stage is depicted in Figures 4(a) and 4(b).

Solutions in the final population are:

$$s_1 = [-8.47, -7.86],$$

$$s_2 = [-2.26, -1.56],$$

$$s_3 = [4.01, 4.69],$$

$$s_4 = [10.29, 10.99].$$

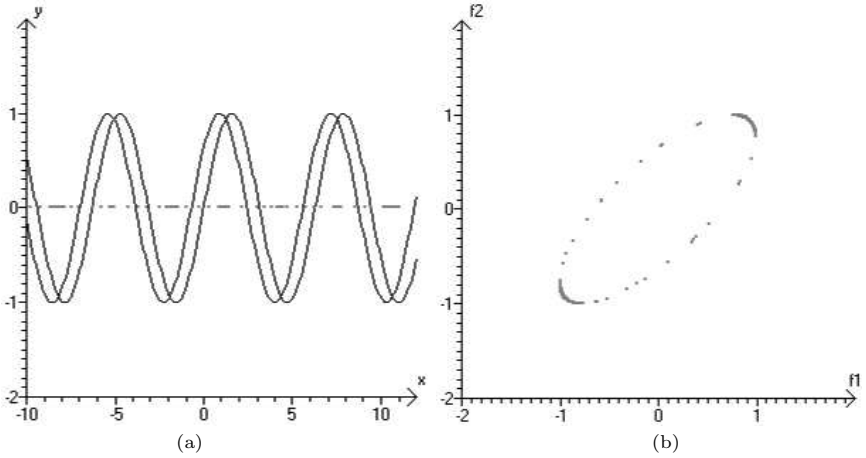


FIGURE 2. (a) Population obtained after 3 generations represented within solution space; (b) Population obtained after 3 generations represented within objective space

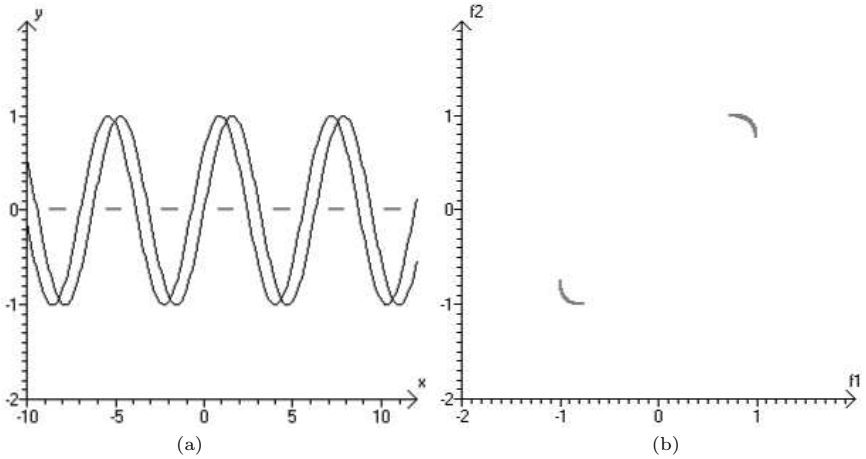


FIGURE 3. (a) Population obtained at convergence (after 42 generations) represented within solution space; (b) Population obtained at convergence (after 42 generations) represented within objective space

Example 2. Consider the functions $f_1, f_2 : [-10, 20] \to \mathcal{R}$ defined as

$$f_1(x) = sin(x),$$

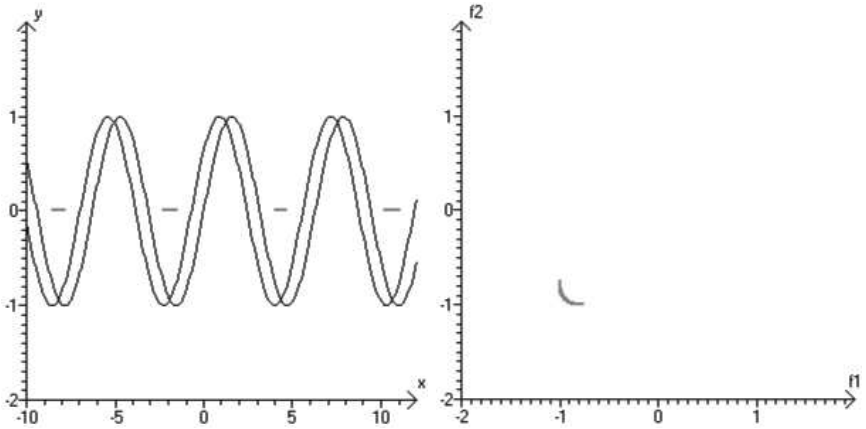


FIGURE 4. (a) Final population obtained after fine tuning stage represented within solution space; (b) Final population obtained after fine tuning stage represented within objective space

$$f_2(x) = 2 \cdot sin(x) + 1.$$

Consider the multiobjective optimization problem:

$$\begin{cases} \text{optimize } f(x) = (f_1(x), f_2(x)) \\ \text{subject to } x \in [-10, 20] \end{cases}$$

The initial population is depicted in Figures 5(a) and 5(b).

For the value

$$\sigma = 0.1$$

solutions obtained after 14 generations are depicted in Figures 6(a) and (b).

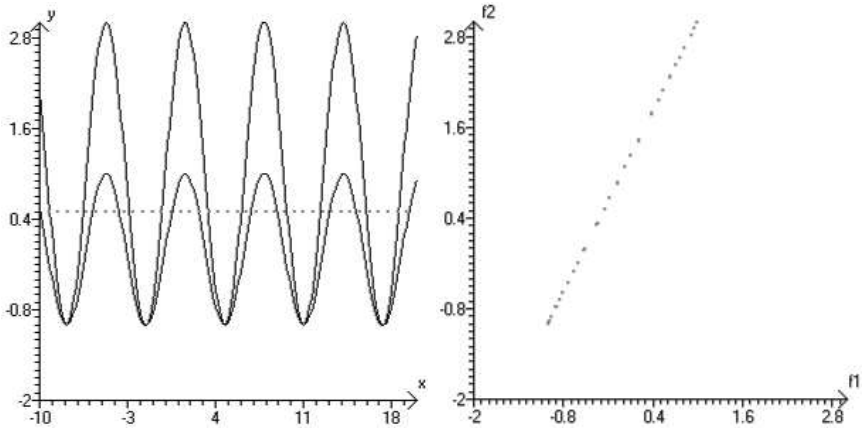
The final population, obtained after 70 generations, is depicted in Figures 7(a) and 7(b).

Example 3. Consider the functions $f_1, f_2 : [0, 24] \rightarrow \mathcal{R}$ defined as

$$f_1(x) = sin(x),$$

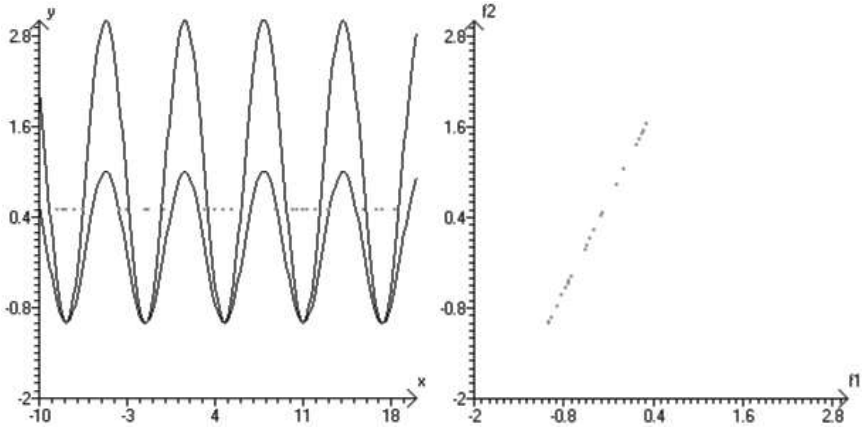
$$f_2(x) = \begin{cases} \frac{-4 \cdot x}{\pi} + 8 \cdot k, & 2 \cdot k \cdot \pi \leq x < 2 \cdot k \cdot \pi + \frac{\pi}{2}, \\ \frac{4 \cdot x}{\pi} - 4 \cdot (2 \cdot k + 1), & 2 \cdot k \cdot \pi + \frac{\pi}{2} \leq x < (2 \cdot k + 1) \cdot \pi, \\ \frac{-2 \cdot x}{\pi} + 2 \cdot (2 \cdot k + 1), & (2 \cdot k + 1) \cdot \pi \leq x < (2 \cdot k + 1) \cdot \pi + \frac{3 \cdot \pi}{2}, \\ \frac{2 \cdot x}{\pi} - 4 \cdot (k + 1), & 2 \cdot k \cdot \pi + \frac{3 \cdot \pi}{2} \leq x < 2 \cdot (k + 1) \cdot \pi + \frac{\pi}{2}. \end{cases}$$

where $k \in \mathbb{Z}^+$.



(a) (b)

FIGURE 5. (a) Initial population represented within solution space; (b) Initial population represented within objective space

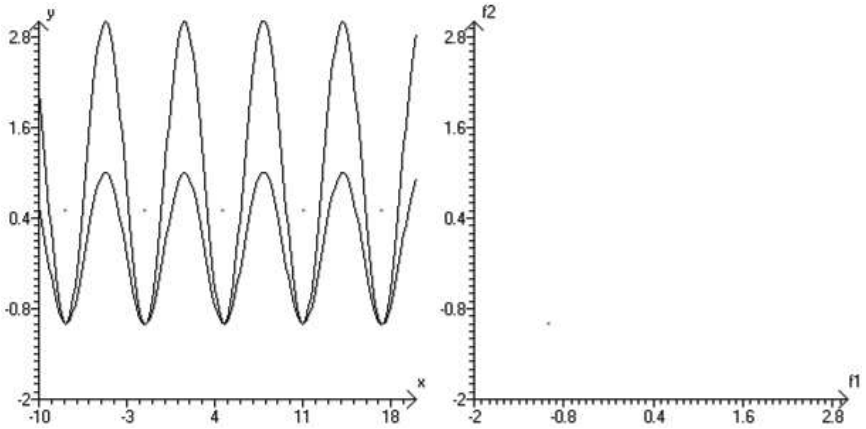


(a) (b)

FIGURE 6. (a) Population obtained after 14 generations represented within solution space; (b) Population obtained after 14 generations represented within objective space

Consider the multiobjective optimization problem:

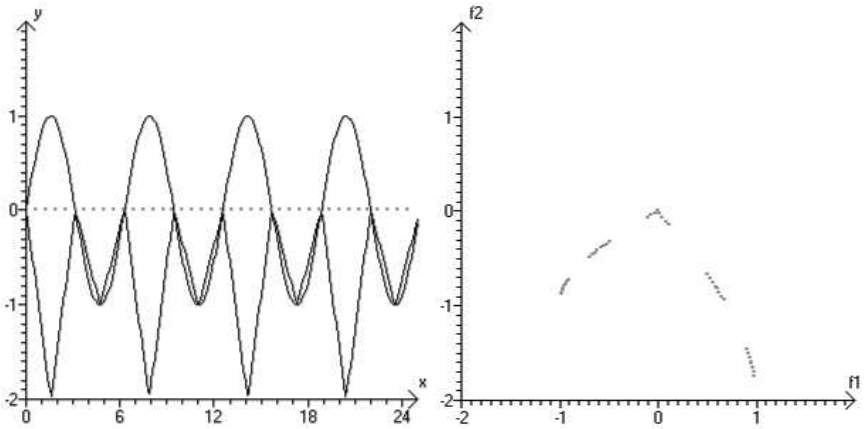
$$\begin{cases} \text{optimize } f(x) = (f_1(x), f_2(x)) \\ \text{subject to } x \in [0, 24] \end{cases}$$



(a) (b)

FIGURE 7. (a) Final population obtained after 70 generations represented within solution space; (b) Final population obtained after 70 generations represented within objective space

The initial population is depicted in Figure 8(a) and 8(b).



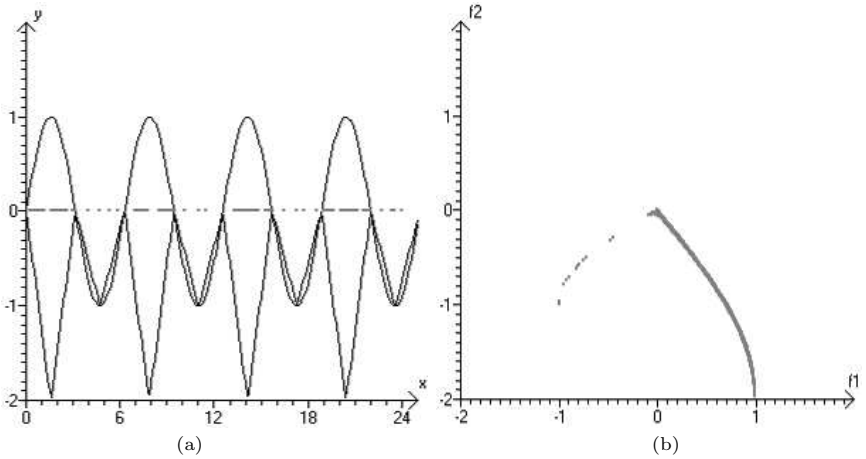
(a) (b)

FIGURE 8. (a) Initial population represented within solution space; (b) Initial population represented within objective space

For the value

$$\sigma = 0.1$$

solutions obtained after 4 generations are depicted in Figures 9(a) and 9(b).



(a) (b)

FIGURE 9. (a) Population obtained after 4 generations represented within solution space; (b) Population obtained after 4 generations represented within objective space

The final population, obtained after 60 generations, is depicted in Figures 10(a) and 10(b).

The final population after the refinement stage is depicted in Figure 11(a) and 11(b).

10. Concluding remarks and further research

A new evolutionary technique for solving multiobjective optimization problems involving one variable functions is proposed. A new solution representation is used. Standard search (variation) operators are modified accordingly. The proposed evolutionary multiobjective optimization technique uses only one population. This population consists of nondominated solutions already founded.

All known standard or recent multiobjective optimization techniques supply a discrete picture of Pareto optimal solutions and of Pareto frontier. But Pareto optimal set is usually non-discrete. Finding Pareto optimal set and Pareto optimal frontiers using a discrete representation is not a very easy computationally task (see [11]).

Evolutionary technique proposed in this paper supplies directly a continuous picture of Pareto optimal set and of Pareto frontier. This makes our approach

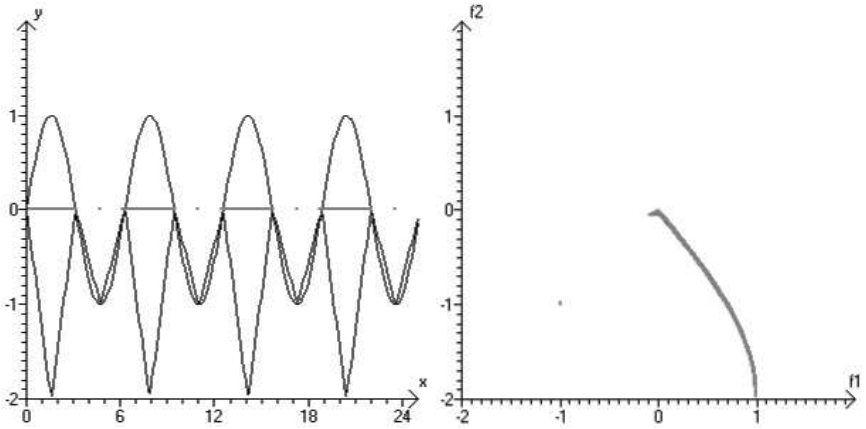


FIGURE 10. (a) Population obtained at convergence (after 60 generations) represented within solution space; (b) Population obtained at convergence (after 60 generations) represented within objective space

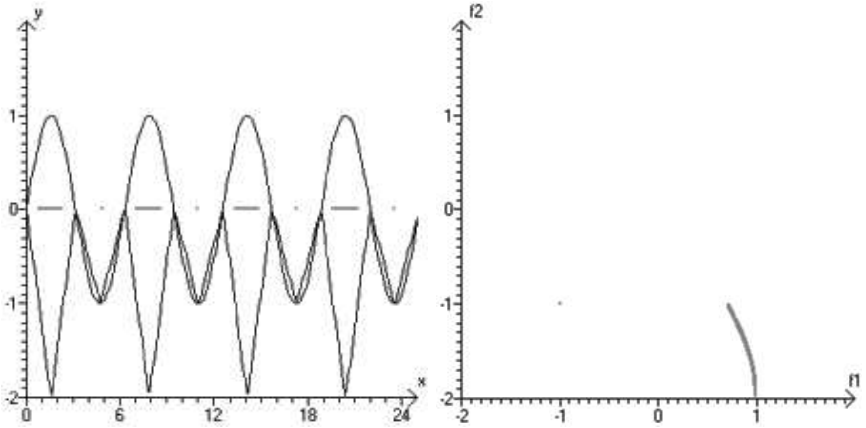


FIGURE 11. (a) Final population obtained after fine tuning stage represented within solution space; (b) Final population obtained after fine tuning stage represented within objective space

very appealing for solving problems where very accurate solutions detection is needed.

Another advantage is that CPS technique has a natural termination condition derived from the nature of evolutionary method used for preserving population diversity.

Experimental results suggest that CPS algorithm supplies correct solutions after few generations.

Further research will focus on the possibilities to extend the proposed technique to deal with multidimensional domains.

Another research direction is to exploit the solution representation as intervals for solving inequality systems and other problems for which this representation seems to be natural.

References

[1] C. A. C. Coello, A comprehensive survey of evolutionary-based multiobjective optimization techniques, Knowledge and Information Systems, 1(3), pp. 269–308, 1999.

[2] K. Deb, Multiobjective evolutionary algorithms: problem difficulties and construction of test problems, Evolutionary Computation, 7, pp. 205–230, 1999.

[3] D. Dumitrescu, B., Lazzerini, L.C. Jain, A. Dumitrescu, Evolutionary Computation, CRC Press, Boca Raton, FL, 2000.

[4] D. Dumitrescu, Genetic chromodynamics, Studia, Babes-Bolyai University, Ser. Informatica, 45, 2000, pp 39–50, 2000

[5] D. Dumitrescu, C. Grosan, M. Oltean, An evolutionary algorithm for detecting continuous Pareto regions, Studia Babes-Bolyai University, Ser. Informatica, 45, pp. 51–68, 2000.

[6] D. E. Goldberg, Evolutionary Algorithms in Search, Optimization and Machine Learning, Addison Wesley, Reading, 1989.

[7] J. Horn, N. Nafpliotis, Multiobjective optimization using niched pareto evolutionary algorithms, IlliGAL Report 93005, Illionois Evolutionary Algorithms Laboratory, University of Illinois, Urbana Champaingn.

[8] J. Horn, N. Nafpliotis, D. E. Goldberg, A niche Pareto evolutionary algorithm for multiobjective optimization, Proc. 1st IEEE Conf. Evolutionary Computation, Piscataway, vol 1, pp. 82–87, 1994.

[9] J. D. Schaffer, Multiple objective optimization with vector evaluated evolutionary algorithms, Evolutionary Algorithms and Their Applications, J.J. Grefenstette (Ed.), Erlbaum, Hillsdale, NJ, pp. 93–100, 1985.

[10] N. Srinivas, K., Deb, Multiobjective function optimization using nondominated sorting evolutionary algorithms, Evolutionary Computing, 2, pp. 221–248, 1994.

[11] D.A.V. Veldhuizen, Multiobjective Evolutionary Algorithms: Classification, Analyses and New Innovations, Ph.D Thesis, Graduated School of Engineering of the Air Force Institute of Technology, Air University, 1999.

[12] D.A.V. Veldhuizen, G. B. Lamont, Multiobjective evolutionary algorithms: analyzing the state- of-the-art, Evolutionary Computation, 8, pp. 125–147, 2000.

[13] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: A comparative study and the strength Pareto approach, IEEE Trans on Evolutionary Computation, 3, pp. 257–271, 1999.

[14] E. Zitzler, Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, Doctoral Dissertation, Swiss Federal Institut of Technology Zurich,Tik-Schriftenreihe nr. 30, 1999.

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

E-mail address: ddumitr|cgrosan|moltean@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

CODING OBJECTS RELATED TO CATALAN NUMBERS

ANTAL BEGE AND ZOLTÁN KÁSA

Abstract. A coding method using binary sequences is presented for different computation problems related to Catalan numbers. This method proves in a very easy way the equivalence of these problems.

1. Introduction

The Catalan numbers, named after the french mathematician E. C. Catalan, defined as

$$C_n = \frac{1}{n+1}\binom{2n}{n},$$

are as known as the Fibonacci numbers. These numbers arise in a lot of combinatorial problems as the number of some objects. The Catalan number C_n describe, among other things,

- the number of binary trees with n nodes,
- the number of ways in which parantheses can be placed in a sequence of $n+1$ numbers to be multiplied two at a time,
- the number of well-formed reverse Polish expressions with n operands and $n+1$ operators,
- the number of paths in a grid from $(0,0)$ to (n,n), increasing just one coordinate by one at each step, without crossing the main diagonal,
- the number of n-bit sequences that the number of 1s never exceeds the number of 0s in each position from left to right,
- the number of ways you can draw non-crossing segments between $2n$ points on a circle in the plane,
- the number of sequences $(x_1, x_2, \ldots, x_{2n})$, with $x_i \in \{-1, 1\}$ for all i between 1 and $2n$ and having the following properties for all partial sums: $x_1 \geq 0,\ x_1+x_2 \geq 0,\ \ldots,\ x_1+x_2+\ldots+x_{2n-1} \geq 0,\ x_1+x_2+\ldots+x_{2n} = 0$,
- the number of ways a polygon with $n+2$ sides can be cut into n triangles,
- the number of frieze pattern with $n+1$ rows,

2000 *Mathematics Subject Classification.* 11B75, 68P30, 68R05.

1998 *CR Categories and Descriptors.* G.2.1 [**Mathematics and Computing**]: Discrete Mathematics – *Combinatorics, Counting problems.*

Research supported by Sapientia Foundation: `http://www.sapientia.ro`.

- the number of mountain ranges you can draw using n upstrokes and n downstrokes,
- the number of ways n votes can come in for each of two candidates in an election, with the first never behind the second.

The Catalan numbers are the solution of the following recurrence equation:

$$C_{n+1} = C_0C_n + C_1C_{n-1} + \ldots + C_nC_0 \quad \text{for } n \geq 0, \text{ with } C_0 = 1.$$

Another recurrence equation for the Catalan numbers is:

$$(n+2)C_{n+1} = (4n+2)C_n \quad \text{for } n \geq 0, \text{ with } C_0 = 1.$$

The generating function of these numbers is

$$\sum_{n\geq 0} C_n z^n = \frac{1-\sqrt{1-4z}}{2z},$$

which can be obtained from the first recurrence equation given above using generating function techniques (see e. g. [5] for computing the number of n-node binary trees).

Let $C(z) = \sum_{n\geq 0} C_n z^n$ be the generating function corresponding to the Catalan numbers. By the recurrence equation this function satisfy the following equation:

$$zC^2(z) = C(z) - 1, \quad \text{with } C(0) = 1.$$

From this

$$C(z) = \frac{1-\sqrt{1-4z}}{2z}$$

results. By developping in series we will get the followings:

$$C(z) = \frac{1}{2z}\left(1-\sqrt{1-4z}\right) = \frac{1}{2z}\left(1 - \sum_{n\geq 0}\binom{\frac{1}{2}}{n}(-4z)^n\right) =$$

$$= \sum_{n\geq 0}\binom{\frac{1}{2}}{n+1}(-1)^n 2^{2n+1} z^n = \sum_{n\geq 0}\frac{1}{n+1}\binom{2n}{n},$$

and from this the formula for Catalan numbers results.

2. The Encoding

We shall present here an encoding method of objects whose number is a Catalan number. Each object will be codified by a binary sequence in which the number of 0s is equal to the number of 1s, and from the beginning to any position of the sequence, the number of 1s never exceeds the number of 0s. Let us call these sequences *Catalan sequences*.

The mathematical definition of the Catalan sequence is given below. Let us denote by $n_1(u)$ the number of 1s and by $n_0(u)$ the number of 0s in the sequence

u. A sequence $u = u_1u_2 \dots u_{2n}$, with $u_i \in \{0,1\}$ for $i = 1, 2, \dots, 2n$, is a *Catalan sequence* if

$$\begin{aligned} n_1(u_1u_2 \dots u_i) &\leq n_0(u_1u_2 \dots u_i) \quad \text{for} \quad i = 1, 2, \dots, 2n \\ n_1(u) &= n_0(u) \end{aligned}$$

Our coding method is different from the one given in [8] for binary trees.

There are a lot of papers which deal with the Catalan numbers, in references we give only a few of them.

2.1. **Encoding of binary trees.** The encoding of a binary tree is the following: when a vertex has only one descendant, we put the sequence 01 for a single left edge, 10 for a single right edge, and 00 for the left edge resp. 11 for the right edge when there are two descendants. We complete the resulting sequence with 0 at the beginning and 1 at the end. The encoding is made using a preorder traversal of the binary tree. In the case of the binary trees with 3 nodes we shall have the encoding in Fig. 1.

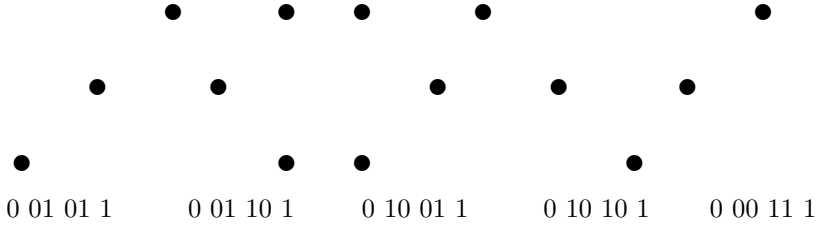


Fig. 1. Encoding of binary trees

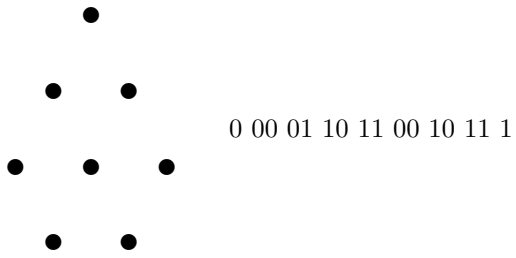


Fig. 2. A more complex example

A more complex example, when the preorder traversal can be easily seen, is given in Fig. 2.

The encoding algorithm for a binary tree B is given as follow in a pseudocode-form. Let us denote by $\emptyset$ the empty binary tree (with no vertices). The **put** statement puts its argument in the resulting output sequence.

Algorithm for encoding a binary tree
put 0
procedure encoding (B):
  Let B_L be the left and B_R the right subtree of B
  if $B_L \neq \emptyset$ and $B_R = \emptyset$ **then**
    put 01
    call encoding (B_L)
  if $B_L = \emptyset$ and $B_R \neq \emptyset$ **then**
    put 10
    call encoding (B_R)
  if $B_L \neq \emptyset$ and $B_R \neq \emptyset$ **then**
    put 00
    call encoding (B_L)
    put 11
    call encoding (B_R)
end procedure
put 1

For an empty binary tree the procedure has no effect. The proof that the resulting sequence is a Catalan sequence is immediately by the above algorithm.

2.2. **Encoding of paths in grid.** We shall put 0 for a horizontal unit of the path and 1 for a vertical one. The resulting sequence is a Catalan one because the path never cross the main diagonal of the grid.

In the case of a grid 3×3 the following paths and codes results (see Fig. 3).

001011 001101 010011 010101 000111

Fig. 3. Encoding of paths in grid

2.3. **Encoding of expressions with multiplications.** To encode expressions we first attach to each expression for multiplication a binary tree by a very simple method. If we multiple a by b, this yields a binary tree with a root and two descendant nodes a and b. A multiplication of two expressions yields a binary tree with two subtrees which are the binary trees corresponding to the two expressions. In the resulting binary tree each internal nodes has exactly two descendants. Such trees are called *extended binary trees*. To encode an extended binary tree we shall omit all leaves (with of course the corresponding edges) in the tree corresponding to the multiplication expression and use the encoding method presented before

for the resulting binary tree. For $n = 4$ we shall have the expressions and the corresponding extended binary trees in Fig. 4. If we omit all leaves with the adjacent edges in these extended trees the binary trees and the corresponding encoding result.

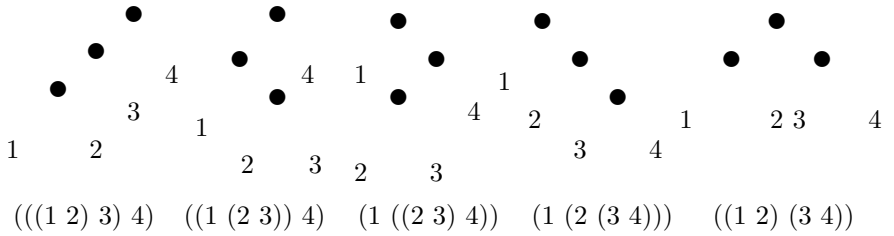


Fig. 4. Encoding of multiplications

2.4. **Encoding of sequences.** We encode sequences $(x_1, x_2, \ldots, x_{2n})$, with $x_i \in \{-1, 1\}$ for all i between 1 and $2n$ and having the following properties for all partial sums: $x_1 \geq 0,\ x_1 + x_2 \geq 0,\ \ldots,\ x_1 + x_2 + \ldots + x_{2n-1} \geq 0,\ x_1 + x_2 + \ldots + x_{2n} = 0$. We shall code -1 in the sequence by 1 and 1 by 0. It is easy to see that in any positions the number of 1s never exceeds the number of 0s, and they are equals in the sequence (because the sum of all $2n$ elements is equal to 0), so the resulting sequence is a Catalan one. For example:

1,	1	1,	−1,	−1,	−1	coded by:	000111
1,	1	−1,	1,	−1,	−1	coded by:	001011
1,	1	−1,	−1,	1,	−1	coded by:	001101
1,	−1	1,	1,	−1,	−1	coded by:	010011
1,	−1	1,	−1,	1,	−1	coded by:	010101

2.5. **Encoding of segments.** If we have $2n$ points on a circle in the plane and n non-crossing segments between them, the encoding is the following: Let us mark the points clockwise on the circle with numbers from 1 to $2n$. For a segment between i and j $(i < j)$ put 0 in the i^{th} position and 1 in the j^{th} position in the code sequence. For $n = 3$ see Fig. 5. It is easy to see that the resulting sequence is a Catalan one.

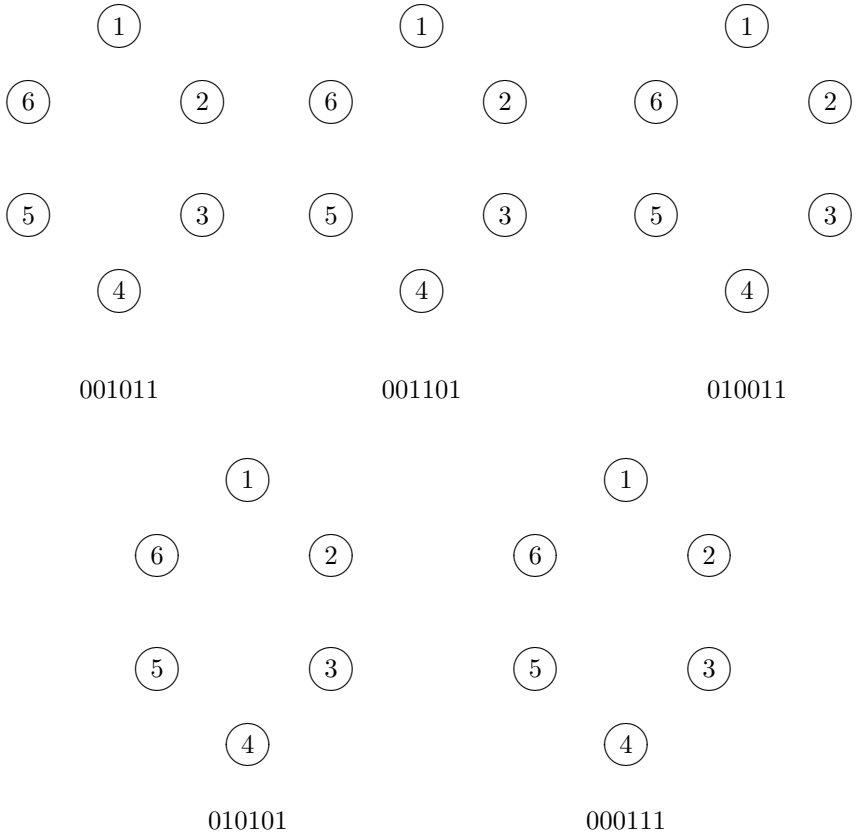


Fig. 5. Encoding of segments

2.6. **Encoding of reverse Polish expressions.** We shall code each operand by 0 and each operator by 1, and add at the end of the resulting sequence an 1. For example, if we have the reverse Polish expression $aaa \times a \times \times$ — which corresponds to the expression $(a \times ((a \times a) \times a))$ — the resulting code is 00010111.

2.7. **Encoding of polygons.** The polygon is divided into triangles. We consider one node in each triangle, and one outside of each side of the polygon. Join by an edge two nodes if the corresponding triangles (or a triangle and the outside of the polygon) have a side in common. We shall get a tree, on which the encoding will be made. If we mark one side of the polygon and the corresponding edge of the tree, and eliminate all edges from the tree that have an endpoint as a leave, we shall get a binary tree (the root will be the node which is adjacent with the marked edge). The exemplification will be made for $n = 3$ (pentagon). The marked side is AB. (See Fig. 6)

A B A B A B A B A B

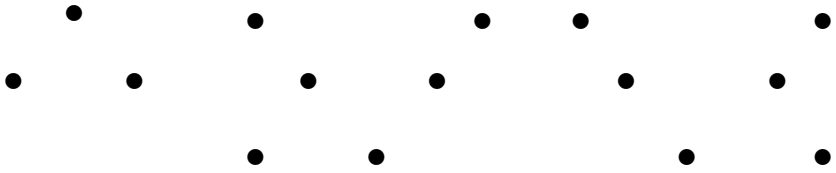


Fig. 6. Encoding of polygons

3. The Decoding

If we have a Catalan sequence, from this the corresponding object can be easily obtained. Let us consider for exemplification the sequence 00010111.

If we want to obtain the corresponding binary tree, we shall omit the first 0 and the last 1. The subsequence 00 is for a left edge (in a stack we shall keep its position), the following subsequence is 10 corresponding to a single right edge, the remaining subsequence 11 is a right edge (corresponding to the edge kept in the stack). The binary tree obtained is in Fig. 7.a.

For the segments we search the first subsequence 01, trace the corresponding segment, omit it from the sequence and continue with the remaining sequence (keeping the original positions) (Fig. 7.b).

For the multiplication we first draw the corresponding binary tree (Fig. 7.a), complete it to having two descendants for each node. The resulting extended binary tree give us the order of multiplications (Fig. 7.c).

The path in the grid is obtained immediately: we draw a horizontal unit segment for each 0 and a vertical one for each 1 (Fig. 7.d).

From these examples general algorithms to obtain related objects from the Catalan sequences can be easily given. We shall describe only the algorithm to obtain a binary tree from a Catalan sequence.

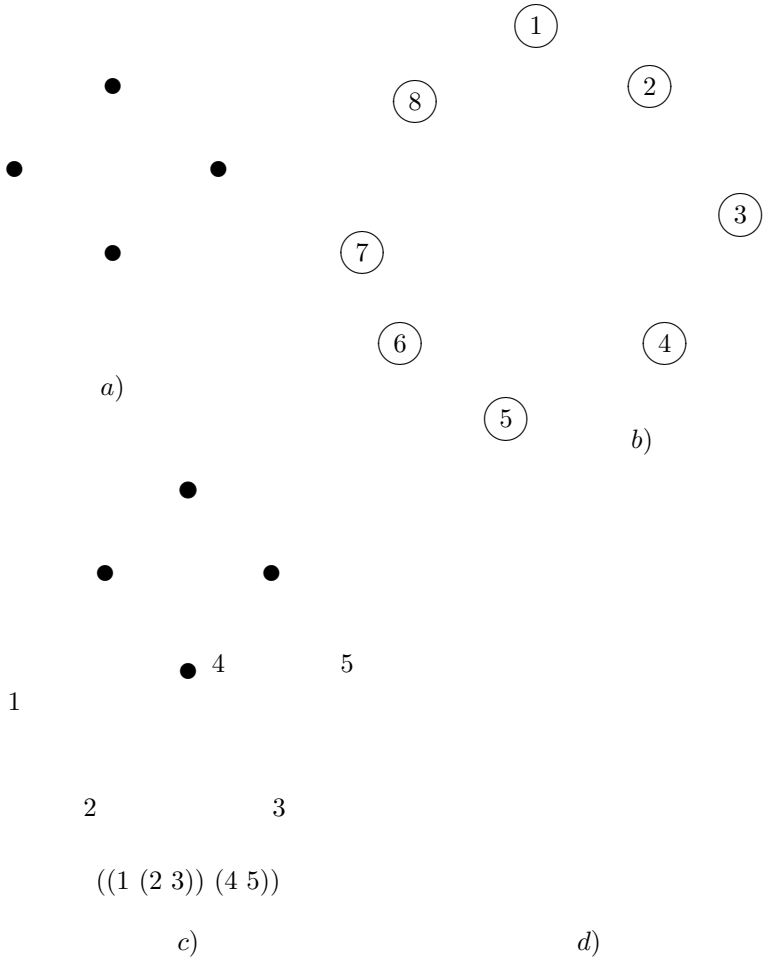


Fig. 7. Decoding

The following recursive algorithm is to decode a Catalan sequence in a binary tree. This algorithm is valid only for correct Catalan sequences. The **get** statement gets the next two digits from the sequence. We shall use the notion of current vertex to denote a vertex from which an edge is drawn. After drawing an edge from the current vertex the adjacent new vertex will be the current one.

Algorithm to decode a Catalan sequence into a binary tree

Input a Catalan sequence
Output a binary tree
delete 0 from the beginning and 1 from the end of the input sequence,
and draw a vertex (the root of the tree) as current vertex
procedure decoding (c):
get ab
delete ab from c
if $ab = 01$ **then**
draw a left edge from the current vertex
call decoding (c)
if $ab = 10$ **then**
draw a right edge from the current vertex
call decoding (c)
if $ab = 00$ **then**
put in the queue the position of the current vertex
draw a left edge from the current vertex
call decoding (c)
if $ab = 11$ **then**
get from the queue the position of a vertex,
this will be the current vertex
draw a right edge from the current vertex
call decoding (c)
end procedure

4. Conclusions

Our presentation give a uniform method to encode objects whose number is a Catalan number. The resulting code is a so-called Catalan sequence formed of equal number of binary digits 0 and 1, in which the number of 1s never exceeds the number of 0s from left to right. This method is important, beside the easy handling, because coding an object in a Catalan sequence and after decoding it in another kind of object, the equivalence of these problems can be easily seen. To prove that the number of objects in a class is a Catalan number it is enough to use the encoding method to obtain a Catalan sequence.

References

[1] Cofman, J., Catalan numbers for the classroom?, *Elemente der Mathematik*, **52** (1997), 108–117.
[2] Dickau, R. M., Catalan numbers, `http://forum.swarthmore.edu/advanced/robertd/catalan.html`
[3] Dershowitz, N., Zaks, S., Enumeration of ordered trees, *Discrete Mathematics*, **31** (1980), 9–28.

[4] Egleton, R. B., Guy, R. K., Catalan strikes again! How likely is a function to be convex?, *Mathematics Magazine*, **67**, 4 (1988), 211–219.
[5] Knuth, D. E., The art of computer programming, Volume 1 (Fundamental algorithms), Addison-Wesley, 1981 (Second edition).
[6] Singmaster, D., An elementary evaluation of the Catalan numbers, *Amer. Math. Monthly*, **85**, 5 (1978), 366–368.
[7] Vilenkin, N. I., Combinatorica (in Russian), Nauka Moscow, 1969.
[8] Weisstein, E. W., Catalan numbers, `http://mathworld.wolfram.com/CatalanNumbers.html`
[9] ***, Information on binary trees, `http://www.theory.csc.uvic.ca/~cos/inf/tree/BinaryTrees.html`

Babeş-Bolyai University, Faculty of Mathematics and Computer Science,
RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania
E-mail address: `bege@math.ubbcluj.ro, kasa@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 1, 2001

INTENSIONAL LOGIC TRANSLATION FOR QUANTITATIVE NATURAL LANGUAGE SENTENCES

ADRIAN ONEŢ, DOINA TĂTAR

Abstract. The performance of some natural language processing tasks improves if semantic processing is involved. Moreover, some tasks (database query) cannot be carried out at all without semantic processing. The first semantic description was developed by Montague and all later approach to semantic in the frame of discourse representation theory follow Montague in using more powerful logic language. The present paper is a contribution in treatment of quantitative natural sentences.

1. Introduction

At present, there doesn't exists a general theory of the semantics of natural language. A rigorous analysis of natural language can't be realized without the intensional logic introduced in computational linguistic by Montague. The intensional logic is a further development from the model provided by first order logic. In his semantic analysis of a sentence, Montague distinguishes two elements: intension (or sense) and extension (or reference). The intension of a sentence is even the proposition it expresses, and the extension is its truth value. An extensional logic can only assign truth value to sentences while an intensional logic can, in addition, assign a meaning to these sentences. Moreover, Montague saw in type theory a powerful system which could correspond to the system of syntactic categories of a natural language.

The following sections aim at acquainting the reader with the fundamentals of intensional logic. The last section introduces some proposals in treatment of quantitative natural sentences.

2. Intensional logic

The intensional logic [7] contains, besides new ones, all the concepts in a predicate logic of first order, L_1.

2000 *Mathematics Subject Classification.* 68U35.

1998 *CR Categories and Descriptors.* I.2.1 [**Computing Methodologies**]: Artificial Intelligence – *Appplications and Expert Systems*.

2.1. **A type-theoretic version of the language L_1.** Let t and e be two basic symbols that respectively represent 'truth' and 'entity'. The set of types is recursively defined as follows:

Definition:

- t and e are types (base);
- If a and b are types, then $\langle a, b\rangle$ is a type;
- All types are obtained by applying the base and induction rule a finite number of times.

Example:
$\langle e, \langle t, e\rangle\rangle$ is a type.

In the following the symbol D_a will represent the set of all denotations for the expressions of type a, with respect to a given interpretation domain A. The sets D_a are recursively defined as:

- $D_e = A$;
- $D_t = 0, 1 \, or \, T, F$;
- $D_{\langle a,b\rangle} = \, the \, set \, of \, functions \, from \, D_a \, to \, D_b$.

If the type is $\langle a_1, \langle a_2, \langle \cdots \langle a_n, b\rangle \cdots ..\rangle$ then $D_{\langle a_1,\rangle a_2,\langle \cdots \langle a_n,b\rangle \cdots ..\rangle}$ is the set of functions from $D_{a_1} \times D_{a_2} \cdots \times D_{a_n}$ to D_b.

The syntactic categories are set of expressions in this logic language. We shall assign to each of these categories a syntactic label (a type). The rule of this labeling is denoted *cancellation rule* and can be stated formally as follows [7]:

If α is an expression of type $\langle a, b\rangle$ and β is an expression of type a then the juxtaposition $\alpha(\beta)$ is of the type b.

Let us remark that the rule is conform with the denotation of types: *If α is a function from D_a to D_b and β is an element from D_a then juxtaposition $\alpha(\beta)$ is an element from D_b.*

We can now define the language L_1 *with types*, denoted L_t, as follows:

Definition:

- The type of the constants c_i, and variables x_j is e;
- The type for one-place predicate constant P_i is $\langle e, t\rangle$ (a function from D_e to $D_t = \{T, F\}$);
- The type for two-place predicate constant P_i is $\langle e, \langle e, t\rangle\rangle$ (a function from $D_e \times D_e$ to $D_t = \{T, F\}$);
- The type for n-place predicate constant P_i is $\langle e, \langle e, \cdots \langle e, t\rangle \cdots \rangle\rangle$, with n occurrences of e; (a function from $D_e \cdots \times D_e$ to $D_t = \{T, F\}$);
- If α is an expression of type $\langle a, b\rangle$ and β is an expression of type a then the juxtaposition $\alpha(\beta)$ is of the type b.
- The type of the formulas is t;
- The type of the connective $\neg$ is $\langle t, t\rangle$;
- The type of the connectives $\wedge, \vee, \rightarrow$ is $\langle t, \langle t, t\rangle\rangle$;

- If A is a formula (of type t) and x is a variable (of type e), then $[\forall xA]$, $[\exists xA]$ are of type t.

2.2. **Lambda calculus.** The language L_t will be expanded by adding the lambda operator (λ-operator). The lambda calculus was introduced in the linguistic's community by Montague, and in the logic by Alonzo Church (1941). Replacing the well known notation for the sets as: $\{x|x\ has\ a\ certain\ propriety\}$ the following notation is used: $\lambda x[formula\ containing\ x]$. The expression

$$\lambda x[formula\ containing\, x]$$

is called λ-expression or λx-abstraction.

The type of a λ-expression as above is $\langle e, t\rangle$. If we shall combine this expression with a constant (of type e), from the cancellation rule results a formula (of type t. This process is named λ-conversion and may be written as follows:

$$\lambda x[\cdots x\cdots](\alpha) = [\cdots \alpha \cdots].$$

In the formula $[\cdots \alpha \cdots]$ each free occurrence of x is replaced with α, the result is $[\cdots \alpha \cdots]$.

Most of the time, in the language L_t, x and A can be of types more general than e and t. The rule is:

If α is an expression of the type a and x is a variable of the type b, then $\lambda x[\alpha]$ is an expression of type $\langle b, a\rangle$.

For example the type of x can be $\langle e, t\rangle$, and in this case λ- operator make a λ abstraction after a predicate (not a variable).

Montague gave some examples of λ-operator in natural language sentences in English [7]. Let us consider two sentences: *Every student walks* and *Every students reads*. Their usual translation in predicate logic is:

$\forall x(S(x) \to W(x))$ and $\forall x(S(x) \to R(x))$.

These sentences are instances of a more general sentence whose translation is a second-order logic formula, i.e. they are λ-conversions of the λ-expression:

$$\lambda Y[\forall x(S(x) \to Y(x))].$$

The first conversion is

$$\lambda Y[\forall x(S(x) \to Y(x))](W)$$

and the second one is

$$\lambda Y[\forall x(S(x) \to Y(x))](R).$$

Therefore, the λ -expression $\lambda Y[\forall x(S(x) \to Y(x))]$ will have the type $\langle\langle e, t\rangle, t\rangle$ and it is equivalent to the English sentence *every student*.

Analogously, we can obtain a λ expression for *every*: if *every student* is $\lambda Y[\forall x$ $(S(x) \to Y(x))]$, then this expression may be seen as conversion of the more general expression: $\lambda Z[\lambda Y[\forall x(Z(x) \to Y(x))]]$ for S, that means $\lambda Z[\lambda Y[\forall x(Z(x) \to$

$Y(x))]](S)$. So *every* can be translated in $\lambda Z[\lambda Y[\forall x(Z(x) \rightarrow Y(x))]]$ with the type $\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$.

Analogously, *some student* or *a student* is translated in expression $\lambda Y[\exists x(S(x) \wedge Y(x))]$ with the type $\langle\langle e,t\rangle,t\rangle$, and *some*, *a*, *an*, in expression $\lambda Z[\lambda Y[\exists x(S(x) \wedge Y(x))]]$ with the type $\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$.

2.3. **Intensionality.** In his theory Montague make a distinction between the *sense* (intention) of an expression and $reference$ (extension): the $reference$ of an expression corresponds to semantic (truth) value of this expression, the *sense* corresponds to the meaning of the expression. The distinction between *sense* and $reference$ is important when the operators $\Box$ (necessarily) and $\Diamond$ (possibly) are used. For example, $\Box A$ cannot be described as function of the references of its parts ($\Box$) and A) but can be described as a function of the senses of these parts. The intensionality in natural language is induced by propositional attitude verbs as: *think, believe, regret*, etc.

In the following we shall denote by α^i and α^e, *intension* and *extension* of an expression α, respectively. There is a *cancellation rule* ^{e,i} very important in simplification of the expressions produced by the translation of sentences in natural language: $\alpha^{i,e} = \alpha^{e,i} = \alpha$.

The expressions for determinants *every* and $a, some$ will be, considering the intensionality:

$$\lambda Z[\lambda Y[\forall x(Z^e(x) \rightarrow Y^e(x))]]$$

respectively

$$\lambda Z[\lambda Y[\exists x(Z^e(x) \wedge Y^e(x))]].$$

As in the case of type-theoretic version of the language L_1, L_t, the syntax contains recursive definitions of the types. The basic types are, as above t, e . Additionally, is used the symbol s for *sense*, which will allow to associate with every type a a new type $\langle s,a\rangle$.

Observation:

The expression of type $\langle s,a\rangle$ have as extension, intension of expression of type a.

The formation rules for types are as follows:

Definition:

- t and e are types;
- If a and b are types, then $\langle a,b\rangle$ is a type;
- If a is a type, then $\langle s,a\rangle$ is a type;
- All types are obtained by applying the induction rules of a finite number of times.

3. Montague's grammar

In his paper "The proper treatment of quantification in ordinary Englis" (1973), Montague formulated a syntax and a semantics for natural language (NL) which has the same rigor and precision as the syntax and semantics of formal languages. He introduced for NL a categorial grammar with the name PTQ, from the name of the paper. This grammar PTQ uses 11 types (syntactic categories) of words: *sentences*, *intransitive verbs, term(en)s (NP, proper nouns, pronouns,etc), transitive verbs, adverbs of type VP, sentence adverbs, common nouns, prepositions, sentence complement verbs, (believe, assert, etc, used with "that"), infinitive complement verbs (try, wish, etc)* , *determiners* (*every, the, a, an*).

In the categorial grammar PTQ of Montague, the type (syntactic categorie) is defined as [7, 8, 3]

- t is a syntactic category, of the expressions to which a truth value can be assigned, i.e. the category of sentences;
- e is a syntactic category, of the expressions to which entities can be assigned;
- if A is a syntactic category, and B is a syntactic category then A/B is a syntactic category.

The *rule of the categorial cancellation* is the following: *if an expression of category A/B combines with an expression of category B then is proceeded an expression of category A.*

The nine categories above have "predefinited" types: for example, intransitive verbs are of type t/e, terms have the type $t/(t/e)$, "believes that" have the type $(t/e)/t$, etc (as in the bellow figure).

If we would analyses how the sentences *John walks* or *John believes that Mary walks*, are obtained by the categorial cancellation rule we shall obtain sentences of type t. Indeed, denoting by + juxtaposition, we have:

$$John_{t/(t,e)} + walks_{t/e} = (John\ walks)_t$$

$$John_{t/(t,e)} + (believes\, that)_{(t/e)/t} + (Mary\, walks)_t = (John\, believes\, that\, Mary\, walks)_t.$$

Definition:

A sentence is any recursive combination of basic expressions that produces, after a finite number of applications of the categorial cancellation rule, an expression of category t.

In Table 1 we will present the syntactic categories of the PTQ grammar.

3.1. **Syntactic rules in the PTQ categorial grammar.** In the following we will use the notation: if A is a syntactic category then B_A is the set of words in the dictionary(lexical entries) of category A and P_A is the set of groups of words of category A. For $A = t$, B_t is the empty set.

The first of the syntactic rules is the following:

Name	Categorial definition	Denotation	Example
t	$-$	$sentence$	$-$
IV	t/e	$VP; intrans.\, verb$	$run, walk, talk$
$T(term)$	$t/IV or\ t/(t/e)$	$NP; proper\, name$	$John, Mary$
TV	$IV/Tor(t/e)/(t/(t/e))$	$transitive\ verb$	$find, eat, love$
IAV	IV/IV	$VP\, adverb$	$rapidly, slowly$
CN	$t//e$	$common\, noun$	$man, woman$
SA	t/t	$sentence\, adverb$	$necessarily$
$Prep$	IAV/T	$preposition$	$in, about$
SCV	IV/t	$sentence\, compl.\, verb$	$believe, assert$
ICV	$IV//IV$	$infinitive\, compl.\, verb$	$try, wish$
DET	T/CN	$determiner$	$every, the, a, an$

TABLE 1. The syntactic categories of the PTQ grammar

- S1. For each syntactic category A, the set B_A is included in P_A.
  All the syntactic rules forming complex expressions have the general form:
- Si: If $\alpha \in P_{A/B}$ and if $\beta \in P_B$ then $F_i(\alpha, \beta) \in P_A$.

Some examples of the rules proposed by Montague are given hereafter :

- S2 (Complex expressions of category Term):
  If $\alpha \in P_{T/CN}$ and if $\beta \in P_{CN}$ then $F_2(\alpha, \beta) \in P_T$.
  The function $F_2(\alpha, \beta)$ is $\alpha^*\beta$ where α^* is α except in the case where α is a and the first word in β begins with a vowel; in this case $\alpha^* = an$.
- S4 (Complex expressions of category t, sentences):
  If $\alpha \in P_T$ and if $\beta \in P_{IV}$ then $F_4(\alpha, \beta) \in P_t$.
  The function $F_4(\alpha, \beta)$ is $\alpha\beta^*$ where β^* is the result of replacing the first verb from β by its third person singular present form.

The rules introduced by Montague permit the translation in *de dicto* (non-specific) mode and also in *de re* (specific) mode for a sentence.

He first defined a correspondence between syntactic category and the types in the form of a function f.

Definition:

- $f(t) = t$;
- $f(CN) = f(IV) = \langle e, t\rangle$;
- $f(A/B) = \langle\langle s, f(B)\rangle, f(A)\rangle$ for all syntactic category A and B.

The correspondence between syntactic categories and types is indicated in Table 2.

Beside the types, the expressions themselves are translated. Let us observe first that basic expressions (lexical entries) are translated in constants of type $\langle e, t\rangle$

Syntactic category	Type
t	t
IV	$\langle e,t\rangle$
CN	$\langle e,t\rangle$
$T(term) = t/IV$	$\langle\langle s,\langle e,t\rangle\rangle,t\rangle$
$TV = IV/T$	$\langle\langle s,\langle\langle s,\langle e,t\rangle\rangle,t\rangle\rangle,\langle e,t\rangle\rangle$
$IAV = IV/IV$	$\langle\langle s,\langle e,t\rangle\rangle,\langle e,t\rangle\rangle$
T/CN	$\langle\langle s,\langle e,t\rangle\rangle,\langle\langle s,\langle e,t\rangle\rangle,t\rangle\rangle$
t/t	$\langle\langle s,t\rangle,t\rangle$
$SCV = IV/t$	$\langle\langle s,t\rangle,\langle e,t\rangle\rangle$
$ICV = IV//IV$	$\langle\langle s,\langle e,t\rangle\rangle,\langle e,t\rangle\rangle$
IAV/T	–

TABLE 2

Formal expression	Associated type
P	$\langle s,\langle e,t\rangle\rangle$
P^e	$\langle e,t\rangle$
j	e
$P^e(j)$	$t(categorial\ cancellation\ rule)$
$\lambda P[P^e(j)]$	$\langle\langle s,\langle e,t\rangle\rangle,t\rangle(\ cancellation\ rule\ for \lambda expressions)$

TABLE 3

as for example: lexical entries *man* and *walk* are translated in *man'* and *walk'*. The translation of proper nouns is defined as follows: *John* has as translation $John' = \lambda P[P^e(j)]$ where P is a predicate variable and j is a constant which represents *John*. Let us verify that $\lambda P[P^e(j)]$ is of type specific for terms, that means $\langle\langle s,\langle e,t\rangle\rangle,t\rangle$. See Table 3.

For translation formalism is enough to define for each syntactic functional application rule Sj (j=1,...14) of the form:

Sj: If $\alpha \in P_{A/B}$ and if $\beta \in P_B$ then $F_j(\alpha,\beta) \in P_A$

a translation rule Tj of the form:

Tj: If $\alpha \in P_{A/B}$ and $\beta \in P_B$, and if α and β translate into α' and β' respectively, then $F_j(\alpha,\beta)$ translates into $\alpha'(\beta'^i)$.

Example:

Let us compute translation of *Mary talks*. In grammar PTQ we have:

$Mary \in B_T$ so, by S1, $Mary \in P_T$.

$talk \in B_{IV}$ and, by S1, $talk \in P_{IV}$.

By S4, $Mary\,talks \in P_t$.

The translation of $Mary$ is $\lambda P[P^e(m)]$, and the translation of $talk$ este $talk'$. Henceforth, the translation of $F_4(Mary, talk)$ is $\alpha'(\beta'^i) = \lambda P[P^e(m)](talk'^i)$. This last formula may be simplified to

$\lambda P[P^e(m)](talk'^i) = talk'^{e,i}(m) = talk'(m)$.

The first simplification is a $\lambda conversion$, the second is by the rule ^{i,e}.

Example

We can verify that the determiner *every* (an expression of the category T/CN) has the appropriate type:

$f(T/CN) = f((t/IV)/CN) = \langle\langle s, f(CN)\rangle, f(t/IV)\rangle = \langle\langle s, \langle e, t\rangle\rangle, \langle\langle s, f(IV)\rangle, f(t)\rangle\rangle = \langle\langle s, \rangle e, t\rangle\rangle, \langle\langle s, \langle e, t\rangle\rangle, t\rangle\rangle$.

Indeed, for *every*, which is translated in $\lambda Z[\lambda Y[\forall x(Z^e(x) \to Y^e(x))]]$ we have: expression $\forall x(Z^e(x) \to Y^e(x))$ is of type t. As Y is of type $\langle s, \langle e, t\rangle\rangle$, expression $\lambda Y[\forall x(Z^e(x) \to Y^e(x))$ is of type $\langle\langle s, \langle e, t\rangle\rangle, t\rangle$. Z is also of type $\langle s, \langle e, t\rangle\rangle$, and then $\lambda Z[\lambda Y[\forall x(Z^e(x) \to Y^e(x))]]$ is of $\langle\langle s, \langle e, t\rangle\rangle, \langle\langle s, \langle e, t\rangle\rangle, t\rangle\rangle$.

4. The treatment of quantitative sentences

In the following we will try to explain the utility of intensional logic for the semantic representation of some significant quantitative sentences. This kind of natural language sentences presents a special importance because of it's mostly use as query language over the internet and most of these sentences refer to quantity (products, money). On the other hand these types of sentences can also be used for the acquisition of new knowledge in a knowledge base which has as input natural language sentences.

For an easier exemplification we split the quantitative sentences in three categories:

• Definite quantity sentences (those sentences of which quantifiers represent exactly the expressed quantity). Example: *Four men cry. John eats an apple.*

• Indefinite quantity sentences (the quantifiers of these sentences gives us an approximation of the expressed quantity without specify it). Example: *Most women cry. A number of people read.*

• Restrictive quantity sentences (in this case the quantifiers restrict with precision the expressed quantity). Example: *Maximum five children answer.*

Generally speaking, the quantitative sentences are generated by the presence of numerals, but there are also cases when there aren't any numeral in these sentences (for example, the indefinite quantity sentences). This is why we will first try to present the way that these numerals are translated in the intensional logic.

There are two types of numerals with more importance in the construction of quantitative sentences, numerals which will be translated in different way:

a) Singles numerals which are determinants of the sentences where they belong (e.g. *Five man laugh*).

b) Numerals which are preceded by an adverb with the role of pre-determinant, with which it forms the sentence determinant. (e.g. *Maximum eleven people left the room*).

In the first case the numerals represent the sentence determinant, so their type is T/CN and their translation in the intensional logic will be:

$\lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge N = j))]]$ - where j represents the number indicated by the numeral

In the construction of this expression we used the notation given in [3]. In the following we will prove that this expression is of type T/CN:

– Z^e is of type $\langle e, t\rangle$ and X is of type e $\Rightarrow$ (with the use of cancellation rule) $Z^e(X)$ is of type t;

– in the same way we can prove that the expression $Y^e(X)$ is of type t;

– N and j are of the same type e $\Rightarrow$ (using the definition of the intensional logic) the expression N=j is of type t;

– again using the definition of intensional logic which says that: if two expressions A and B have the same type t, then the expression A$\wedge$B is also of type t. Using this definition on the above expression $\Rightarrow$ the expression $Z^e(X) \wedge Y^e(X) \wedge N = j$ is of type t;

– if the expression $Z^e(X) \wedge Y^e(X) \wedge N = j$ is of type t $\Rightarrow$ the expression $\exists_N X(Z^e(X) \wedge Y^e(X) \wedge N = j)$ is also of type t;

– the expression Y is of type $\langle s, \langle e, t\rangle\rangle \Rightarrow \lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge N = j)]$ has the type $\langle\langle s, \langle e, t\rangle\rangle, t\rangle$;

– the expression Z is of type $\langle s, \langle e, t\rangle\rangle \Rightarrow \lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge N = j)]]$ has the type $\langle\langle s, \langle e, t\rangle\rangle, \langle\langle s, \langle e, t\rangle\rangle, t\rangle$, type which is T/CN □

In the second situation the numerals will be translated such as base expression by their semantics (value for the numeral). For example, the numeral *four* has the translation 4'.

In the following we will explain (based on an example) the way that definite quantity sentences will be translated in the intensional logic form (in this example, we will also show the way that differed type of ambiguities are solved and how we represent the **a** and the **the** determinants). Let us consider the following natural language sentence that we have to translate in its corresponding intensional logic formula:

Five men enter a room.

First of all we observe that the sentence is composed by the following atoms:
five – numeral with determinant role
men – common noun;
enter – transitive verb;
a – determinant;
room – common noun;

FIGURE 1. The derivation tree for the sentence: *Five men enter a room*

First we prove that this sentence is semantically correct (which is equivalent to prove that the sentence type is t). Next we will show that we can map this sentence in an intensional logic formula.

To prove that the type of this sentence is t, we used a bottom-up algorithm which constructs the sentence from its atomic parts and then, using the cancellation rule, groups these atoms (see Figure 1).

It can be notice that this is not the only derivation three which we can construct (at the second step of the construction we could apply the cancellation rule to couple enter with *five men* instead with the sequence *a man* - this is named the de re interpretation). This two way interpretation of the sentence is caused by its ambiguity introduced by the scope of the quantifiers.

Now we try to translate this sentence. As we saw earlier, there are two interpretations for this sentence: de dicto and de re. Each type of interpretation will gives us another formula of the intensional logic (this is possible because of the semantic ambiguity introduced by this sentence).

To translate this sentence we have to add a new cancellation rule for the composition of the sequence α of any type and the sequence β of type T. Thus, the

cancellation rule for the sequence $\alpha\beta$ is (F_T) $F_T(\alpha,\beta) = \beta'(\alpha'^i)$ where α', β' represents the translation of the α, β respectively.

First, we will analyze the de dicto interpretation for the sentence translation. For the sentence translation we have to follow the steps below:

• *five*$\in P_{T/CN}$ and this expression translation is (with the respect of what we had shown at the beginning of this chapter) $\lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge N = 5)]]$;

• *men*$\in P_{CN}$ and its translation is *men*'. Thus, the expression *five men* will be translated (after the use of the λ-conversion and the ^{i,e}-rule) as:

$$\lambda Y[\exists_N X(men'(X) \wedge Y^e(X) \wedge N = 5)]] \text{ (which has the type } T);$$

• *a*$\in$ $P_{T/CN}$ and the translation of this expression is $\lambda A[\lambda B[\exists_1 J(A^e(J) \wedge B^e(J))]]$ (to be more specific, we use another set of variables - the notation is taken from [4]); *room* $\in P_{CN}$ and its translation is *room*'. Thus, after applying the cancellation rule and after the use of the λ-conversion and the ^{i,e}-rule, the expression *a room* will be $\lambda B[\in_1 J(room'(J) \wedge B^e(J))]$ whose type is T.

• *enter*$\in P_{IV/T}$ with the translation $enter'^2$ (where the numeral 2 means that this predicate needs two arguments - this can be also expressed with the help of lambda calculus:$\lambda A[\lambda B(enter(A,B))]$). After the use of the cancellation rule to the expressions *enter* (of type IV/T) and *a room* (of type T) we receive (we use the F_T cancellation rule because the type of the expression *a room* is T) - after applying the λ-conversion and the ^{i,e}-rule - the expression $\exists_1 J(room'(J) \wedge enter'^1(J))$ whose type is IV (the predicate $enter'^1$ means that the predicate *enter*' needs another argument)

• Now we only have to couple the remaining two expression: *five men* (whose type is T) and *enter a room* (whose type is IV), which has the following translations $\lambda Y[\exists_N X(men'(X) \wedge Y^e(X) \wedge N = 5)]]$ and $\exists_1 J(room'(J) \wedge enter'^1(J))$ respectively. After applying the λ-conversion and after applying the ^{i,e}-rule we will obtain the following expression of the intensional logic:

$$\exists_N X(men'(X) \wedge \exists_1 J(room'(J) \wedge enter'^1(J))(X) \wedge N = 5) \Longleftrightarrow$$

$$\exists_N X(men'(X) \wedge (\exists_1 J(room'(J) \wedge enter'(J,X))) \wedge N = 5)$$

The semantic of this formula express the fact that there are exactly *five men* who enter in a room (not necessarily the same room).

To obtain the other translation of the sentence we must follow the de re interpretation (we will give only the important steps of the process - the steps which was excluded are the same with the previous interpretation):

• *five* $\in P_{T/CN}$ and this expression translation is (with the respect of what we had shown at the beginning of this chapter) $\lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge N = 5)]]$;

• *men* $\in P_{CN}$ and its translation is *men*'. Thus, the expression *five men* will be translated (after applying the λ-conversion and the ^{i,e}-rule) as:

$$\lambda Y[\lambda_N X(men'(X) \wedge Y^e(X) \wedge N = 5)]]$$

(which has the type T);

• *enter* $\in P_{IV/T}$ with the translation $enter'^2$. This, the expression *Five men enter* will be mapped in:

$$\exists_N X(men'(X) \wedge enter'^1(X) \wedge N = 5)]$$

(whose type is t/T); (I)

• $a \in P_{T/CN}$ and the translation of this expression is $\lambda A[\lambda B[\exists_1 J(A^e(J) \wedge B^e(J))]]$ (to be more specific, we use another set of variables - the notation is taken from [4]); *room* $\in P_{CN}$ and its translation is *room*'. Thus, after applying the cancellation rule and after the use of the λ-conversion and the i, e-rule, the expression a room will be $\lambda B[\exists_1 J(room'(J) \wedge B^e(J))]$whose type is T; (II)

• after the combining the two expressions (I) and (II) and after using the cancellation rule F_T we will obtain the following formula:

$$\exists_1 J(room'(J) \wedge (\exists_N X(men'(X) \wedge enter'^1(X) \wedge N = 5))(J)) \Longleftrightarrow$$

$$\exists_1 J(room'(J) \wedge (\exists_N X(men'(X) \wedge enter'(X, J) \wedge N = 5)))$$

The semantic of this formula express the fact that there is exactly one room in which five men enter. In the same way we can translate the sentence *Five men enter the room* (formula which is obtained by replacing the "a" determinant with the "the" determinant) which has the following to translation (corresponding to the de dicto and de re interpretation):

$$\exists_N X(men'(X) \wedge (\exists_1 J(\forall K(room'(J) \equiv J = K) \wedge enter'(J, X))) \wedge N = 5)$$

(de re) and

$$\exists_1 J((\forall K(room'(J) \equiv J = K) \wedge (\exists_N X(men'(X) \wedge enter'(X, J) \wedge N = 5)))$$

(de dicto)

As it may be seen these two expressions are equivalent semantically speaking.

Now we show how we can map another type of quantitative sentences: the restrictive quantity sentences. As we had done in the previous case we will start to analyze an example sentence. Thus, let consider the sentence: *Minimum ten men laugh*, where:

minimum – adverb (pre-determinant) with the type $T/CN/T/CN$;

ten – numeral;

men – noun;

laugh – intransitive verb.

It can be prove that this sentence is correct (from the syntactic point of view) in a similar way as it was proved for the exact quantity sentence.

For the formula translation we have to follow the following steps (as we had done earlier with the other example):

• *minimum* $\in P_{T/CN/T/CN}$ and its translation:

$$\lambda K[\lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge minimum'(N, K^e))]]]$$

Which has the type T/CN/T/CN (to prove the type we must do the steps shown for the numeral)

• *ten* $\in P_{T/CN}$ and its translation will be *10*'. After applying the cancellation rule for the above two expression (after applying the λ-conversion and the ^{i,e}-rule) we obtain:

$$\lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge minimum'(N, 10'))]];$$

• *men* $\in P_{CN}$ which translation is *men*'. Thus, the translation of the expression *minimum ten men* (after applying the λ-conversion and the ^{i,e}-rule) is:

$$\lambda Y[\exists N X(men'(X) \wedge Y^e(X) \wedge minimum'(N, 10'))];$$

• *laugh* $\in P_{IV}$ with the translation *laugh*'. Thus the final translation of the sentence minimum ten men laugh (after applying the λ-conversion and the ^{i,e}-rule) will be:

$$\exists_N X(men'(X) \wedge laugh'(X) \wedge minimum'(N, 10'))$$

- with the type t;

The semantic of this formula means that there are minimum ten men who laugh. It must be remarked that in this case we had also a λ-variable (K) which took place as a predicate argument (predicate which is also expressed by an λ-variable).

The last type of sentences which will be discussed here is the indefinite quantity sentences. This type of sentences does not need the presence of the numeral. For this kind of sentences we choose the following sentence: *Most people run* where the syntactic types of its atoms are:

most – determinant;

people – noun

run – intransitive verb

Similarly to the previous two formulas we can prove that this formula type is t (this, the sentence is correct in the syntactic point of view).

As we have already seen in the other cases, we will follow the next steps:

• *most* $\in P_{T/CN}$ with the translation:

$$\lambda Z[\lambda Y[\exists_N X(Z^e(X) \wedge Y^e(X) \wedge most'(N))]]$$

- of the type T/CN

• *people* $\in P_{CN}$ its translation is *people*'. After applying the cancellation rule with the expression most (after applying the λ-conversion and the ^{i,e}-rule) we obtain:

$$\lambda Y[\exists_N X(people(X) \wedge Y^e(X) \wedge most'(N))]$$

- of type T

• $run \in P_{IV}$ with the translation *run*'. After applying the cancellation rule with the above expression (after applying the λ-conversion and the ^{i,e}-rule) we obtain the final formula:$\exists_N X(people(X) \wedge run'(X) \wedge most'(N))$ - which expresses the fact that there are N people who run and N follows the **most**' predicate.

By the previous three examples we wanted to show the importance of the categorical grammars and of the intensional logic in the natural language representation in general and of the quantitative sentences in particular. As we have seen, this translation technique of the natural language semantic can be used successfully in real life. It can also be applied to express more complex sentences that can hardly solved with the use of other grammars.

References

[1] J.Allen : "Natural language understanding", Benjamin/Cummings Publ., 2nd ed., 1995.
[2] D. Jurafsky, J. Martin: "Speech and language processing", Prentice Hall, 2000.
[3] G. Morrill:"Type Logical Grammar.Categorial Logic of Signs", Kluwer Academic Publishers, 1994.
[4] A. Onet, D. Tatar: "Semantic representation of the quantitative natural languange sentences", Studia Univ. "Babes-Bolyai", Seria Informatica, 1999, nr 2, pg 99-109.
[5] A. Onet, D. Tatar: "Semantic analysis in dialogue interfaces", Studia Univ. "Babes-Bolyai", Seria Informatica, 2000, nr 1, pp79-89.
[6] D. Tatar: "Inteligenta artificiala: demonstrare automata de teoreme, prelucrarea limbajului natural", Editura Microinformatica, 2001.
[7] A. Thayse (editor): "From modal logic to deductive databases", John Wiley and Sons, 1989.
[8] A. Thayse (editor): "From natural language processing to logic for expert systems", John Wiley and Sons, 1990.

Department of Computer Science, Faculty of Mathematics and Computer Science, "Babeş-Bolyai" University, Cluj-Napoca, Romania
E-mail address: adrian|dtatar@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

A PRACTICAL COALITION-RESISTANT GROUP BLIND SIGNATURE SCHEME

CONSTANTIN POPESCU

Abstract. A group signature allow any member of a group to sign on behalf of the group. A group blind signature requires that a group member signs on group's behalf a document without knowing its content. In this paper we propose a practical coalition-resistant group blind signature scheme based on the strong RSA and the decisional Diffie-Hellman assumptions. Our scheme is an extension of the group signature scheme proposed in [3] that adds the blindness property.

Keywords: Group blind signature scheme, group signatures, blind signatures, strong RSA assumption

1. Introduction

Group signature schemes are a relatively recent cryptographic concept introduced by Chaum and van Heyst [12] in 1991. Group signatures are publicly verifiable but anonymous in that, no one, with the exception of a designated group manager, can establish the identity of a signer. Furthermore, group signatures are unlinkable which makes computationally hard to establish whether or not multiple signatures are produced by the same group member. At the same time, no one, including the group manager, can misattribute a valid group signature. A group signature scheme could for instance be used in many specialized applications, such as voting and bidding. They can, for example, be used in invitations to submit tenders. All companies submitting a tender form a group and each company signs its tender anonymously using the group signature. Once the preferred tender is selected, the winner can be traced while the other bidders remain anonymous. More generally, group signatures can be used to conceal organizational structures, e.g., when a company or a government agency issues a signed statement. Also, a group signature scheme could be used by an employee of a large company to sign documents on behalf of the company. A further application of a group signature schemes is electronic cash as was pointed out in [18]. In this case, several banks issue coins, but it is impossible for shops to find out which bank issued a coin

2000 *Mathematics Subject Classification.* 68P25.
1998 *CR Categories and Descriptors.* E.3 [**Data**]: Data Encryption.

that is obtained from a customer. The central bank plays the role of the group manager and all other banks issuing coins are group members.

Group signatures were first introduced by Chaum and van Heijst [12]. A number of improvements and enhancements followed [1, 17, 21, 25, 26]. However, in the schemes presented in [8, 12, 19, 20, 21, 22] the length of signatures and the size of the group's public key depend on the size of the group and thus these schemes are not suitable for large groups. The first group signature suitable for large groups is that of Camenisch and Stadler [7], where both the length of the group public key and the group signatures are independent of the group's size. The Camenisch-Stadler scheme was improved by Camenisch and Michels in [5], which undoubtedly represents the state of the art in the field.

In this paper we propose a group blind signature scheme which combines the notions of group signatures and blind signatures [6, 10, 11, 16]. Our scheme is an extension of the group signature scheme from reference [3] that adds the blindness property and is more efficient and secure than [23] and the Lysyanskaya-Ramzan scheme [18]. In particular, our scheme's registration protocol (Join) for new members is an order of magnitude more efficient. Our scheme is based on the strong RSA and the decisional Diffie-Hellman assumptions and is as secure and efficient as the basic group signature scheme proposed in [3].

2. The Group Blind Signature Scheme

Our group blind signature scheme is an extension of the group signature scheme from reference [3] that adds the blindness property. Participants are group members, a group manager and several users. Our group blind signature scheme allows the members of a group to sign messages on behalf of the group such that the following properties hold:

(1) Blindness of signatures: The signer (a group member) signs on group's behalf a message without knowing its content. Moreover, the signer should have no recollection of having signed a particular document even though he can verify that he did indeed sign it.
(2) Unforgeability: Only group members are able to sign messages on behalf of the group.
(3) Anonimity: Given a signature, identifying the actual signer is computationally hard for everyone but the group manager.
(4) Unlinkability: Deciding whether two different signatures were computed by the same group member is computationally hard.
(5) Traceability: The group manager can always establish the identity of the member who issued a valid signature.
(6) No framing: Even if the group manager and some of the group members collude, they cannot sign on behalf of non-involved group members.

(7) Coalition-resistance: A colluding subset of group members cannot generate a valid signature that the group manager cannot link to one of the colluding group members.

Definition 1. *A group blind signature scheme is a digital signature scheme comprised of the following algorithms:*

(1) ***Setup****: The public output is the group's public key P. The private outputs are the individual secret keys x_G for each group member, the secret key x_M for the group manager.*
(2) ***Join****: An interactive protocol between the group manager and a user that results in the user becoming a new group member.*
(3) ***Sign****: An interactive protocol between the group member A and an external user, which on input message m from the user, the A's secret key x_G and the group's public key P outputs a blind signature σ.*
(4) ***Verify****: An algorithm that for an input composed of a message m, a signature σ and the group's public key P returns 1 if and only if σ was generated by any group member using the protocol* ***Sign*** *on input x_G, m and P.*
(5) ***Tracing****: A tracing algorithm that for an input composed of a signature σ, a message m, the group manager's secret key x_M and the group's public key P returns the identity ID of the group member who issued the signature σ together with an argument arg of this fact.*
(6) ***Vertracing****: A tracing verification algorithm that for an input composed of a signature σ, a message m, the group's public key P, the identity ID of a group member and an argument arg outputs 1 if and only if arg was generated by tracing with respect to m, σ, P and x_M.*

In this section we review some cryptographic assumptions necessary in the subsequent design of our group blind signature scheme. The Strong RSA Assumption was independently introduced by Baric and Pfitzman [4] and by Fujisaki and Okamoto [14].

Definition 2 (Strong RSA Problem)**.** *Let $n = pq$ be an RSA-like modulus and let G be a cyclic subgroup of $\mathbb{Z}_n^*$ of order l_g. Given n and $z \in G$, the Strong RSA Problem consists of finding $u \in G$ and $e \in \mathbb{Z}_{>1}$ satisfying $z \equiv u^e (mod\ n)$.*

Assumption 1 (Strong RSA Assumption)**.** *There exists a probabilistic polynomial time algorithm K which on input 1^{l_g} outputs a pair (n, z) such that for all probabilistic polynomial-time algorithms P the probability that P can solve the Strong RSA Problem is negligible.*

Assumption 2 (Decisional Diffie-Hellman Assumption)**.** *Let $n = pq$ be an RSA-like modulus and let α be a quadratic residue modulo n that has a large order in $\mathbb{Z}_n^*$. Let $G =< \alpha >$. Given as input a triplet $(\alpha^a, \alpha^b, \alpha^c)$ in G^3, it is hard to decide whether $(\alpha^a, \alpha^b, \alpha^c)$ is a Diffie-Hellman triplet $(\alpha^a, \alpha^b, \alpha^{ab})$ or a random triplet.*

For the Decisional Diffie-Hellman Assumption see [5] for more details.

The security of our group blind signature scheme is based on these assumptions.

3. Signatures of Knowledge

In this section we present some well studied techniques for proving knowledge of discrete logarithms. A signature of knowledge is a construct that uniquely corresponds to a given message m that cannot be obtained without the help of a party that knows a secret such that as the discrete logarithm of a given $y \in G$ to the base g ($G =< g >$). Let $k, l_1, l_2 < l_g$ and $\varepsilon > 1$ be security parameters.

We use the following notations:

- The symbol $\|$ denotes the concatenation of two binary string (or of the binary representation of group elements and integers).
- We assume a collision-resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^k$ which maps a binary string of arbitrary length to a k-bit hash value.
- The notation $H(m \parallel g \parallel y \parallel g^s y^c)$ denotes the message digest of the block of data $m \parallel g \parallel y \parallel g^s y^c$.
- The notation $r \in_R \{0,1\}^{\varepsilon(l_g+k)}$ denotes that r is random in $\{0,1\}^{\varepsilon(l_g+k)}$.
- We denote $\log_g y = \alpha$.
- $SPK\{(\alpha) : y = g^\alpha\}(m)$ is a signature of a message $m \in \{0,1\}^*$ with respect to y.
- $SPK\{(\alpha) : y_1 = g^\alpha \wedge y_2 = h^\alpha\}(m)$ is a signature of a message $m \in \{0,1\}^*$ with respect to y_1 and y_2.
- $SPK\left\{(\alpha) : h = g^\alpha \wedge \delta = \beta^\alpha \wedge (2^{l_1} - 2^{\varepsilon(l_2+k)+1} < \alpha < 2^{l_1} + 2^{\varepsilon(l_2+k)+1})\right\}$ (m) is a proof of knowledge of the discrete logarithm of h with respect to base g and of δ with respect to β, $log_g h = log_\beta \delta$ and $log_g h$ is in the interval $\left\{2^{l_1} - 2^{\varepsilon(l_2+k)+1}, ..., 2^{l_1} + 2^{\varepsilon(l_2+k)+1}\right\}$.

A proof of knowledge is a way for one person to convince another person that he knows some fact without actually revealing that fact. A signature of knowledge is used both for the purpose of signing a message and proving knowledge of a secret. Signatures of knowledge were used by Camenisch and Michels [5] and their construction is based on the Schnorr signature scheme [24] to prove knowledge.

Showing the knowledge of a discrete logarithm [5] can be done easily as stated by the following definition.

Definition 3. *Let $\varepsilon > 1$ be a security parameter. A pair $(c, s) \in \{0,1\}^k \times \left\{-2^{l_g+k}, \ldots, 2^{\varepsilon(l_g+k)}\right\}$ satisfying $c = H(m \parallel g \parallel y \parallel g^s y^c)$ is a signature of a message $m \in \{0,1\}^*$ with respect to y and is denoted by $SPK\{(\alpha) : y = g^\alpha\}(m)$.*

A signature $(c, s) = SPK\{(\alpha) : y = g^\alpha\}(m)$ of a message $m \in \{0,1\}^*$ can be computed as follows. An entity knowing the secret key $\alpha \in \{0,1\}^{l_g}$ such that

$y = g^{\alpha}$, chooses $r \in_R \{0,1\}^{\varepsilon(l_g+k)}$ and computes $t = g^r,\ c = H(m \parallel g \parallel y \parallel t),\ s = r - c\alpha$.

A slight modification of the previous definition enables to show the knowledge and equality of two discrete logarithms described in [5].

Definition 4. *A pair* $(c,s) \in \{0,1\}^k \times \left\{-2^{l_g+k}, ..., 2^{\varepsilon(l_g+k)}\right\}$ *satisfying* $c = H(m \parallel g \parallel h \parallel y_1 \parallel y_2 \parallel y_1^c g^s \parallel y_2^c h^s)$ *is a signature of a message* $m \in \{0,1\}^*$ *with respect to* y_1 *and* y_2 *and is denoted by* $SPK\{(\alpha) : y_1 = g^{\alpha} \wedge y_2 = h^{\alpha}\}(m)$.

A signature $(c,s) = SPK\{(\alpha) : y_1 = g^{\alpha} \wedge y_2 = h^{\alpha}\}(m)$ of a message $m \in \{0,1\}^*$ can be computed as follows. An entity knowing the secret key $\alpha \in \{0,1\}^{l_g}$ such that $y_1 = g^{\alpha}$ and $y_2 = h^{\alpha}$, chooses $r \in_R \{0,1\}^{\varepsilon(l_g+k)}$ and computes $t_1 = g^r,\ t_2 = h^r,\ c = H(m \parallel g \parallel h \parallel y_1 \parallel y_2 \parallel t_1 \parallel t_2),\ s = r - c\alpha$.

Definition 5. *A tuple* $(c_1, c_2, s_1, s_2) \in \{0,1\}^k \times \{0,1\}^k \times \left\{-2^{l_g+k}, ..., 2^{\varepsilon(l_g+k)}\right\} \times \left\{-2^{l_g+k}, ..., 2^{\varepsilon(l_g+k)}\right\}$ *satisfying* $c_1 \oplus c_2 = H(m \| g \| h \| y_1 \| y_2 \| y_1^{c_1} g^{s_1} \| y_2^{c_2} h^{s_2})$ *is a signature of a message* $m \in \{0,1\}^*$ *with respect to* y_1 *and* y_2 *and is denoted by* $SPK\left\{(\alpha,\beta) : y_1 = g^{\alpha} \vee y_2 = h^{\beta}\right\}(m)$.

This definition shows the knowledge of one out of two discrete logarithms [5]. If the signer knows the secret key $\alpha \in \{0,1\}^{l_g}$ such that $y_1 = g^{\alpha}$, then he can compute this signature as follows. The signer chooses $r_1 \in_R \{0,1\}^{\varepsilon(l_g+k)},\ r_2 \in_R \{0,1\}^{\varepsilon(l_g+k)},\ c_2 \in_R \{0,1\}^k$ and computes $t_1 = g^{r_1},\ t_2 = h^{r_2} y_2^{c_2},\ c_1 = c_2 \oplus H(m \parallel g \parallel h \parallel y_1 \parallel y_2 \parallel t_1 \parallel t_2),\ s_1 = r_1 - c_1\alpha,\ s_2 = r_2$.

The next block is based on a proof that the secret the prover knows lies in a given interval. This building block is related to the new Range Bounded Commitment protocol (RBC) of Chan et al. [9]. It is also related to a protocol given by Camenisch and Michels [5].

Definition 6. *A proof of knowledge of the discrete logarithm of* h *with respect to base* g *and of* δ *with respect to* β*, which also proves that* $log_g h = log_\beta \delta$ *and that* $log_g h$ *is in the interval* $\left\{2^{l_1} - 2^{\varepsilon(l_2+k)+1}, ..., 2^{l_1} + 2^{\varepsilon(l_2+k)+1}\right\}$ *is a pair* (c,s)*, and is denoted by*
$SPK\left\{(\alpha) : h = g^{\alpha} \wedge \delta = \beta^{\alpha} \wedge (2^{l_1} - 2^{\varepsilon(l_2+k)+1}) < \alpha < (2^{l_1} + 2^{\varepsilon(l_2+k)+1})\right\}(m)$,
where $c = H(m \| g \| h \| \beta \| \delta \| g^{s-c2^{l_1}} h^c \| \beta^{s-c2^{l_1}} \delta^c)$ *and* s *is in the interval* $\left\{-(2^k-1)(2^{l_2}-1), ..., 2^{\varepsilon(l_2+k)}\right\}$.

This signature can be computed as follows. If the signer knows an integer $\alpha \in \left\{2^{l_1}, ..., 2^{l_1} + 2^{l_2} - 1\right\}$ such that $h = g^{\alpha}$ and $\delta = \beta^{\alpha}$, he chooses a random $t \in \{0,1\}^{\varepsilon(l_2+k)}$ and computes $c = H(m \| g \| h \| \beta \| \delta \| g^t \| \beta^t),\ s = t - c(\alpha - 2^{l_1})$.

The security of all the presented building blocks has been proven in the random oracle model [13] under the strong RSA assumption in [5, 14, 15].

4. The Proposed Group Blind Signature Scheme

We propose a realization of a group blind signature scheme the security of which is based on the Strong RSA Assumption and Decisional Diffie-Hellman Assumption. Our scheme is as secure and efficient as the basic group signature scheme proposed in [3].

Let G be a cyclic subgroup of $\mathbb{Z}_n^*$ of order l_g. Let $k, l_1, l_2 < l_g$ and $\varepsilon > 1$ be security parameters. Finally, let H be a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^k$.

4.1. **Setup.** The setup procedure of our scheme (as in [3]) is as follow. The group manager executes the following steps:

(1) Select random secret l_g-bit primes p', q' and computes $p = 2p' + 1$ and $q = 2q' + 1$. Set the modulus $n = pq$. It is a good habit to restrict the operation to the subgroup of quadratic residues modulo n, i.e., the cyclic subgroup $QR(n)$ generated by an element of order $p'q'$. This is because the order $p'q'$ of $QR(n)$ has no small factors.
(2) Choose random elements $a, a_0, g, h \in QR(n)$ of order $p'q'$.
(3) Choose a random secret element $x \in \mathbb{Z}_{p'q'}^*$ and set $y = g^x \bmod n$.
(4) The group public key is $P = (n, a, a_0, y, g, h)$.
(5) The corresponding secret key is $S = (p', q', x)$.

4.2. **Join.** Suppose now that a user wants to join the group. We assume that communication between the group member and the group manager is secure, i.e., private and authentic. To obtain his membership certificate, each user U_i must perform the following protocol with the group manager.

(1) The user U_i generates a secret exponent $x_i' \in \left[0, 2^{l_2}\right]$, a random integer $r \in \left[0, 2^{n^2}\right]$ and sends $C_1 = g^{x_i'} h^r \bmod n$ to group manager and proves him knowledge of the representation of C_1 with respect to bases g and h.
(2) The group manager checks that $C_1 \in QR(n)$. If this is the case, the group manager selects $\alpha_i, \beta_i \in \left[0, 2^{l_2}\right]$ at random and sends (α_i, β_i) to U_i.
(3) The user U_i computes $x_i = 2^{l_1} + \left(\alpha_i x_i' + \beta_i \bmod 2^{l_2}\right)$ and sends to group manager the value $C_2 = a^{x_i} \bmod n$. The user also proves to group manager:
   (a) that the discrete logarithm of C_2 with respect to base a lies in the interval $\left[2^{l_1} - 2^{l_2}, 2^{l_1} + 2^{l_2}\right]$.
   (b) knowledge of integers u, v, w such that: u lies in the interval $\left[-2^{l_2}, 2^{l_2}\right]$, u equals the discrete logarithm of $C_2/a^{2^{l_1}}$ with respect to base a and $C_1^{\alpha_i} g^{\beta_i}$ equals $g^u \left(g^{2^{l_2}}\right)^v h^w$.

(4) The group manager checks that $C_2 \in QR(n)$. If this is the case and all the above proofs were correct, group manager selects a random prime $e_i \in \left[2^{l_1} - 2^{l_2}, 2^{l_1} + 2^{l_2}\right]$ and conmputes $A_i = (C_2 a_0)^{1/e_i} \pmod n$. Finally, group manager sends to U_i the new membership certificate (A_i, e_i).
(5) The user U_i verifies that $a^{x_i} a_0 \equiv A_i^{e_i} \pmod n$.
(6) The group manager creates a new entry in the membership table and stores (A_i, e_i) in the new entry.

4.3. **Sign.** In this subsection we present our signature protocol which is blind, unlike [3]. First, we define a group blind signature and then we show how a group member can generate such a group blind signature.

Definition 7. *Let ε, l_1, l_2 be security parameters such that $\varepsilon > 1, l_2 < l_1 < l_g$ and $l_2 < \frac{l_g - 2}{\varepsilon} - k$ holds. A group blind signature of a message $m \in \{0,1\}^*$ is $(c, s_1, s_2, s_3, s_4, A, B, D) \in \{0,1\}^k \times \left\{-2^{\varepsilon(l_2+k)+1}, ..., 2^{\varepsilon(l_2+k)+1}\right\} \times \left\{-2^{\varepsilon(l_2+k)+1}, ..., 2^{\varepsilon(l_2+k)+1}\right\} \times \left\{-2^{\varepsilon(l_1+2l_g+k+1)+1}, ..., 2^{\varepsilon(l_1+2l_g+k+1)+1}\right\} \times \left\{-2^{\varepsilon(2l_g+k)+1}, ..., 2^{\varepsilon(2l_g+k)+1}\right\} \times G^3$ satisfying $c = H(m \parallel g \parallel h \parallel y \parallel A \parallel B \parallel D \parallel a_0^c A^{s_1 - c2^{l_1}} / \left(a^{s_2 - c2^{l_1}} y^{s_3}\right) \parallel B^{s_1 - c2^{l_1}} / g^{s_3} \parallel B^c g^{s_4} \parallel D^c g^{s_1 - c2^{l_1}} h^{s_4})$.*

The protocol for obtaining a group blind signature is as follows. When responding to a sign request, the signer (the group member U_i) does the following:

(1) Chooses an integer $w \in_R \{0,1\}^{2l_g}$ and computes

$$A = A_i y^w \pmod n, \; B = g^w \pmod n, \; D = g^{e_i} h^w \pmod n.$$

(2) Chooses $\tilde{r_1} \in_R \{0,1\}^{\varepsilon(l_2+k)}$, $\tilde{r_2} \in_R \{0,1\}^{\varepsilon(l_g+l_1+k)}$, $\tilde{r_3} \in_R \{0,1\}^{\varepsilon(l_g+k)}$, $\tilde{r_4} \in_R \{0,1\}^{\varepsilon(l_2+k)}$ and computes

$$\begin{aligned} \tilde{t_1} &= A^{\tilde{r_1}} / \left(a^{\tilde{r_2}} y^{\tilde{r_3}}\right) \\ \tilde{t_2} &= B^{\tilde{r_1}} / g^{\tilde{r_3}} \\ \tilde{t_3} &= g^{\tilde{r_4}} \\ \tilde{t_4} &= g^{\tilde{r_1}} h^{\tilde{r_4}}. \end{aligned}$$

(3) Sends $(A, B, D, \tilde{t_1}, \tilde{t_2}, \tilde{t_3}, \tilde{t_4})$ to the user.

In turn, the user does the following:

(1) Chooses $\gamma_1, \gamma_2, \gamma_3, \gamma_4, \delta \in_R \{0,1\}^{\varepsilon(l_g+k)}$ and computes

$$\begin{aligned} t_1 &= a_0^\delta \tilde{t_1} A^{\gamma_1 - \delta 2^{l_1}} / \left(a^{\gamma_2 - \delta 2^{l_1}} y^{\gamma_3}\right) \\ t_2 &= \tilde{t_2} B^{\gamma_1 - \delta 2^{l_1}} / g^{\gamma_3} \\ t_3 &= \tilde{t_3} B^\delta g^{\gamma_4} \\ t_4 &= \tilde{t_4} D^\delta g^{\gamma_1} h^{\gamma_4}. \end{aligned}$$

(2) Computes

$$\begin{aligned} c &= H(m\|g\|h\|y\|A\|B\|D\|t_1\|t_2\|t_3\|t_4) \\ \tilde{c} &= c - \delta. \end{aligned}$$

(3) Sends $\tilde{c}$ to the signer.

The signer does the following:

(1) Computes

$$\begin{aligned} \tilde{s_1} &= \tilde{r_1} - \tilde{c}(e_i - 2^{l_1}) \\ \tilde{s_2} &= \tilde{r_2} - \tilde{c}(x_i - 2^{l_1}) \\ \tilde{s_3} &= \tilde{r_3} - \tilde{c}e_i w \\ \tilde{s_4} &= \tilde{r_4} - \tilde{c}w. \end{aligned}$$

(2) Sends $(\tilde{s_1}, \tilde{s_2}, \tilde{s_3}, \tilde{s_4})$ to the user.

The user does the following:

(1) Computes

$$\begin{aligned} s_1 &= \tilde{s_1} + \gamma_1 \\ s_2 &= \tilde{s_2} + \gamma_2 \\ s_3 &= \tilde{s_3} + \gamma_3 \\ s_4 &= \tilde{s_4} + \gamma_4. \end{aligned}$$

(2) The resulting signature of a message m is $(c, s_1, s_2, s_3, s_4, A, B, D)$.

The tuple $(c, s_1, s_2, s_3, s_4, A, B, D)$ is a group signature of a message $m \in \{0,1\}^*$ and the above protocol is a group blind signature scheme.

4.4. **Verifying Signatures, Tracing and Verifying Tracing.** The resulting signature $(c, s_1, s_2, s_3, s_4, A, B, D)$ of a message m can be verified as follows:

(1) Compute $c' = H(m \parallel g \parallel h \parallel y \parallel A \parallel B \parallel D \parallel a_0^c A^{s_1 - c2^{l_1}} / \left(a^{s_2 - c2^{l_1}} y^{s_3}\right) \parallel B^{s_1 - c2^{l_1}} / g^{s_3} \parallel B^c g^{s_4} \parallel D^c g^{s_1 - c2^{l_1}} h^{s_4})$.

(2) Accept the signature if and only if $c = c'$ and $s_1 \in \left\{-2^{\varepsilon(l_2+k)+1}, ..., 2^{\varepsilon(l_2+k)+1}\right\}$, $s_2 \in \left\{-2^{\varepsilon(l_2+k)+1}, ..., 2^{\varepsilon(l_2+k)+1}\right\}$, $s_3 \in \left\{-2^{\varepsilon(l_1+2l_g+k+1)+1}, ..., 2^{\varepsilon(l_1+2l_g+k+1)+1}\right\}$, $s_4 \in \left\{-2^{\varepsilon(2l_g+k)+1}, ..., 2^{\varepsilon(2l_g+k)+1}\right\}$.

Given a signature $\sigma = (c, s_1, s_2, s_3, s_4, A, B, D)$ of a message m, the group manager can find out which one of the group members issued this signature by checking its correctness. He aborts if the signature is not correct. Otherwise, he computes $u' = A/B^x$, issues a signature

$$P := SPK\{(\alpha) : y = g^\alpha \wedge A/u' = B^\alpha\}(u'\|\sigma\|m)$$

(see Definition 4) and reveals $arg := u' \| P$. He then looks up u' in the group member list and will find the corresponding u and the group member's identity. Checking whether the group manager correctly revealed the originator of a signature $\sigma = (c, s_1, s_2, s_3, s_4, A, B, D)$ of a message m can simply be done by verifying σ and arg.

5. Security and Efficiency of Our Scheme

Our group blind signature scheme is as secure and efficient as the group signature scheme proposed in [3], but more secure and efficient than group blind signature scheme from reference [23]. This, because our Join protocol is an order of magnitude more effiecient since all proofs that the new group member must provide are efficient proofs of knowledge of discrete logarithms. We show only the correctness and the blindness of the signature. The others security properties of the proposed group blind signature scheme are like in [3].

Theorem 1 (Correctness)**.** *If the user follows the blind signing protocol and accepts, then the tuple* $(c, s_1, s_2, s_3, s_4, A, B, D)$ *is a correct group signature on* $m \in \{0,1\}^*$.

Proof: The group signature $(c, s_1, s_2, s_3, s_4, A, B, D)$ is a correct group signature on m if the equality

$$\begin{aligned} c &= H(m \parallel g \parallel h \parallel y \parallel A \parallel B \parallel D \parallel a_0^c A^{s_1 - c2^{l_1}} / \left(a^{s_2 - c2^{l_1}} y^{s_3}\right) \parallel B^{s_1 - c2^{l_1}} / g^{s_3} \\ &\parallel B^c g^{s_4} \parallel D^c g^{s_1 - c2^{l_1}} h^{s_4}) \end{aligned}$$

is verified. If it can be assumed that $H(\cdot)$ is a collision-resistant, then this is equivalent to proving that $t_1 = a_0^c A^{s_1 - c2^{l_1}} / \left(a^{s_2 - c2^{l_1}} y^{s_3}\right)$, $t_2 = B^{s_1 - c2^{l_1}} / g^{s_3}$, $t_3 = B^c g^{s_4}$, $t_4 = D^c g^{s_1 - c2^{l_1}} h^{s_4}$. We have:

$$\begin{aligned} a_0^c A^{s_1 - c2^{l_1}} / \left(a^{s_2 - c2^{l_1}} y^{s_3}\right) &= a_0^{\tilde{c}+\delta} A^{\tilde{s_1} + \gamma_1 - (\tilde{c}+\delta)2^{l_1}} / \left(a^{\tilde{s_2} + \gamma_2 - (\tilde{c}+\delta)2^{l_1}} y^{\tilde{s_3} + \gamma_3}\right) = \\ \tilde{t_1} a_0^{\delta} A^{\gamma_1 - \delta 2^{l_1}} / (a^{\gamma_2 - \delta 2^{l_1}} y^{\gamma_3}) &= t_1 \\ B^{s_1 - c2^{l_1}} / g^{s_3} &= B^{\tilde{s_1} + \gamma_1 - (\tilde{c}+\delta)2^{l_1}} / g^{\tilde{s_3} + \gamma_3} = \tilde{t_2} B^{\gamma_1 - \delta 2^{l_1}} / g^{\gamma_3} = t_2 \\ B^c g^{s_4} &= B^{\tilde{c}+\delta} g^{\tilde{s_4} + \gamma_4} = \tilde{t_3} B^{\delta} g^{\gamma_4} = t_3 \\ D^c g^{s_1 - c2^{l_1}} h^{s_4} &= D^{\tilde{c}+\delta} g^{\tilde{s_1} + \gamma_1 - (\tilde{c}+\delta)2^{l_1}} h^{\tilde{s_4} + \gamma_4} = \tilde{t_4} D^{\delta} g^{\gamma_1} h^{\gamma_4} = t_4. \end{aligned}$$

This completes the proof. □

Theorem 2 (Blindness)**.** *If the user follows the protocol, then even a signer with unlimited computing power gets no information about* $m \in \{0,1\}^*$ *and the group signature* $(c, s_1, s_2, s_3, s_4, A, B, D)$.

Proof: To prove that the protocol is blind we show that for every possible signer's view there exists a unique tuple of blind factors $(\delta, \gamma_1, \gamma_2, \gamma_3, \gamma_4)$. Given any view consisting of $\tilde{r_1}, \tilde{r_2}, \tilde{r_3}, \tilde{r_4}, \tilde{t_1}, \tilde{t_2}, \tilde{t_3}, \tilde{t_4}, \tilde{c}, \tilde{s_1}, \tilde{s_2}, \tilde{s_3}, \tilde{s_4}$ and any group signature $(c, s_1, s_2, s_3, s_4, A, B, D)$ of a message m, we consider $\delta = c - \tilde{c}$, $\gamma_1 = s_1 - \tilde{s_1}$, $\gamma_2 = s_2 - \tilde{s_2}$, $\gamma_3 = s_3 - \tilde{s_3}$, $\gamma_4 = s_4 - \tilde{s_4}$. It is easy to verify that the following equations hold:

$$\tilde{t_1} a_0^{\delta} A^{\gamma_1 - \delta 2^{l_1}} / \left(a^{\gamma_2 - \delta 2^{l_1}} y^{\gamma_3}\right) = a_0^{c} A^{\tilde{s_1} + \gamma_1 - \delta 2^{l_1}} / \left(a^{\tilde{s_2} + \gamma_2 - \delta 2^{l_1}} y^{\tilde{s_3} + \gamma_3}\right) =$$

$$\begin{aligned} a_0^{c} A^{s_1 - c 2^{l_1}} / \left(a^{s_2 - c 2^{l_1}} y^{s_3}\right) &= t_1 \\ \tilde{t_2} B^{\gamma_1 - \delta 2^{l_1}} / g^{\gamma_3} = B^{\tilde{s_1} + \gamma_1 - \delta 2^{l_1}} / g^{\tilde{s_3} + \gamma_3} = B^{s_1 - c 2^{l_1}} / g^{s_3} &= t_2 \\ \tilde{t_3} B^{\delta} g^{\gamma_4} = g^{\tilde{r_4} + s_4 - \tilde{s_4}} B^{c - \tilde{c}} = B^{c} g^{s_4} &= t_3 \\ \tilde{t_4} D^{\delta} g^{\gamma_1} h^{\gamma_4} = g^{\tilde{r_1} + \gamma_1 - \delta 2^{l_1}} D^{\delta} h^{\gamma_4 + \tilde{r_4}} = D^{c} g^{s_1 - c 2^{l_1}} h^{s_4} &= t_4. \ \square \end{aligned}$$

Therefore, the above protocol is blind and our group signature is blind.

6. Conclusion

In this paper we proposed a group blind signature scheme that is secure and efficient and it is an extension of the group signature scheme from reference [3]. Our group blind signature scheme is more efficient and secure than the group blind signature scheme proposed in [23] because our scheme's registration protocol Join for new members is an order of magnitude more efficient. Also, the proposed scheme is as efficient and secure as the basic group signature scheme proposed in [3].

References

[1] ATENIESE, G. and TSUDIK, G., **Group signature a la carte**, Tenth Anual ACM-SIAM Symposium on Discrete Algorithms (SODA'99), 1999.

[2] ATENIESE, G. and TSUDIK, G., **Some open issues and new directions in group signatures**, Financial Cryptography (FC'99), Lecture Notes in Computer Science, Springer-Verlag, 1999.

[3] ATENIESE, G., CAMENISCH J., JOYE M., TSUDIK G., **A Practical and Provably Secure Coalition-Resistant Group Signature Scheme**, Advances in Cryptology - CRYPTO 2000, vol. 1880, Lecture Notes in Computer Science, Springer Verlag, pp. 255–270, 2000.

[4] BARIC, N. and PFITZMANN, B., **Collision-free accumulators and fail-stop signature schemes without trees**, In Advances in Cryptology-EUROCRYPT'97, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, pp. 480–494.

[5] CAMENISCH, J. and MICHELS, M., **A group signature scheme with improved efficiency**, Advances in Cryptology-ASIACRYPT'98, Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, 1998, pp. 160–174.

[6] CAMENISCH, J., PIVETEAU, J., and STADLER, M., **Blind signatures based on the discrete logarithm problem**, Advances in Cryptology-EUROCRYPT'94, Lecture Notes in Computer Science, vol. 950, Springer-Verlag, 1994, pp. 428–432.

[7] CAMENISCH, J. and STADLER, M., **Efficient group signature schemes for large groups**, Advances in Cryptology-CRYPTO'97, Lecture Notes in Computer Science, vol. 1296, Springer-Verlag, 1997, pp. 410–424.

[8] CAMENISCH, J., **Efficient and generalized group signatures**, Advances in Cryptology-EUROCRYPT'97, Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, pp. 465–479.

[9] CHAN, A., FRANCHEL, Y. and TSIOUNIS, Y., **Easy come-easy go divisible cash**, Updated version with corrections on the Range Bounded Commitment protocol. Available at http://www.ccs.neu.edu/home/yiannis/pubs.html.

[10] CHAUM, D., **Blind signatures for untraceable payments**, Advances in Cryptology-CRYPT0'82, Plenum Press, 1983, pp. 199–203.

[11] CHAUM, D., **Blind signature systems**, Advances in Cryptology-CRYPT0'83, Plenum Press, 1984, pp. 153.

[12] CHAUM, D. and VAN HEYST, E., **Group signatures**, Advances in Cryptology-EUROCRYPT'91, Lecture Notes in Computer Science, vol. 547, Springer-Verlag, 1991, pp. 257–265.

[13] FIAT, A. and SHAMIR, A., **How to Prove Yourself: Practical Solutions to Identification and Signature Problems**, Proceedings of CRYPTO'86, Lecture Notes in Computer Science, Springer-Verlag, vol. 263, 1987, pp. 186–194.

[14] FUJISAKI, E. and OKAMOTO, T., **Statistical zero knowledge protocols to prove modular polynomial relations**, In Advances in Cryptology-CRYPTO'97, Lecture Notes in Computer Science, vol. 1297, Springer-Verlag, 1997, pp. 16–30.

[15] FUJISAKI, E. and OKAMOTO, T., **A practical and provably secure scheme for publicly verifiable secret sharing and its applications**, In Advances in Cryptology-EUROCRYPT'98, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, 1998, pp. 32–46.

[16] HORSTER, P., MICHELS, M. and PETERSEN, H., **Meta-message recovery and meta-blind signature schemes based on the discrete logarithm problem and their applications**, Advances in Cryptology-ASIACRYPT'94, Lecture Notes in Computer Science, vol. 917, Springer-Verlag, 1995, pp. 224–237.

[17] LEE, W. and CHANG, C., **Efficient group signature scheme based on the discrete logarithm**, IEE Proc. Comput. Digit. Tech. 145, no. 1, 1998, pp. 15–18.

[18] LYSYANSKAYA, A. and RAMZAN, Z., **Group blind signature: A scalable solution to electronic cash**, Financial Cryptography (FC'98), Lecture Notes in Computer Science, vol. 1465, Springer-Verlag, 1998, pp. 184–197.

[19] KIM, S., PARK, S. and WON, D., **Convertible group signatures**, Advances in Cryptology-ASIACRYPT'96, Lecture Notes in Computer Science, vol. 1163, Springer-Verlag, 1996, pp. 311–321.

[20] PARK, S., LEE, I. and WON, D., **A practical group signature**, Proceedings of the 1995 Japan-Corea Workshop on Information Security and Cryptography, 1995, pp. 127–133.

[21] KIM, S., PARK, S. and WON, D., **ID-based group signature schemes**, Electronics Letters, 1997, pp. 1616–1617.

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

[23] POPESCU, C., **An efficient group blind signature scheme based on the Strong RSA assumption**, Romanian Journal of Information Science and Technology, Volume 3, Number 4, 2000, 365–374.

[24] SCHNORR, C.P., **Efficient signature generation for smart cards**, Journal of Cryptology, 4(3): 1991, pp. 239–252.

[25] TSENG, Y. and JAN, J., **A novel ID-based group signature**, In T.L. Hwang and A.K. Lenstra editors, 1998 International Computer Symposium, Workshop on Cryptography and Information Security, Tainan, 1998, pp. 159–164.

[26] TSENG, Y. and JAN, J., **Improved group signature scheme based on the discrete logarithm problem**, Electronics Letters 35, no. 1, 1999, pp. 37–38.

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 **XLVI**, Number 1, 2001

RECURSIVE RULES FOR DEMULTIPLEXERS EXPANDING

ANCA VASILESCU

Abstract. This paper introduces a model for the operation of the demultiplexers based on the CCS language. The main result is a set of recursive rules for the one-dimensional expanding of demultiplexers. Starting from the CCS model for 1×2 DMUX and 1×2^2 DMUX we shall infer the CCS model for the general case of a 1×2^n DMUX.

Keywords: recursive rule, demultiplexer, CCS model

1. Introduction

An important part of the internal structure of digital computers consists of digital and logic circuits: combinational or sequential. A demultiplexer (DMUX) [2, 3, 5] is a combinational logic circuit designed for receiving a value from its single input line and transferring that value to one of its multiple output lines. In addition, a DMUX has a set of special input lines for selecting which of the output line will receive the input signal.

Intuitively, the binary combination of selection lines values represents the index of the output line selected to transfer the value received from the input line. So that, there has to be a relation between the number of output lines and the number of selection lines of a DMUX. A *DMUX of type* n has one INPUT line, n SELECTION lines and 2^n OUTPUT lines. Usually, such a DMUX is a 1×2^n DMUX.

The operation of a DMUX is based on the different kinds of activities, such as deciding if the DMUX is or not valid at one moment, loading a binary value on a particular line, interpreting the combination of selection values. All these activities can be regarded as communication between different components of the unit of DMUX.

Considering this, it is proper to model the operation of a DMUX with an algebraic language like CCS, Calculus of Communicating Systems [1, 4]. In such a model, each of the activities of the demultiplexer can be associated with an action of a CCS agent. Besides, each state of the DMUX could be a special agent in the CCS model for the demultiplexer.

2000 *Mathematics Subject Classification.* 68Q85.
1998 *CR Categories and Descriptors.* B.6.1 [**Hardware**]: Logic Design – *Design Styles*.

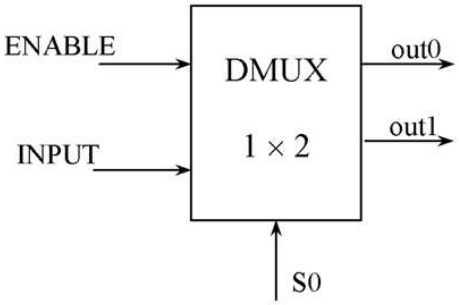


FIGURE 1. Diagrammatic representation of a 1×2 DMUX

In order to increase the power of the systems there is an efficient method to expand more identical systems to obtain a better one. In our case, we shall consider the problem of obtaining a demultiplexer of type n from demultiplexers of type $n-1$.

2. Modeling demultiplexers with CCS

In this section we shall use the algebraic language CCS (Calculus of Communicating Systems) to describe a demultiplexer (DMUX). Moreover, we shall infer a set of relations, some recursive, for expressing the method of obtaining the CCS model for a DMUX of type n, based on the model for DMUX of type $n-1$. Practically, it is useful to expand two or more demultiplexers to a demultiplexer with a large number of outputs.

2.1. **A CCS model for a 1×2 DMUX.** A 1×2 DMUX, or a DMUX of type 1, receives information from its single INPUT data line and directs it to one of the 2 OUTPUT lines, namely out0 and out1. The selection of the particular output data line is determined by a single SELECTION input line, namely S0. The bit value of this SELECTION line determines which output line receives the input value in order to transfer it to output.

For the reason of this paper, we add to this basic model an ENABLE input to control the operation of the unit. When the ENABLE value is 0, the outputs are disabled.

We propose for this unit the CCS model in Figure 3:

For this model, the possible actions are named 0, 1, INPUT, out0 and out1. Action named 0 arises when the unit accepts an input bit 0 from the ENABLE line or from the SELECTION line S0 (see Figure 1). The same for the action named 1. An action INPUT arises when the unit reads the input signal from the INPUT line. Action out0 arises when the unit transfers the input value to the selected

DMUX = DMUX'
DMUX' = 0.DMUX' + 1.D (E)
D = 0.SEL0 + 1.SEL1 (S)
SEL0 = INPUT. .DMUX'
SEL1 = INPUT. .DMUX'

FIGURE 2. CCS model for the 1×2 DMUX

D = 0.D0 + 1.D1
D0 = SEL0
D1 = SEL1

FIGURE 3. Definition of the agent D for a 1×2 DMUX

OUTPUT line out0. Action out1 arises when the unit transfers the input value to the selected OUTPUT line out1.

According to the syntax of CCS [4], the name of output ports, here $\overline{\text{out0}}$ and $\overline{\text{out1}}$, has to have the label overbared.

The agents defined in Code 1 are DMUX, DMUX', D, SEL0, SEL1.

In terms of transition representation, we can describe the operation of the demultiplexer as follows. The unit is initially in state DMUX'. The equation (E) from Figure 2 represents the role of ENABLE input. So that, while the bit value on ENABLE is 0, the unit is constantly in state DMUX'. When the value of ENABLE input is 1, the unit is changing to state D.

In Figure 2 the equation (S) means that the unit reads the bit value from the SELECTION input line. If this value is 0 the unit is changing to state SEL0, otherwise it is changing to state SEL1. The definitions of SEL0 and SEL1 mean that the signal from INPUT is copied on the suitable OUTPUT and the unit returns to the initial state DMUX'.

In order to prepare the recursive rule for generating the higher degree DMUX, it is useful to redefine the expression for the agent D from the Figure 2 as follows.

We have just introduced two new agents, D0 and D1.

2.2. **A CCS model for a 1×2^2 DMUX.** A 1×2^2 DMUX, or a DMUX of type 2, has one INPUT line, 2^2 OUTPUT lines — namely out0, out1, out2 and out3 — and 2 SELECTION lines — namely S0 and S1. In addition, we add the ENABLE input.

Like in the general model, the operation of this DMUX consists in transferring the INPUT value to one of the four OUTPUT lines, which is determined by the bit combinations of SELECTION line values.

For the 2-dimension DMUX, the CCS model could be:

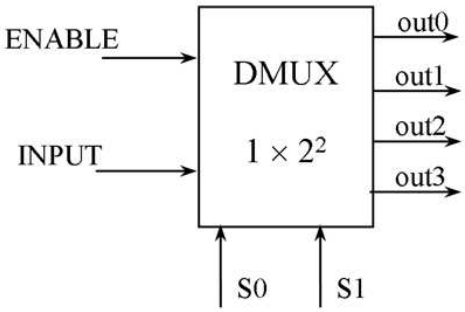


FIGURE 4. Diagrammatic representation of a 1×2^2 DMUX

DMUX = DMUX'
DMUX' = 0.DMUX' + 1.D (E)
D = 0.0.SEL0 + 0.1.SEL1 + 1.0.SEL2 + 1.1.SEL3 (S)

SEL0 = INPUT.$\overline{\text{out0}}$.DMUX'
SEL1 = INPUT.$\overline{\text{out1}}$.DMUX'
SEL2 = INPUT.$\overline{\text{out2}}$.DMUX'
SEL3 = INPUT.$\overline{\text{out3}}$.DMUX'

FIGURE 5. CCS model for the 1×2^2 DMUX

In this model, the meaning of the CCS constants is the same as in Code 1, but it is important to note that there are four bit combinations for the SELECTION input line values: 00, 01, 10, 11. For the demultiplexer from Figure 4, each of these words of 0 and 1 represents a different OUTPUT line selected to receive the INPUT value and, implicitly, a different current agent.

As in the case of 1×2 DMUX (see Figures 2 and 3) we redefine the expression for the agent D from Figure 5 as follows.

Note that for this representation we used much more constants.

2.3. **Recursive rules for DMUX expanding.** The first problem to solve now is to unify the notations used in Code 1and Code 3, namely to make the difference between the agents D from the simulation of 1×2 DMUX and 1×2^2 DMUX (see Figures 5 and 6). So that, we shall use an upper index to represent the type of current DMUX.

For a 1×2 DMUX, combining Figures 2 and 3, we have the next CCS representation:

$$\begin{aligned}
&\mathrm{D} = 0.\mathrm{D0} + 1.\mathrm{D1}\\
&\\
&\mathrm{D0} = 0.\mathrm{D00} + 1.\mathrm{D01}\\
&\mathrm{D1} = 0.\mathrm{D10} + 1.\mathrm{D11}\\
&\\
&\mathrm{D00} = \mathrm{SEL00}\\
&\mathrm{D01} = \mathrm{SEL01}\\
&\mathrm{D10} = \mathrm{SEL10}\\
&\mathrm{D11} = \mathrm{SEL11}\\
&\\
&\mathrm{SEL00} = \mathrm{SEL0}\\
&\mathrm{SEL01} = \mathrm{SEL1}\\
&\mathrm{SEL10} = \mathrm{SEL2}\\
&\mathrm{SEL11} = \mathrm{SEL3}
\end{aligned}$$

FIGURE 6. Definition of the agent D for a 1×2^2 DMUX

$$\begin{aligned}
&\mathrm{DMUX} = \mathrm{DMUX'}\\
&\mathrm{DMUX'} = 0.\mathrm{DMUX'} + 1.\mathrm{D}^{(1)} \quad \text{(E)}\\
&\\
&\mathrm{D}^{(1)} = 0.\mathrm{D0}^{(0)} + 1.\mathrm{D1}^{(0)} \quad \text{(S)}\\
&\mathrm{D0}^{(0)} = \mathrm{SEL0}\\
&\mathrm{D1}^{(0)} = \mathrm{SEL1}\\
&\\
&\mathrm{SEL0} = \mathrm{INPUT}.\overline{\mathrm{out0}}.\mathrm{DMUX'}\\
&\mathrm{SEL1} = \mathrm{INPUT}.\overline{\mathrm{out1}}.\mathrm{DMUX'}
\end{aligned}$$

FIGURE 7. Detailed CCS model for the 1×2 DMUX

For a 1×2^2 DMUX, combining Figures 5 and 6 we have the next CCS representation:

Note as very important that each name of the agent *memories* the sequence of previous actions already done.

We define

$$L_n = \{w \in 0, 1^\star |\ |w| = n\}$$

as the set of n-dimensional words over $\{0, 1\}$ and

$$L = \{w \in \{0,1\}^\star || w| \leq n\} = \bigcup_{k=1}^{0} L_n.$$

As a generalization, we propose the next CCS model for a 1×2^n DMUX:

$$\begin{aligned}
&\text{DMUX} = \text{DMUX'}\\
&\text{DMUX'} = 0.\text{DMUX'} + 1.\text{D}^{(2)} \quad \text{(E)}\\
&\\
&\text{D}^{(2)} = 0.\text{D0}^{(1)} + 1.\text{D1}^{(1)} \quad \text{(S)}\\
&\text{D0}^{(1)} = 0.\text{D00}^{(0)} + 1.\text{D01}^{(0)}\\
&\text{D1}^{(1)} = 0.\text{D10}^{(0)} + 1.\text{D11}^{(0)}\\
&\\
&\text{D00}^{(0)} = \text{SEL0}\\
&\text{D01}^{(0)} = \text{SEL1}\\
&\text{D10}^{(0)} = \text{SEL2}\\
&\text{D11}^{(0)} = \text{SEL3}\\
&\\
&\text{SEL0} = \text{INPUT}.\overline{\text{out0}}.\text{DMUX'}\\
&\text{SEL1} = \text{INPUT}.\overline{\text{out1}}.\text{DMUX'}\\
&\text{SEL2} = \text{INPUT}.\overline{\text{out2}}.\text{DMUX'}\\
&\text{SEL3} = \text{INPUT}.\overline{\text{out3}}.\text{DMUX'}
\end{aligned}$$

FIGURE 8. Detailed CCS model for the 1×2^2 DMUX

$$\begin{aligned}
&\text{DMUX} = \text{DMUX'}\\
&\text{DMUX'} = 0.\text{DMUX'} + 1.\text{D}^{(n)}\\
&\\
&\text{Dw}^{(k)} = 0.\text{Dw0}^{(k1)} + 1.\text{Dw1}^{(k1)}, \text{ for each } k = \overline{n,1} \text{ and } w \in L_{nk} \quad (\star)\\
&\text{Dw}^{(0)} = \text{SELi, for each } w \in L_n, \text{ where } i_{(10)} = w_{(2)} \quad (\star\star)\\
&\text{SELi} = \text{INPUT}.\overline{\text{outi}}.\text{DMUX'},\ i = \overline{0,\ 2^n - 1}
\end{aligned}$$

FIGURE 9. CCS model for a 1×2^n DMUX

In $(\star\star)$ we have to see the word w over $\{0, 1\}$ as a binary representation of the decimal value i.

The most important problem for a correct simulation of a 1×2^n DMUX is to assure that all the bit combinations are generated on the SELECTION lines. For our model this means to demonstrate that the $(\star)$ and $(\star\star)$ relations build all the elements of L_n. Hence, because every word from L_n is bijectively the representation of a decimal number from 0 to 2^n1, we can say that the model defines all the 2^n agents of the type SEL.

Theorem. The relation

$$Dw^{(k)} = 0.Dw0^{(k1)} + 1.Dw1^{(k1)}, fork = \overline{\text{n,1}} \text{ and } w \in L_{nk},\ (T)$$

generates the Dw agents for all the words w from the language L.

Proof. For $k = n$ we have $D^{(n)} = 0.D0^{(n1)} + 1.D1^{(n1)}$, because $|w| = nk = 0$ means $w = \lambda$, the empty word. This level defines the agent D and generates for this the agents $D0$ and $D1$.

The agent $D^{(n)}$ is completely defined if $D0^{(n1)}$ and $D1^{(n1)}$ are defined. That is done by applying the relation (T) for $k = n1$ and $w \in L_1 = \{0, 1\}$.

$$D0^{(n1)} = 0.D00^{(n2)} + 1.D01^{(n2)}$$

$$D1^{(n1)} = 0.D10^{(n2)} + 1.D11^{(n2)}$$

Until now, we have defined the agents D, $D0$, and $D1$ and we have generated the subset $\{\lambda, 0, 1, 00, 01, 10, 11\}$ of L, that means the words of length zero, one and two.

We prove by structural induction that for a fixed value of k, the relation (T) has already generated all the agents Dw with $w \in \{w \in \{0, 1\}^\star || w| \leq n - k + 1\}$. The value of k decreases from n to 1.

The inductive hypothesis is already verified by the previous relations written for $k = n$ and $k = n - 1$. We suppose that for a fixed value of k the relation (T) has already generated all the agents Dw with $w \in \{w \in \{0, 1\}^\star || w| \leq n - k + 1\}$ and we argue that for $k - 1$ the relation (T) generates all the agents Dw with $w \in \{w \in \{0, 1\}^\star || w| \leq n - k + 2\}$.

The induction step consists in proving that the next value of k adds all the words w which have the length with one unit greater than the words already generated. This result is obvious because the relation (T) for $k - 1$, i.e.

$$Dw^{(k-1)} = 0.Dw0^{(k2)} + 1.Dw1^{(k2)}, w \in L_{nk+1}$$

defines the agent Dw for $|w| = n - k + 1$ and generates the new agents $Dw0$ and $Dw1$. These two agents represent the binary words $w0$ and $w1$ obtained from w by adding the suffix 0 or 1. Hence, the new words $w0$ and $w1$ have the length equal with the length of w plus one.

Based on this induction we can say now that the relation (T) for $k = 1$ generates all the words $w \in \{w \in \{0, 1\}^\star || w| \leq n\} = L$. □

Corollary. The relation

$$Dw^{(0)} = SELi, \text{ for each } w \in L_n, \text{ where } i_{(10)} = w_{(2)}$$

generates 2^n agents of the type SEL.

Proof. This issue is obvious considering the next two elementary observations. Firstly, the set L_n has 2^n elements that represent all the n- dimensional bit combinations, from 0^n to 1^n. Secondly, the transformation of numbers from one base to another is a one-to-one function. Hence, the relation $(\star\star)$ defines all the decimal numbers from 0 to $2^n - 1$ which become the indices for the agents SEL. □

3. Conclusions

Practically, the problem of expanding the dimension of a logic circuit like demultiplexer is a very important one because it is useful to enclose more circuits within a single integrated circuit package. Many well-known works [2, 5] refer to this subject in specific terms for logic circuits (gates, integrated circuits, blocks of diagrams and so on). Based on the support of the CCS language [4] we have inferred a set of recursive rules for describing the operation of a 1×2^n DMUX.

The relations from Figure 9 are important not only as a theoretical result, but they connect two different approaches: the diagram representation (Figures 1 and 4) and the algebraic representation (Figures 7 and 8).

Moreover, the use of languages L_n and L in the above considerations suggests working further on a (dia)grammatic representation based on the operation of suitable automata for this model. Because of the importance of the length of the words w, it is certain that this representation involves a Turing machine, perhaps a particular model of it.

References

[1] Bruns G., Distributed Systems Analysis with CCS, International Series in Computer Science, C.A.R. Hoare Series Editor, Prentice Hall, London, 1997
[2] Hennessy J.L., Patterson D.A., Computer Organization and Design — the hardware/ software interface, Morgan Kaufmann Publishers, San Francisco, CA, 1998
[3] Mano M.M., Computer System Architecture, Prentice Hall, Englewood Cliffs, NJ, 1993
[4] Milner R., Communication and Concurrency, International Series in Computer Science, C.A.R. Hoare Series Editor, Prentice Hall, London, 1989
[5] Tanenbaum A.S., Goodman J.R., Structured Computer Organization, Prentice Hall, Englewood Cliffs, NJ, 1999

Department of Computer Science, Transilvania University of Braşov
E-mail address: vasilex@info.unitbv.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

NOTE ON THE TOURNAMENTS DIGRAPHS

DĂNUŢ MARCU

Abstract. In this paper,we show that every k-partite tournament ($k \geq 2$) with at most one vertex of indegree zero contains a 2-4-kernel.

Keywords: directed graphs,k-partite tournaments, kernels.

A *tournament* is an orientation of a complete graph (e.g., see [3]). More generally, a *k-partite tournament* is an orientation of a complete k-partite graph. Landau [2] has observed that every vertex v of maximum outdegree in a tournament T is a $2-kernel$ (e.g., see [3]), i.e., for every vertex u in T there exists a directed path of length at most 2 from v to u. The purpose of this note is to establish an analogous result for k-partite tournaments. Let us define a *2- k-kernel* in a digraph (directed graph) D as a vertex v such that for every vertex u in D there exists a directed path of length at most k from v to u. Thus, a 2-kernel in a tournament is a 2-2-kernel. For references to work on tournaments 2-kernels, see [2, 3], and our terminology is the same as in [1, 3].

A necessary condition for a digraph (directed graph) to have a 2-k-kernel, for some k ,is that at most one vertex has indegree zero.We shall, therefore, consider such orientations. If u and v are distinct vertices in a bipartite tournament, which dominate the same vertices, then none of u or v is a 2-3-kernel. Hence, there are many bipartite tournaments with no 2-3-kernels.

Theorem 1. *If T is a k-partite tournament ($k \geq 2$) with at most one vertex of indegree zero, then T has a 2-4-kernel.*

Proof. Let $V_1, V_2, \ldots, V_k$ be the partite classes in T. Let $v_i \in V_i$ be a vertex of maximum outdegree among the vertices in V_i. Obviously, the subgraph of T, induced by $v_1, v_2, \ldots, v_k$, is a tournament and, therefore, has a 2-2-kernel, say v_1. We claim that v_1 is a 2-4-kernel in T. Let u be any vertex in T, say $u \in V_i$. We shall describe a directed path from v_1 to u of length at most 4. First, T has a directed path P of length at most 2 from v_1 to v_i. If T also has a directed path of length at most 2 from v_i to u, then we are finished. Otherwise, u and v_i dominate (and are dominated by) exactly the same vertices, since the outdegree of u is not bigger than that of v_i. If $i \geq 2$, then the predecessor x of v_i on P

2000 *Mathematics Subject Classification.* 05C40.

1998 *CR Categories and Descriptors.* G.2.2 [**Mathematics of Computing**]: Discrete Mathematics – *Graph Theory.*

dominates u. Replacing the arc (x, v_i) of P by (x, u) results in a directed path of length at most 2 from v_1 to u. So, assume that $i = 1$. We can assume that T has no directed path of length 2 from v_1 to u. Hence, v_1 and u dominate exactly the same vertices. As they do not both have indegree zero, there exists a vertex z dominating u (and v_1). Say $z \in V_j$, $j \geq 2$. By a previous case (where $i \geq 2$), we conclude that T has a (shortest) directed path Q of length at most 4 from v_1 to z. If Q has length at most 3, then we add the arc (z, u) to Q. If Q has length 4, then the minimality of Q implies that a predecessor y of z on Q dominates v_1. But, then, y also dominates u, since v_1 and u are dominated by the same vertices. Replacing the arc (y, z) of Q by (y, u) results in a directed path of length 4 from v_1 to u. This completes the proof. □

It should be pointed out that Theorem 1 can be extended to larger classes of oriented graphs (except that the constant 4 may have to be increased). For example, the method of Theorem 1 can be extended to the following

Theorem 2. *Let G be a graph whose complement is the disjoint union of complete graphs, cycles and paths. Then, every orientation of G with at most one vertex of indegree zero has a 2-6-kernel.* □

References

[1] J.C.Bermond and C.Thomassen, *Cycles in digraphs – a survey*, J. Graph Theory, 5(1981), 1–43.

[2] M.F.Bridgland and K.B.Reid, *Stability of kings in tournaments*, in *Progress in Graph Theory* (eds. J.A.Bondy and U.S.R.Murty), Academic Press, New York, 1984, 117–128.

[3] D.Marcu, *Generalized kernels with considerations to the tournament digraphs*, Mathematica (Cluj), 1–2 (1982), 57–63.

Str. Pasului 3, Sector 2, 70241-Bucharest, Romania

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

THE MV-ALGEBRA STRUCTURE OF RGB MODEL

DAN NOJE AND BARNABÁS BEDE

Abstract. The aim of this paper is to explore the MV-algebra structure of RGB colour space (see [5]). We start with the construction of the MV-algebra structure of one component of the RGB model (this component represents one colour of the three that gives us the colour of a pixel on the screen). Then we define an MV-algebra structure on RGB model. Using Chang's Subdirect Representation Theorem we prove that RGB model is a subdirect product of MV-algebras of one component.

1. Introduction

Fuzzy sets are well known for their applications to image processing. It is also well known that the fuzzy sets have an MV-algebra structure (see [4]). We intend to develop an similar structure on RGB model (see[5]). This will allows us to use MV-algebras in image processing.

First we recall some definitions and properties of MV-algebras (see e.g. [2], [3]) that will be used later.

Definition 1.1. An MV-algebra is an algebra $\langle A, \oplus, \neg, 0_A\rangle$ with a binary operation $\oplus$, a unary operation $\neg$ and a constant 0_A satisfying the following equations:

(MVi) $x \oplus (y \oplus z) = (x \oplus y) \oplus z$;
(MVii) $x \oplus y = y \oplus x$;
(MViii) $x \oplus 0_A = x$;
(MViv) $\neg\neg x = x$;
(MVv) $x \oplus \neg 0_A = \neg 0_A$;
(MVvi) $\neg(\neg x \oplus y) \oplus y = \neg(\neg y \oplus x) \oplus x$.

Remark 1.2. In particular, axioms (MV1)-(MV3) state that $\langle A, \oplus, 0_A\rangle$ is a commutative monoid. As usually, we denote an MV-algebra $\langle A, \oplus, \neg, 0_A\rangle$ by its universe A.

Remark 1.3. The constant 1_A and the operations $\odot$ and $\ominus$ are defined on each MV-algebra A as follows:

2000 *Mathematics Subject Classification.* 06D35.
1998 *CR Categories and Descriptors.* I.3.2 [**Computing Methodologies**]: Computer Graphics – *Graphic Systems*.

i) $1_A =_{def} \neg 0_A$;
ii) $x \odot y =_{def} \neg(\neg x \oplus \neg y)$;
iii) $x \ominus y =_{def} x \odot \neg y$.

The following identities are immediate consequences of (MV4):

(MV7) $\neg 1_A = 0_A$;
(MV8) $x \oplus y = \neg(\neg x \odot \neg y)$.

Axioms (MV5) and (MV6) can now be written as:

(MV5') $x \oplus 1_A = 1_A$;
(MV6') $(x \ominus y) \oplus y = (y \ominus x) \oplus x$.

Setting $y = \neg 0_A$ in MV6) we obtain:

(MV9) $x \oplus \neg x = 1_A$.

Following common usage, we consider that $\neg$ operation is more binding than any other operation. Also we consider that $\odot$ operation is more binding than $\oplus$ operation and $\ominus$ operation.

Definition 1.4. Let A be an MV-algebra and $x, y \in A$. We say that $x \leq y$ if and only if x and y satisfy one of the bellow equivalent conditions:

i) $\neg x \oplus y = 1_A$;
ii) $x \odot \neg y = 0_A$;
iii) $y = x \oplus (y \ominus x)$;
iv) there is an element $z \in A$ such that $x \oplus z = y$.

Remark 1.5. It follows that $\leq$ is a partial order, called the natural order of A.

Definition 1.6. An MV-algebra whose natural order is total is called an MV-chain.

Lemma 1.7. *In every MV-algebra A the natural order $\leq$ has the following properties:*

i) $x \leq y$ if and only if $\neg y \leq \neg x$;
ii) if $x \leq y$ then for each $z \in A$, $x \oplus z \leq y \oplus z$ and $x \odot z \leq y \odot z$;
iii) $x \odot y \leq z$ if and only if $x \leq \neg y \oplus z$.

Proposition 1.8. *On each MV-algebra A the natural order determines a lattice structure. Specifically, the join $x \vee y$ and the meet $x \wedge y$ of the elements x and y are given by*

i) $x \vee y = (x \ominus y) \oplus y$;
ii) $x \wedge y = x \odot (\neg x \oplus y)$.

Definition 1.9. The distance function $d : A \times A \to A$ is defined by

$$d(x, y) =_{def} (x \ominus y) \oplus (y \ominus x).$$

Proposition 1.10. *In every MV-algebra A we have:*

i) $d(x,y) = 0_A$ *if and only if* $x = y$;
ii) $d(x,y) = d(y,x)$;
iii) $d(x,z) \leq d(x,y) \oplus d(y,z)$;
iv) $d(x,y) = d(\neg x, \neg y)$;
v) $d(x \oplus s, y \oplus t) \leq d(x,y) \oplus d(s,t)$.

2. Construction of MV-algebra structure of one component of RGB model.

RGB model represents one of the most used models to determin a pixel's colour on the screen. The number of colours that can be displayed is directly influenced by the number of bits on which the colours are stored in the computers memory. Also it is influenced by the properties of the screen.

RGB (see [5]) is defined as the set of triplets $(red, green, blue)$ or (r, g, b). The numbers form triplets represent how much red, green, respectively blue contains the pixel's colour.

We consider the number that represents one component of the triplet, stored on t bits. Therefore as the set of possible values for one colour component of RGB model we consider the set

$$C =_{def} \left\{ x \in \mathbb{R} \mid 0 \leq x \leq 2^{t-1} \right\}.$$

We introduce a binary operation $\oplus$, a unary operation $\neg$ and a constant 0_C as follows (when $x, y \in C$):

$$x \oplus y =_{def} \min(2^{t-1}, x + y); \tag{2.1}$$

$$\neg x =_{def} 2^{t-1} - x; \tag{2.2}$$

and 0_C is represented by the real number 0.

Lemma 2.1. *The quadruple* $\langle C, \oplus, \neg, 0 \rangle$ *is an MV-algebra.*

Proof. For proving that $\langle C, \oplus, \neg, 0 \rangle$ is an MV-algebra we have to show that the axioms (MV1)-(MV6) are satisfied.

(MV1): From equation (2.1) we have

$$x \oplus (y \oplus z) = x \oplus \min(2^{t-1}, y + z)$$

$$= \min(2^{t-1}, x + \min(2^{t-1}, y + z)) = \min(2^{t-1}, x + y + z)$$

$$= \min(2^{t-1}, \min(2^{t-1}, x + y) + z) = \min(2^{t-1}, x + y) \oplus z = (x \oplus y) \oplus z.$$

(MV2): Also by equation (2.1) we have

$$x \oplus y = \min(2^{t-1}, x + y) = \min(2^{t-1}, y + x) = y \oplus x.$$

(MV3): Using equation (2.1) we obtain

$$x \oplus 0 = \min(2^{t-1}, x + 0) = \min(2^{t-1}, x) = x.$$

(MV4):From equation (2.2) we obtain

$$\neg\neg x = \neg(2^{t-1} - x) = 2^{t-1} - (2^{t-1} - x) = x.$$

(MV5): By equations (2.1) and (2.2) it is easy to see that

$$x \oplus \neg 0 = x \oplus (2^{t-1} - 0) = x \oplus 2^{t-1} = \min(2^{t-1}, x + 2^{t-1})$$
$$= 2^{t-1} = 2^{t-1} - 0 = \neg 0.$$

(MV6): For proving (MV6) we will transform each side of equation to the same expression.

Using equation (2.2) the left side of axiom (MV6) is

$$\neg(\neg x \oplus y) \oplus y = \neg((2^{t-1} - x) \oplus y) \oplus y.$$

Then applying equation (2.1) we obtain

$$\neg(\neg x \oplus y) \oplus y = \neg \min(2^{t-1}, 2^{t-1} - x + y) \oplus y.$$

Applying now several times the equations (2.1) and (2.2) we have

$$\neg(\neg x \oplus y) \oplus y = (2^{t-1} - \min(2^{t-1}, 2^{t-1} - x + y)) \oplus y = \max(0, x - y) \oplus y$$
$$= \min(2^{t-1}, \max(0, x - y) + y) = \min(2^{t-1}, \max(y, x)).$$

Since both x and y are less or equals then 2^{t-1}we have

$$\max(y, x) \leq 2^{t-1}$$

and then we obtain

$$\neg(\neg x \oplus y) \oplus y = \max(y, x).$$

Using now the commutativity of max function we obtain

$$\neg(\neg x \oplus y) \oplus y = \max(x, y) \qquad (1).$$

Using equation (2.2) the right side of axiom (MV6) is

$$\neg(\neg y \oplus x) \oplus x = \neg((2^{t-1} - y) \oplus x) \oplus x.$$

Then applying equation (2.1) we obtain

$$\neg(\neg y \oplus x) \oplus x = \neg \min(2^{t-1}, 2^{t-1} - y + x) \oplus x.$$

Applying now several times the equations (2.1) and (2.2) we have

$$\neg(\neg y \oplus x) \oplus x = (2^{t-1} - \min(2^{t-1}, 2^{t-1} - y + x)) \oplus x = \max(0, y - x) \oplus x$$
$$= \min(2^{t-1}, \max(0, y - x) + x) = \min(2^{t-1}, \max(x, y)).$$

Since both x and y are less or equals then 2^{t-1} we have

$$\max(x, y) \leq 2^{t-1}$$

and then we obtain

$$\neg(\neg y \oplus x) \oplus x = \max(x, y) \qquad (2).$$

From the equations (1) and (2) we obtain

$$\neg(\neg x \oplus y) \oplus y = \neg(\neg y \oplus x) \oplus x.$$

□

The constant 1_C and the operations $\odot$ and $\ominus$ are defined on MV-algebra C as follows:

i) $1_C =_{def} \neg 0$.

From equation (2.2) we obtain

$$1_C = 2^{t-1}. \tag{2.3}$$

ii) $x \odot y =_{def} \neg(\neg x \oplus \neg y)$.

Also by equation (2.2) we obtain

$$x \odot y = 2^{t-1} - ((2^{t-1} - x) \oplus (2^{t-1} - y)).$$

From equation (2.1) we have

$$x \odot y = 2^{t-1} - \min(2^{t-1}, 2^{t-1} - x + 2^{t-1} - y).$$

After calculations we obtain

$$x \odot y = \max(0, x + y - 2^{t-1}).$$

From equation (2.3) we have

$$x \odot y = \max(0, x + y - 1_C). \tag{2.4}$$

iii) $x \ominus y =_{def} x \odot \neg y$.

From equation (2.2) we obtain

$$x \ominus y = x \odot (2^{t-1} - y).$$

By the equations (2.3) and (2.4) we have

$$x \ominus y = \max(0, x - y). \tag{2.5}$$

Remark 2.2. We can introduce an order relation $\leq$ on $\langle C, \oplus, \neg, 0\rangle$ in the same way like in the general case (see Definition 1.4).

It is easy to see that the natural order on $\langle C, \oplus, \neg, 0\rangle$ is a total order and thus the MV-algebra C is an MV-chain. Since $C \subseteq \mathbb{R}$, it is obvious that C is a complete latice.

The order relation on MV-algebra C is induced by the order relation of real numbers.

The distance function $d : C \times C \to C$ is defined by:

$$d(x, y) =_{def} (x \ominus y) \oplus (y \ominus x)$$

Using equation (2.5) we obtain

$$d(x, y) = \min(1_C, \max(0, x - y) + \max(0, y - x)) = \min(1_C, |x - y|).$$

Since both x and y are less or equals then 1_C we have

$$|x - y| \leq 1_C$$

and then we obtain

$$d(x, y) = |x - y| . \tag{2.6}$$

Remark 2.3. The distance function is defined following the general case (See Definition 1.9). It is obvious that all the properties of the distance are fullfiled (see Proposition 1.10).

Observe also that we obtain the classical Euclidean distance.

3. Construction of MV-Algebra structure of RGB model

In the previous section we have considered the set C as the set of possible values for one colour component of RGB model (see [5]). We also have proved that $\langle C, \oplus, \neg, 0\rangle$ is an MV-algebra.

In this section we will introduce on RGB model an MV-algebra structure. Let consider the set:

$$RGB =_{def} \{(c_1, c_2, c_3) \mid c_i \in C,\ i \in \{1, 2, 3\}\} .$$

In other words RGB is the direct product of family $\{C_i\}_{i\in\{1,2,3\}}$, where

$$C_i = C \text{ for all } i \in \{1, 2, 3\} .$$

On RGB set we introduce the operations $\oplus$, $\neg$ and the constant 0 as follows:

$$(c_1, c_2, c_3) \oplus (d_1, d_2, d_3) =_{def} (c_1 \oplus d_1, c_2 \oplus d_2, c_3 \oplus d_3) \tag{3.1}$$

for each (c_1, c_2, c_3) and (d_1, d_2, d_3) from RGB;

$$\neg(c_1, c_2, c_3) =_{def} (\neg c_1, \neg c_2, \neg c_3) \tag{3.2}$$

for each (c_1, c_2, c_3) from RGB;

$$0_{RGB} =_{def} (0, 0, 0) \tag{3.3}$$

where 0 is the constant 0 on MV-algebra C.

Theorem 3.1. *The quadruple $\langle RGB, \oplus, \neg, 0_{RGB}\rangle$ is an MV-algebra.*

Proof. For proving that $\langle RGB, \oplus, \neg, 0_{RGB}\rangle$ is an MV-algebra we have to show that the axioms (MV1)-(MV6) are satisfied.

(MV1): From equation (3.1) we have

$$\begin{aligned}&(a_1,a_2,a_3)\oplus((b_1,b_2,b_3)\oplus(c_1,c_2,c_3))\\&=(a_1,a_2,a_3)\oplus(b_1\oplus c_1,b_2\oplus c_2,b_3\oplus c_3)\\&=(a_1\oplus(b_1\oplus c_1),a_2\oplus(b_2\oplus c_2),a_3\oplus(b_3\oplus c_3)).\end{aligned}$$

By Lemma 2.1, the associative law holds in the MV-algebra C

$$a_i\oplus(b_i\oplus c_i)=(a_i\oplus b_i)\oplus c_i \text{ for } i=1,2,3.$$

From this law we have

$$\begin{aligned}&(a_1,a_2,a_3)\oplus((b_1,b_2,b_3)\oplus(c_1,c_2,c_3))=\\&=((a_1\oplus b_1)\oplus c_1,(a_2\oplus b_2)\oplus c_2,(a_3\oplus b_3)\oplus c_3).\end{aligned}$$

From this equality and from equation (3.1) we obtain

$$\begin{aligned}&(a_1,a_2,a_3)\oplus((b_1,b_2,b_3)\oplus(c_1,c_2,c_3))\\&=(a_1\oplus b_1,a_2\oplus b_2,a_3\oplus b_3)\oplus(c_1,c_2,c_3)\\&=((a_1,a_2,a_3)\oplus(b_1,b_2,b_3))\oplus(c_1,c_2,c_3).\end{aligned}$$

(MV2): From equation (3.1) we have

$$(a_1,a_2,a_3)\oplus(b_1,b_2,b_3)=(a_1\oplus b_1,a_2\oplus b_2,a_3\oplus b_3).$$

Using the commutativity of MV-algebra C we obtain

$$(a_1,a_2,a_3)\oplus(b_1,b_2,b_3)=(b_1\oplus a_1,b_2\oplus a_2,b_3\oplus a_3).$$

From equation (3.1) we have

$$(a_1,a_2,a_3)\oplus(b_1,b_2,b_3)=(b_1,b_2,b_3)\oplus(a_1,a_2,a_3).$$

(MV3): From the equations (3.1) and (3.3) it is easy to see that

$$(a_1,a_2,a_3)\oplus(0,0,0)=(a_1\oplus 0,a_2\oplus 0,a_3\oplus 0)=(a_1,a_2,a_3)$$

since 0 is the neutral element of MV-algebra C.

(MV4): Applying several times the equation (3.2) we obtain

$$\begin{aligned}&\neg\neg(a_1,a_2,a_3)=\neg(\neg a_1,\neg a_2,\neg a_3)\\&=(\neg\neg a_1,\neg\neg a_2,\neg\neg a_3)=(a_1,a_2,a_3),\end{aligned}$$

since $\neg\neg a=a$ in MV-algebra C.

(MV5): From equation (3.2) we have

$$(a_1,a_2,a_3)\oplus\neg(0,0,0)=(a_1,a_2,a_3)\oplus(\neg 0,\neg 0,\neg 0).$$

Using this equality and from equation (3.1) we obtain

$$(a_1, a_2, a_3) \oplus \neg (0, 0, 0) = (a_1 \oplus \neg 0, a_2 \oplus \neg 0, a_3 \oplus \neg 0)$$
$$= (\neg 0, \neg 0, \neg 0) = \neg (0, 0, 0),$$

since $a \oplus \neg 0 = \neg 0$, in MV-algebra C.

(MV6): From equation (3.2) we have

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= \neg ((\neg a_1, \neg a_2, \neg a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3).$$

From equation (3.1) we obtain

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= \neg (\neg a_1 \oplus b_1, \neg a_2 \oplus b_2, \neg a_3 \oplus b_3) \oplus (b_1, b_2, b_3).$$

Applying successively equations (3.2) and (3.1) we obtain

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= (\neg (\neg a_1 \oplus b_1) \oplus b_1, \neg (\neg a_2 \oplus b_2) \oplus b_2, \neg (\neg a_1 \oplus b_3) \oplus b_3).$$

By (MV6) of MV-algebra C applied for each component we have

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= (\neg (\neg b_1 \oplus a_1) \oplus a_1, \neg (\neg b_2 \oplus a_2) \oplus a_2, \neg (\neg b_3 \oplus a_3) \oplus a_3).$$

Applying now successively equation (3.1) we obtain

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= (\neg (\neg b_1 \oplus a_1), \neg (\neg b_2 \oplus a_2), \neg (\neg b_3 \oplus a_3)) \oplus (a_1, a_2, a_3).$$

By equation (3.2) we have

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= \neg (\neg b_1 \oplus a_1, \neg b_2 \oplus a_2, \neg b_3 \oplus a_3) \oplus (a_1, a_2, a_3).$$

Applying equations (3.1) and (3.2) we obtain

$$\neg (\neg (a_1, a_2, a_3) \oplus (b_1, b_2, b_3)) \oplus (b_1, b_2, b_3)$$
$$= \neg ((\neg b_1, \neg b_2, \neg b_3) \oplus (a_1, a_2, a_3)) \oplus (a_1, a_2, a_3)$$
$$= \neg (\neg (b_1, b_2, b_3) \oplus (a_1, a_2, a_3)) \oplus (a_1, a_2, a_3).$$

□

The constant 1_{RGB}, and the operations $\odot$ and $\ominus$ are defined as follows:

i) $1_{RGB} =_{def} \neg (0, 0, 0)$.

From equations (2.2) and (3.2) we obtain

$$1_{RGB} = (\neg 0, \neg 0, \neg 0) = \left(2^{t-1}, 2^{t-1}, 2^{t-1}\right).$$

By equation (2.3) we obtain

$$1_{RGB} = (1_C, 1_C, 1_C)$$

ii) $(a_1, a_2, a_3) \odot (b_1, b_2, b_3) =_{def} \neg(\neg (a_1, a_2, a_3) \oplus \neg (b_1, b_2, b_3))$.
Applying equation (3.2) we have:

$$(a_1, a_2, a_3) \odot (b_1, b_2, b_3) = \neg ((\neg a_1, \neg a_2, \neg a_3) \oplus (\neg b_1, \neg b_2, \neg b_3)) .$$

By equation (3.1) we obtain

$$(a_1, a_2, a_3) \odot (b_1, b_2, b_3) = \neg (\neg a_1 \oplus \neg b_1, \neg a_2 \oplus \neg b_2, \neg a_3 \oplus \neg b_3) .$$

Applying equation (3.2) we have

$$(a_1, a_2, a_3) \odot (b_1, b_2, b_3) = (\neg (\neg a_1 \oplus \neg b_1) , \neg (\neg a_2 \oplus \neg b_2) , \neg (\neg a_3 \oplus \neg b_3))$$

By definition of $\odot$ on MV-algebra C we obtain:

$$(a_1, a_2, a_3) \odot (b_1, b_2, b_3) = (a_1 \odot b_1, a_2 \odot b_2, a_3 \odot b_3) \tag{3.4}$$

iii) $(a_1, a_2, a_3) \ominus (b_1, b_2, b_3) =_{def} (a_1, a_2, a_3) \odot \neg (b_1, b_2, b_3)$.
By equation (3.1) and 3.4 we have:

$$(a_1, a_2, a_3) \ominus (b_1, b_2, b_3) = (a_1, a_2, a_3) \odot (\neg b_1, \neg b_2, \neg b_3) .$$

From equation (3.4) we obtain

$$(a_1, a_2, a_3) \ominus (b_1, b_2, b_3) = (a_1 \odot \neg b_1, a_2 \odot \neg b_2, a_3 \odot \neg b_3) .$$

By definition of $\ominus$ on MV-algebra C we obtain:

$$(a_1, a_2, a_3) \ominus (b_1, b_2, b_3) = (a_1 \ominus b_1, a_2 \ominus b_2, a_3 \ominus b_3) . \tag{3.5}$$

Remark 3.2. We can introduce a partial order relation $\leq$ on $\langle RGB, \oplus, \neg, 0_{RGB} \rangle$ in the same way like in the general case.

It is easy to verify that:

i) $(a_1, a_2, a_3) \leq (b_1, b_2, b_3)$ if and only if $a_i \leq b_i$, for each $i = 1, 2, 3$.
ii) $(a_1, a_2, a_3) \vee (b_1, b_2, b_3) = (a_1 \vee b_1, a_2 \vee b_2, a_3 \vee b_3)$
iii) $(a_1, a_2, a_3) \wedge (b_1, b_2, b_3) = (a_1 \wedge b_1, a_2 \wedge b_2, a_3 \wedge b_3)$.

Proposition 3.3. *RGB has a complete lattice structure, and this implies that RGB is an MV σ-algebra (see* [1]*).*

Remark 3.4. Convergent sequences can be defined on RGB, using MV σ-algebra properties of RGB (see [1]).

4. Distance on RGB

To use the MV-algebra structure of RGB model for image processing, we have to define a distance function $d : RGB \times RGB \to RGB$ as follows

$$d\left((a_1, a_2, a_3), (b_1, b_2, b_3)\right) =_{def}$$
$$= \left((a_1, a_2, a_3) \ominus (b_1, b_2, b_3)\right) \oplus \left((b_1, b_2, b_3) \ominus (a_1, a_2, a_3)\right).$$

From equation (3.5) we have

$$d\left((a_1, a_2, a_3), (b_1, b_2, b_3)\right)$$
$$= \left(a_1 \ominus b_1, a_2 \ominus b_2, a_3 \ominus b_3\right) \oplus \left(b_1 \ominus a_1, b_2 \ominus a_2, b_3 \ominus a_3\right).$$

Applying equation (3.1) we obtain

$$d\left((a_1, a_2, a_3), (b_1, b_2, b_3)\right)$$
$$= \left((a_1 \ominus b_1) \oplus (b_1 \ominus a_1), (a_2 \ominus b_2) \oplus (b_2 \ominus a_2), (a_3 \ominus b_3) \oplus (b_3 \ominus a_3)\right).$$

From definition of the distance function on MV-algebra C we obtain:

$$d\left((a_1, a_2, a_3), (b_1, b_2, b_3)\right) = \left(d(a_1, b_1), d(a_2, b_2), d(a_3, b_3)\right).$$

In Section 2 of this paper we have shown that $\langle C, \oplus, \neg, 0\rangle$ is an MV-chain. In Section 3 we have defined RGB set as a direct product of C set. It is obviously that RGB set is also a subdirect product of C set. From Chang's Subdirect Representation Theorem (see [2]) we obtain:

Proposition 4.1. *The MV-algebra $\langle RGB, \oplus, \neg, 0_{RGB}\rangle$ is a subdirect product of the MV-chains $\langle C, \oplus, \neg, 0\rangle$.*

References

[1] Ban, A. I. (2001) "Ergodicity of MV-dynamical systems", Soft Computing 5: 327-333;
[2] Cignoli, R., D'Ottaviano, Itala M.L. and Mundici, D. (2000) "Algebraic Foundations of many-valued Reasoning", Kluwer Academic Publishers , Trends in Logic, Volume 7;
[3] Chang, C.C. (1958) "Algebraic Analysis of many-valued logic", Trans Amer Math Soc 88: 467-490;
[4] Dumitrescu, D., Lazzerini, B., Jain, L.C. (2000) "Fuzzy Sets and their Application to Clustering and Training", Boca Raton, Fl.;
[5] Ionescu, F. (2000) "Grafica in realitatea virtuala", Editura Tehnica, Bucuresti.

Department of Mathematics, University of Oradea, Armatei Romane 5, 3700 Oradea, Romania

E-mail address: dnoje|bbede@math.uoradea.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

CHARACTER RECOGNITION USING MORPHOLOGICAL TRANSFORMATIONS

VASILE PREJMEREAN AND SIMONA MOTOGNA

Abstract. Starting from a digital image representing a character (or, more general, an object), we will obtain formal descriptions of the object structures, formed from primitives and the relations between them, using morphological transformations (thinning, prunning, determining the corners, determining the primitives). From such a description we will construct a description grammar for the object under study, grammars that are used in the syntactical recognition of similar objects. The object being recognized has a corresponding description through morphological transformations that allows us to study if the description belongs to the language generated by the designed grammar.

1. Introduction

Generally speaking, we can say that an image transformation is dedicated to human eyes or is made for a pattern recognition reason.

This paper presents a method for character recognition using morphological transformations, whose purpose is to obtain certain structures formed from lines and curves (type 3 images [6]) needed for pattern recognition. The notion of morphology comes from animal and plants form study, but for us *morphological processing* ([3, 5]), means the determination of the object structures from their images.

2. Morphological Processing

Morphological processings consist in operations through which an object X is modified by a structuring element B, yielding to a form convenient for future processing (pattern recognition). The two interacting elements (X and B) are represented as sets in the Euclidian bidimensional space.

Most of the morphological operations can be defined by two basic operations, erosion and dilation described in the following.

2000 *Mathematics Subject Classification.* 68Q45, 68T10.

1998 *CR Categories and Descriptors.* F.4.2 [**Theory of Computation**]: Mathematical Logic and Formal Languages – *Grammars and Other Rewriting Systems*; I.5.2 [**Computing Methodologies**]: Pattern Recognition – *Design Methodology*.

Notation 2.1.

The translation of B in x denoted by B_x, is the translation of the structural element B such that the origin O_B is located in x.

Definition 2.1.

The erosion of X by B, denoted by $X \ominus B$, is the set of all points x such that B_x is included in X:

$$X \ominus B = \{x | B_x \subset X\}.$$

Remark: Erosion is an operation that decreases the object.

Definition 1.2.

The dilation of X by B, denoted by $X \oplus B$, is the set of all points x such that B_x and X have a nonempty intersection:

$$X \oplus B = \{x | B_x \cap X \neq \emptyset\}.$$

Remark: Dilation is an operation that increases the object.

The two presented basic operations have the following properties: translation invariant, distributivity, local knowledge, iteration, increasing, duality, and so on [3].

Next, we shall present some usual transformations obtained from the two basic operations described above (X^C denotes the complement of X).

a. *Hit-Miss*, denoted by $X * B$, verifies if a structure $B \in X$ and $B^C \in X^C$:

$$X * B = (X \ominus B) \cap (X^C \ominus B^C) = (X \ominus B) \cap (X \oplus B^C)^C =$$
$$= \quad (X \ominus B_{Ob}) \backslash (X \oplus B_{Bk}) \qquad (\textit{we denote } B \textit{ by } B_{Ob}, \textit{ and } B^C \textit{ by } B_{Bk})$$

because from:$(X^C \oplus B) = (X \ominus B)^C$ (duality property for all X and all B) it results that

$$(X \oplus B^C) = (X^C \ominus B^C)^C \quad (\textit{applied to } X^C \textit{ and } B^C). \tag{1}$$

B_{Ob} must be *matched* with the object X, and B_{Bk} with the ***Background***;

b. *Open* of X relative to B, denoted by X_B is the domain scanned by all of the translations of B included in X:

$$X_B = (X \ominus B) \oplus B$$

c. *Close* of X relative to B, denoted by X^B, is the reverse operation to *open*:

$$X^B = (X \oplus B) \ominus B$$

d. *Boundary determination* (δX):

$$\delta X = X \backslash (X \ominus G)$$

The usual structuring element is $G = \begin{matrix} o & o & o \\ o & \mathbf{O} & o \\ o & o & o \end{matrix}$ where $\begin{matrix} o & - & \text{object} \\ \mathbf{O} & - & \text{origin} \end{matrix}$

e. *Thinning*, using morphological transformation, is defined as follows:

$$X \otimes B = X \backslash (X * B)$$

The usual structuring element is $B = \begin{matrix} \mathrm{o} & \mathrm{o} & \mathrm{o} \\ * & \mathbf{O} & * \\ \mathbf{o} & \mathbf{o} & \mathbf{o} \end{matrix}$ where $\begin{matrix} \mathbf{o} & - & \text{object} \\ \mathrm{o} & - & \text{background} \\ \mathbf{O} & - & \text{origin} \\ * & - & \text{don't care} \end{matrix}$

To obtain a simetrical thinning we must apply successively the operation described above, using as structuring element the rotate object B:

$$X \otimes^s B = ((...((X \otimes B^1) \otimes B^2) \otimes ...) \otimes B^n),$$

where $B^1 = B$ and $B^i = Rotate(B^{i-1}), 2 \leq i \leq n$.

f. *Thicking* of X through B, denoted $X \odot B$, is the reverse operation to *thinning* and is defined as follows:

$$X \odot B = X \cup (X * B)$$

g. The *skeleton* of an object X, denoted by $S(X)$, is defined as:

$$S(X) = \bigcup_{n=0}^{n_{max}} s_n(x) = \bigcup_{n=0}^{n_{max}} [(X \ominus nG) \backslash (X \ominus nG)_G],$$

where n_{max} is the smallest n such that $X \ominus nG = \emptyset$.

The reconstructed object X is:

$$X = \bigcup_{n=0}^{n_{max}} [s_n(x) \oplus nG],$$

where $X \ominus nG = (((X \ominus G) \ominus G) \ominus \ldots) \ominus G$.

h. *Prunning* deletes (suppresses) the *parasite branches*, that can be the results after a thinning operation:

$$X_{pn} = X_1 \cup [(X_2 \oplus G) \cap X], \text{ where}$$

$$X_1 = X \otimes^s E;$$

$$X_2 = \bigcup_{j=1}^{8} [X_1 * E^j];$$

$$E = \begin{matrix} * & * & * \\ o & \mathbf{o} & o \\ o & o & o \end{matrix}$$

3. Character Recognition

There are several types of recognitions, and we use syntactical recognition because it has the advantage of being able to identify an infinity set of complex forms using a small number of production rules. Syntactical recognition of a form assumes *the identification of the primitives* that compund the form (these primitives must be easy to recognise) which is achieved with morphological processing, then *syntactical analysis* of the form in order to identify and perphaps obtain its structure, which will be presented in the next paragraph.

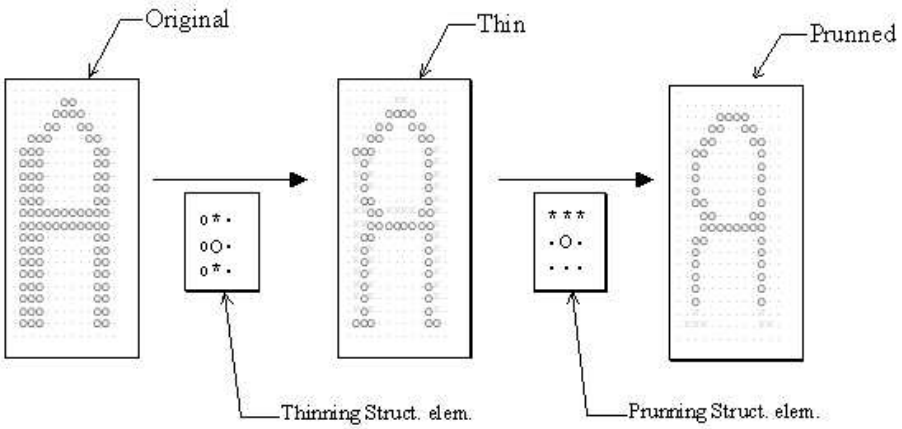


Figure 1. Chracter 'A' during recognitin: original, thinned and prunned

Character recognition can be divided in the following steps, as shown in Figure 1:

a) Thinning, described in Section 2, paragraph e;

Figure 2. The significance of signs from Figure 1

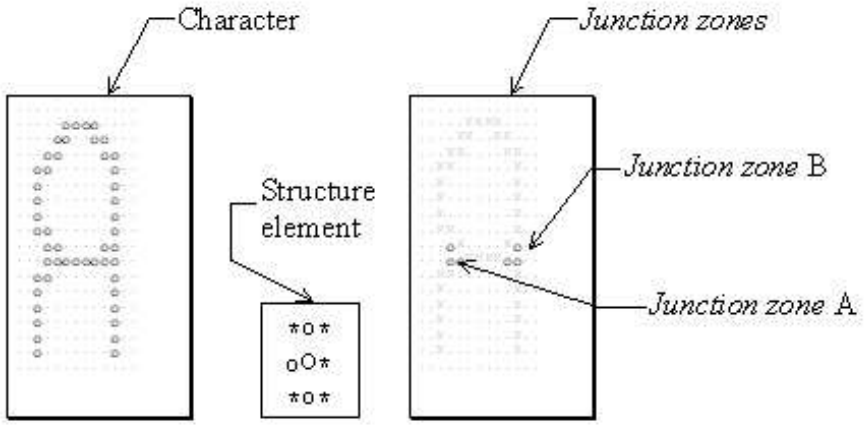


FIGURE 3. The junction zones for character 'A'

b) Then the object is prunned, according to the rules described in Section 2, paragraph h;
c) To obtain the *corners* (Junction zone) we may apply successively the hit-miss operation, using as structuring element the rotate object B, as in 3:

$$B = \begin{matrix} * & \mathbf{o} & * \\ \mathbf{o} & \mathbf{O} & * \\ * & \mathbf{o} & * \end{matrix} \quad \text{where} \quad \begin{matrix} \mathbf{o} & - & \text{object} \\ \mathbf{O} & - & \text{Origin} \\ * & - & \text{don't care} \end{matrix}$$

$$X \otimes B = \bigcup_{i=1}^{4} (X * B^i) \; where: \; B^1 = B \; and \; B^i = Rotate(B^{i-1}),\; 2 \le i \le 4.$$

d) To obtain the *primitives* that compose the image of the character we apply the hit-miss operation with the following structuring elements B:

$$- \; vertical \; lines : X|B = \bigcup_{i=1}^{3} (X \otimes B_i^{|}),\; 1 \le i \le 3,\; with:$$

$B_1^{\|}$			$B_2^{\|}$			$B_3^{\|}$		
*	**o**	*	*	**o**	*	*	**o**	*
o	**O**	*	**o**	**O**	*	**o**	**O**	*
*	**o**	*	*	**o**	*	*	**o**	*

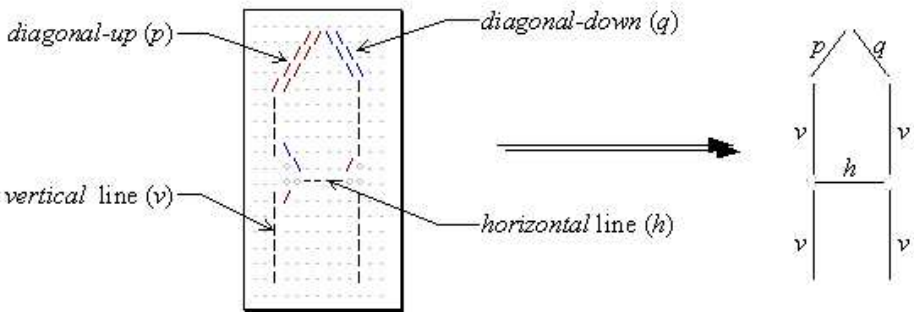


FIGURE 4. The primitives for character 'A'

where the notations have the same significance as in c).

$$- \ horizontal \ lines : X - B = \bigcup_{i=1}^{3} (X \otimes B_i^{-}), \ where \ B_i^{-} = Rotate(B_i^{|}), 1 \leq i \leq 3$$

$$- \ diagonal - down \ lines : X \backslash B = \bigcup_{i=1}^{2} (X \otimes B_i^{\backslash}), \ where$$

$$B_2^{\backslash} = Rotate^2(B_1^{\backslash}) \text{ and } B_1^{\backslash} = \begin{matrix} * & \mathbf{o} & * \\ * & \mathbf{O} & \mathbf{o} \\ * & * & * \end{matrix}$$

$$- \ diagonal - up \ lines : X/B = \bigcup_{i=1}^{2} (X \otimes B_i^{/}), \ where B_i^{/} = Rotate(B_i^{\backslash}), 1 \leq i \leq 2.$$

Now, it is easy for someone to recognise the primitives that compose the picture and after that to describe the character. It's not difficult to find the relations of relative positions of these primitives (see Figure 4).

4. GRAMAMTICAL DESCRIPTION

A form will be the result of several concatenation operations, that unifies the simplest forms in subforms more and more complex.

A form description language defines their structure, and a form description grammar defines the subform compounding rules.

Structural representations of a form is recommended when there exist complex concatenation relations between the primitives that have been used, because this representation gives us a graphical image of the way in which the primitives are interconnected.

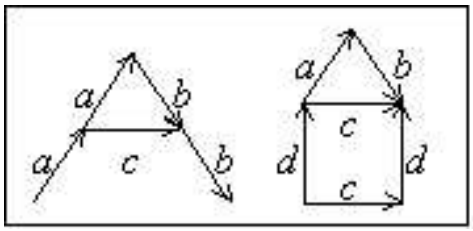


FIGURE 5. Forms built with 'diagonal' primitives

In order to build a form representation, we choose a starting point, which will be the initial node in the structural descrition. This node is then decomposed in two subforms together with the relation between them, as shown in Figure 8. If these subforms are also compound from other subforms, then we will continue the description until all the nodes will contain only primitives.

The syntactical recognition process assumes the following three steps:

- the selection of the primitives in order to obtain the form description alphabet;
- the form representation using a description language;
- the definition of a grammar, such that it will generate the description language.

The first two steps have already been discussed. We will focus next on some grammar classes, which are used for certain form representation methods.

The forms that can be represented through a primitive sequence (using primitive concatenation operation) are called *one-dimension forms*, and their corresponding grammars are called *one-dimension grammars*.

The grammar $G_1 = (\{S, A, C, D\}, \{a, b, c, d, (,), +, *, \neg\}, P, S)$, with the production rules: $P = \{S \to A/C, D \to (a+b)*c, A \to a+D+b, C \to ((\neg d)+c+d)*D\}$, generates the language $L(G_1) = \{a + (a + b) * c + b, ((\neg d) + c + d) * (a + b) * c\}$, where $+$ is the usual concatenation operator, $*$ is for parallel concatenation, and $\neg$ is the operator for extremes inversion [4, 1].

This language describes the two forms from Figure 5, built with the primitives $\{a, b, c, d\} = \{\nearrow, \searrow, \rightarrow, \uparrow\}$.

We will present now the image desciption model that uses grammars whose terminal symbols are graphical primitives (horizontal, vertical and diagonal lines, as shown in Figure 4) and predicates [2]. To each primitives we attach characteristics regarding their position in the image frame (framework coordinates). The predicates express relations that can exist between certain image components (primitives and compound forms). The compound forms correspond to the nonterminal symbols from a traditional grammar. For example, $on(x, y)$ ("x is over

y") is a predicat that has the value true if and only if all the components of object x are *on* all the components of object y. The relative positions of the image components can be obtained from their attached coordinates (as we shall see in the next section).

Such a grammar $G = (N, \Sigma, P, S)$ consists of the set of nonterminal symbols $N(S \in N)$, the set Σ of terminal symbols containing primitives and predicates, the set P of production rules (each of them defining a form) and the start symbol S. A production of this grammar has the form $A \rightarrow (\omega) : \alpha; \pi$, where $A \in N$ is a nonterminal representing the name of the defining graphical form, ω is a list of arguments of the production, α is a list of objects (primitives or compound objects), and π is a predicate (α and π are statements that should be true for arguments from the list ω.

In order to ilustrate these concepts, we will use as example the 'A' character from Figure 4. Suppose that the terminal alphabet is:

- types of graphical primitives: $/, \backslash, |, -$;
- predicates: 'in','on','near','right'.

The productions described above can be represented in a parsing tree. For this example, the corresponding tree is shown in Figure 8.

We shall present now an algorithm for *generating graphical subforms and for building the production rules* (corresponding to the first step of the syntactical recognition process).

At the begining, the description list ($ObIni$) contains the graphical primitives from the drawing (we denote by $GPNo$ the total number of graphical primitives), storing for each primitive p its type ($PrimitiveType(p)$), the window in which is framed ($Domain_p = (x_1, y_1, x_2, y_2)$) and the set of components ($MOb_p = \{p\}$). For each graphical primitive from the analized drawing we will store the following information:

$$(p, PrimitiveType(p), Domain_p, MOb_p = \{p\}), \quad (for\ each\ p = 1, 2, ..., GPNo).$$

This list is enriched with new compound objects ($NewOb$) formed from the ones existing in the list $ObIni$ and are in a convenient relative position, denoted by relation ϕ (for example, (s, d), if s is inside d, i.e. $Domain_s \subset Domain_d$). For a compound object o we will store:

$$(o, (s, d), \phi(s, d), Domain_o, MOb_o = MOb_s \cup MOb_d),$$

where $Domain_o$ is computed using the following formula:

$$x_1 := Minim(s.x_1, d.x_1), \quad x_2 := Maxim(s.x_2, d.x_2),$$
$$y_1 := Minim(s.y_1, d.y_1), \quad y_2 := Maxim(s.y_2, d.y_2).$$

(see Figure 6).

The addition of new elements is continued until it is no more possible ($NewOb = \emptyset$).

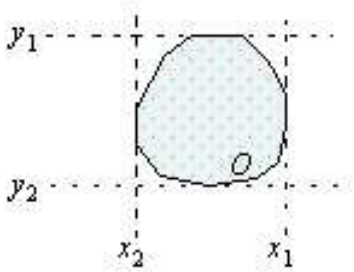


FIGURE 6. Coordinates specification for an object

Those objects o that contain all the graphical primitives of the drawing ($MOb_o \supset \{1, 2, ..., GPNo\}$), represent drawing descriptions.

Algorithm 3.1.

Initialize $ObIni = \{Ob_p, p = 1, GPNo\}$; {List of graphical primitives}
$ObAd := ObIni; LPrim := ObIni; o := m;$
Repeat
$NewOb := \emptyset; LOb := ObIni \cup ObAd;$
For each $\alpha = (s, d) \in LOb \times ObAd \cup ObAd \times LOb$ execute
For each relation ϕ execute { see the Table 1 }
If $\phi(s, d)$ *and* $Ob_s \cap Ob_d = \emptyset$ then {s ϕ d and no common primitives}
$o := o + 1$; *Compute* $Domain_o$;
$NewObi := NewOb \cup \{(o, (s, d), \phi(s, d), Domain_o, MOb_s \cup MOb_d)\}$;
$ObIni := LOb$; $ObAd := NewOb$;
Until $NewOb = \emptyset$; {no more new objects}

The set of descriptions: $= \{o | \{1, 2, ..., GPNo\} \subset MOb_o\}$.

Two objects can be in the relations presented in 1, representing relations between their coordinates (specified in 6).

ϕ	O_1 *in* O_2	O_1 *on* O_2	O_1 *near* O_2	O_1 *left* O_2
	$O_2.x_1 < O_1.x_1$	$O_1.y2 \leq O_2.y_1$	$O_1.x_2 \leq O_2.x_1$	$O_1.x_2 < O_2.x_1$
	$O_1.x_2 < O_2.x_2$	$O_1.y_2 \approx O_2.y_1$	$O_1.x_2 \approx O_2.x_1$	$O_1.x_2 \not\approx O_2.x_1$
	$O_2.y_1 < O_1.y_1$	$O_1.x_1 \approx O_2.x_1$	$O_1.y_1 \approx O_2.y_1$	$O_1.y_1 \approx O_2.y_1$
	$O_1.y_2 < O_2.y_2$	$O_1.x_2 \approx O_2.x_2$	$O_1.y_2 \approx O_2.y_2$	$O_1.y_2 \approx O_2.y_2$

TABLE 1. Definition of relations ϕ, where $\alpha \approx \beta$ means $|\alpha - \beta| \leq \varepsilon$, namely are very close to each other.

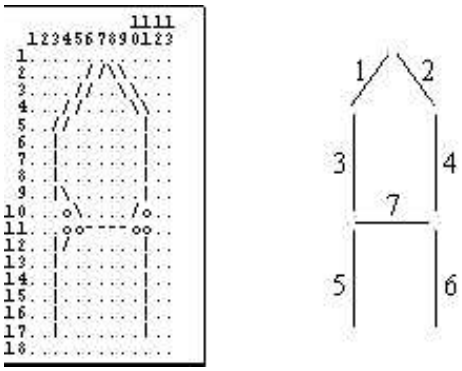


FIGURE 7. The numbers associated to primitives

Example 4.1.
We consider the drawing from Figure 4, and we will number the primitives (as in 7) in order to build the initial subform list, as presented in Table 2.

	Pr.	x_1	y_1	x_2	y_2
1.	/	3	2	7	5
2.	\	8	2	11	4
3.	|	3	6	3	9
4.	|	11	5	11	9
5.	|	3	11	3	16
6.	|	11	11	11	16
7.	-	6	10	9	10

TABLE 2. The primitives used in Example 4.1

Applying the predicates 'in', 'on', 'near' and 'left' to all object pairs from the initial list we will obtain the following list (when we add new elements, we also determine information regarding the new objects postions):

8-(1,2):near(1,2);
9-(3,4):left(3,4);
10-(3,5):on(3,5);
11-(4,6):on(4,6);
12-(5,6):left(5,6)

The process is repeated using the extended list (1-12). The new list elements obtained in the third step are:

13-(8,9):on(8,9);
14-(9,12):on(9,12);
15-(10,11):left(10,11);

The next step will produce the following elements:

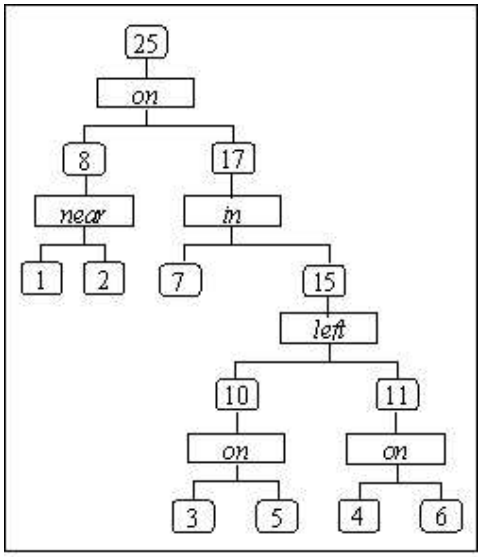


FIGURE 8. The parsing tree for character 'A'

16-(7,14):in(7,14);
17-(7,15):in(7,15);
18-(8,14):on(8,14);
19-(8,15):on(8,15);
20-(12,13):on(13,12);

Since there is no object that contains all the primitives, the process will continue, adding the last elements:

21-(7,18):in(7,18);
22-(7,19):in(7,19);
23-(7,20):in(7,20);
24-(8,16):on(8,16);
25-(8,17):on(8,17).

We notice that these last objects (21–25) contain all the primitives as components. The parsing tree corresponding to the object 25 is shown in Figure 8.

Next, we will replace variables with constant names, in three steps:

1) symbols from 8 to 20, representing intermediary compound objects will be replaced with nonterminals (from A to M):

A=8-(1 near 2)	E=12-(5 left 6)	I=16-(h in G)
B=9-(3 left 4)	F=13-(8 on 9)	J=17-(h in H)
C=10-(3 on 5)	G=14-(9 on 12)	K=18-(A on G)
D=11-(4 on 6)	H=15-(10 left 11)	L=19-(A on H)
		M=20-(F on E)

2) symbols from 1 to 7, representing the primitives will be replaced with the primitive names:

A=8-(p near q)	E=12-(v left v)	I=16-(7 in 14)
B=9-(v left v)	F=13-(A on B)	J=17-(7 in 15)
C=10-(v on v)	G=14-(B on E)	K=18-(8 on 14)
D=11-(v on v)	H=15-(C left D)	L=19-(8 on 15)
		M=20-(13 pe 12)

3) symbols from 21 to 25, representing final objects have the following associated grammar:

21: S → h in K;
K → A on G;
A → p near q;
G → B on E;
B → v left v;
E → v left v.

22: S → h in L;
L → A on H;
A → p near q;
H → C left D;
C → v on v ;
D → v on v.

23:S → h in M;
M → F on E;
E → v left v;
F → A on B;
A → p near q;
B → v left v.

24: S → A on I;
A → p near q;
I → h in G;
G → B on E;
B → v left v;
E → v left v.

25: S → A on J;
A → p near q;
J → h in H;
H → C left D;
C → v on v;
D → v on v.

We will proceed now with the reduction of the grammar, in order to obtain a simpler one. The reduction takes into consideration the following remarks:

a. we have some identical rules: C and D, B and E, respectively;
b. rules for K and L, I and J differ only by a symbol (G, respectively H), and we will use unification;
c. rules for M and F differ only by a symbol (F, respectively A), so we will use unification.

The final grammar, capable of generating the original sample is:

S → h in K | h in M | A on I;
K → A on G;
G → B on B | C left C;
M → M on B | A on B;
A → p near q;
B → v left v;
C → v on v ;
I → h in G;

Since the rule for the symbol M generates sequences of the form $A(on\ B)^n, n > 0$, we can extend it even more such that $n \geq 0$, yielding, eventually, to $M \rightarrow M\ on\ B|A$. This grammar generates character of the form shown in 9.

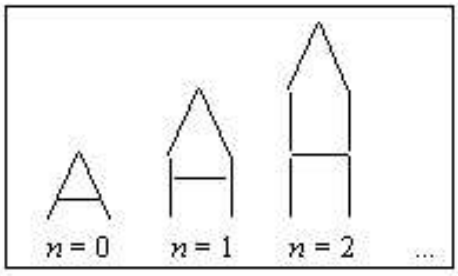


FIGURE 9. Forms obtained for different values of n with the determined grammar

As it can be seen in the figure 9, the proposed method allows us to obtain classes of generalized images, starting from an initial pattern, images that satisfy the same rules.

The ability of understanding an image will be increased if we know the rules under which they were generated (if the images had been defined by relations between components, and not by static or semantic rules). In such cases, the importance of lingvistical methods is obvious, since these methods represent tools useful in solving recognition problems.

The final step in pattern recognition process is the parsing (syntactical analysis), in which the forms are classified according to the description grammars of the form classes. The parsing decides if a certain form belongs or not to a form class, and when the answer is afirmative it will provide information about the form structure.

In our case, a form is a drawing, and a form class is a picture language. The problem of recognising a drawing d (representing the studied form) from a picture language P (a given form class) is: "$d \in P$?". Consequently, we have presented here a method of solving this problem.

REFERENCES

[1] K.S. Fu, *Syntactic Pattern Recognition, Application*, Springer-Verlag, New York, 1977.
[2] E.B. Hunt, *Artificial Intelligence, Pattern Recognition, Grammatical Pattern Classification*, Academic Press, New York, San Francisco, London, 1975, pg. 144-182.
[3] A.K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, London, 1989.
[4] A.C. Shaw, *A formal picture description scheme as a basis for picture processing system*, Inform. and Control, 14, 9, 1969.
[5] A. Vlaicu, *Prelucrarea digitală a imaginilor*, Editura Albastră, Cluj-Napoca, 1997.
[6] T. Pavlidis, *Algorithms for Graphics and Image Processing*, Springer-Verlag (Berlin, Heildelberg), 1982.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania
E-mail address: per|motogna@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 1, 2001

PURELY FUNCTIONAL PROGRAMMING AND THE OBJECT-ORIENTED INHERITANCE AND POLYMORPHISM

LEHEL KOVÁCS, GÁBOR LÉGRÁDI, AND ZOLTÁN CSÖRNYEI

Abstract. According to the purely functional paradigm, the value of an expression depends only on the values of its subexpressions, if any.

In this paper we introduce this principle in the object-oriented paradigm. The simplicity and power of functional languages is due to properties like pure values, first-class functions, and implicit storage management. We must extend these properties with a strong type-system.

The values must be typed, the type system used for this purpose is the higher-order, explicitly-typed, polymorphic λ-calculus with subtyping, called $F^{\omega}_{\leq:}$.

This type-system must be prepared for basic mechanisms of object-oriented programming: encapsulation, message passing, subtyping and inheritance. Polymorphic functions arise naturally when lists are manipulated and lists with elements of any types can be accomplished by a straightforward generalization of inheritance.

Interesting questions are also, how to introduce the object- oriented inheritance, the subtyping mechanism and the object oriented polymorphism.
Key Words and Phrases: untyped and typed λ-calculus, object-oriented programming, inheritance, polymorphism.

1. Introduction

In this paper we introduce the principle of purely functional paradigm into the object-oriented paradigm and we concentrate the solutions of problems related to each kind of inheritance and polymorphism.

We are going to focus our attention on a simple object model in which an object has a value that can be modified by messages.

An object has an internal state and methods, the states have unusual feature: the internal state of all objects is immutable, the result of methods are values rather side-effects of variables.

The simplicity and power of functional languages are due to properties like pure values, first-class functions, and implicit storage management. We must extend

2000 *Mathematics Subject Classification.* 68P05.

1998 *CR Categories and Descriptors.* D.1.1 [**Software**]: Programming Techniques – *Applicative (Functional) Programming*; D.1.5 [**Software**]: Programming Techniques – *Object-oriented Programming*.

these properties with a strong type system, the type system $F^{\omega}_{\leq}$ is applied. A summary of this type system is given in the next section.

There are three object models from literature, the record model, the existential-type model and the axiomatic model [2]. We use the the record model to refer to the representation of objects by recursively defined records in which a class is represented by a record of variables and methods. Using λ-calculus and type system $F^{\omega}_{\leq}$ extended by record structures we have a purely functional model of object-oriented programming.

We also show how to write well-typed polymorphic functions that operates on different objects. A polymorphic function can be applied to arguments of more than one type. We concentrate on parametric polymorphism, a special kind of polymorphism, in which type expressions are parametrized.

2. Type system $F^{\omega}_{\leq}$

Church's ordinary typed λ-calculus has the name F_1, F_2 correspond to Girard's and Reynold's second-order typed λ-calculus. F_3 is obtained from F_2 by allowing type constructors that transform existing types into new types, and it follows that the *kind* has the form $*$ or $* \rightarrow kind$. Using successively higher kind, we obtain systems F_4, F_5,... The union of all these systems is called F^{ω}. In system F^{ω} the syntax of *kinds* has the form

$$\begin{array}{lcl} kind & := & * \\ & | & kind \rightarrow kind \end{array}$$

One of the natural extension of type system F^{ω} deals with subtyping, a simple version of such a system is $F^{\omega}_{\leq}$. The description of this system is described for example in [3].

A further extension of $F^{\omega}_{\leq}$ introduces *records* [5], it is mainly used to formulate models for object-oriented systems [1,4].

The syntax of record type, record term construction and record term selection are as follows:

$$\begin{array}{lcl} < type > & := & \{| < name_1 >:< type_1 >, \ldots, < name_n >:< type_n > |\} \\ < term > & := & \{< name_1 >=< term_1 >, \ldots, < name_n >=< term_n >\} \\ & | & < term > . < name > \end{array}$$

Now we will extend the rule-system $F^{\omega}_{\leq}$ to allow records.

(1) The *kinding* rule for record:

$$\Gamma \ \vdash \ < type_i > \in * \ \text{ for each } i$$
$$\Rightarrow \Gamma \ \vdash \ \{| < name_1 >:< type_1 >, \ldots, < name_n >:< type_n > |\} \in *$$

(2) The rule of record *introduction*:

$$\Gamma \vdash < term_i >:< type_i > \quad \text{for each } i$$
$$\Rightarrow \Gamma \vdash \{< name_1 >=< term_1 >, \ldots, < name_n >=< term_n >\}$$
$$: \{| < name_1 >:< type_1 >, \ldots, < name_n >:< type_n > |\}$$

(3) The rule of record *elimination* for the record $< term >\equiv \{< name_1 >=< term_1 >, \ldots, < name_n >=< term_n >\}$:

$$\Gamma \vdash < term >: \{| < name_1 >:< type_1 >, \ldots, < name_n >:< type_n > |\}$$
$$\Rightarrow < term > . < name_i >:< type_i >$$

(4) And finally, the *subtype* rule has two assumptions,
- the subtype record has at least the same fields as the other record type,
- each of the types of the fields of the subtype need to be subtypes of the types of the corresponding fields (if they exist) in the other type.

$$\{name_{1,1}, \ldots, name_{1,m}\} \supseteq \{name_{2,1}, \ldots, name_{2,n}\},\ m \geq n$$
$$\Gamma \vdash < type_{1,i} > \leq < type_{2,i} > \quad \text{for each } < name_{1,i} >=< name_{2,i} >$$
$$\Rightarrow \Gamma \vdash \{| < name_{1,1} >:< type_{1,1} >, \ldots, < name_{1,m} >:< type_{1,m} > |\}$$
$$\leq \{| < name_{2,1} >:< type_{2,1} >, \ldots, < name_{2,n} >:< type_{2,n} > |\}$$

3. The object-oriented inheritance

A class (child class) inherits state and behavior from its superclass (parent class). Inheritance provides a powerful and natural mechanism for organizing and structuring software programs. The instance variables and the methods of the child class are an extension of the structure and the behavior of the parent class. The child class extends the properties, the structure of the parent class, and constitutes a limitation of the meaning.

Definition 3.1 (The principle of substitutability)**.** *An instance of the child class can imitate the behavior of a parent class and can not be distinguished from an instance of the parent class if it is used in a similar situation. If C is the child oclass and P is the parent class, $C = subst(P)$ means that each instance of C may be used instead of an instance of P.*

Definition 3.2 (Subtype)**.** *A subtype is a class which fulfills the principle of substitutability ($C = subst(P)$).*

Informally, a type σ is a subtype of τ, written $\sigma \leq \tau$, if an expression of type σ can be used in any context that expects an expression of type τ.

The rule for subtyping functions states that $\sigma \to \tau \leq \sigma' \to \tau'$ iff $\sigma' \leq \sigma$ and $\tau \leq \tau'$.

This is formalized by extending our λ-calculus with a subtype relation, written $\Gamma \vdash S \leq T$ to mean that S is a subtype of T under assumptions Γ.

Definition 3.3 (Subclass)**.** *A subclass is an arbitrary class created by inheritance, regardless of the principle of substitutability ($C \neq subst(P)$).*

A subclass may also override the definitions of methods it would otherwise inherit by redefining them. Because a subclass inherits code for methods, it also inherits interface type information for the methods that it does not override.

We write $\Gamma \vdash S < T$ to mean that S is not a subtype of T, but is a subclass of T under assumptions Γ.

In the object-oriented paradigm a class is a prototype that defines the variables and the methods common to all objects of a certain kind.

Let $v_1, v_2, \ldots, v_i$ be the variables and let $m_1, m_2, \ldots, m_j$ be the methods, let $V = \{V_1, V_2, \ldots, V_i\}$ be the typeclass of each variable and let $M = \{M_1, M_2, \ldots, M_j\}$ be the typeclass of each method (the signature of methods) for a given class S, where i is the number of variables and j is the number of methods for a class $S = V \cup M$.

In that case, the subtyping rule between classes S and T is:

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$

$$V' = \{V'_1, V'_2, \ldots, V'_{i'}\}, \quad M' = \{M'_1, M'_2, \ldots, M'_{j'}\},$$

$$V \subseteq V', \ M \subseteq M', \ i \geq i', \ j \geq j',$$

$$S = V \cup M, \ T = V' \cup M',$$

$$T = \mathrm{subst}(S),$$

$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$

$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_{i'} : V'_{i'}, m'_1 : M'_1, \ldots, m'_{j'} : M'_{j'}\ |\} \in *$$

$$\Rightarrow \Gamma \vdash S \leq T$$

For a better formalization of $T = \mathrm{subst}(S)$ or $T \neq \mathrm{subst}(S)$ we must give the subtyping (or subclassing) rule for each kind of inheritance.

3.1. **Specialization.**

Definition 3.4 (Specialization)**.** *The child class is a specialized form of the parent class.*

Additional functionalities, principle of substitutability is guaranteed. Does not change the old variables and methods (from parent class), but brings in new variables and new methods in child class.

$$V = \{V_1', V_2', \ldots, V_{i'}', \ldots, V_i\}, \quad M = \{M_1', M_2', \ldots, M_{j'}', \ldots, M_j\},$$

$$V' = \{V_1', V_2', \ldots, V_{i'}'\}, \quad M' = \{M_1', M_2', \ldots, M_{j'}'\},$$

$$V \subseteq V',\ M \subseteq M',\ i > i',\ j > j',$$

$$S = V \cup M,\ T = V' \cup M',$$

$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$

$$\Gamma \vdash \{|\ v_1' : V_1', \ldots, v_{i'}' : V_{i'}', m_1' : M_1', \ldots, m_{j'}' : M_{j'}'\ |\} \in *$$

$$\Rightarrow \Gamma \vdash S \leq T$$

3.2. **Specification.**

Definition 3.5 (Specification)**.** *The parent class defines an interface, the child class gives the implementation.*

Changes the old variables of parent class, no additional functionalities, principle of substitutability is guaranted.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1', M_2', \ldots, M_j'\},$$

$$V' = \{V_1', V_2', \ldots, V_i'\}, \quad M' = \{M_1', M_2', \ldots, M_j'\},$$

$$V \subseteq V',\ M = M',$$

$$S = V \cup M,\ T = V' \cup M',$$

$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1', \ldots, m_j : M_j'\ |\} \in *,$$

$$\Gamma \vdash \{|\ v_1' : V_1', \ldots, v_i' : V_i', m_1' : M_1', \ldots, m_j' : M_j'\ |\} \in *$$

$$\Rightarrow \Gamma \vdash S \leq T$$

3.3. **Construction.**

Definition 3.6 (Construction)**.** *The parent class provides the functionality, but gives no logical context to the child class.*

Typical kind of subclassing, changes the old variables, methodes, adds new variables and methods, no substitutability.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V_1', V_2', \ldots, V_{i'}'\}, \quad M' = \{M_1', M_2', \ldots, M_{j'}'\},$$
$$V \subseteq V',\ M \subseteq M',\ i \geq i',\ j \geq j',$$
$$S = V \cup M,\ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v_1' : V_1', \ldots, v_{i'}' : V_{i'}', m_1' : M_1', \ldots, m_{j'}' : M_{j'}'\ |\} \in *$$
$$\Rightarrow \Gamma \vdash S < T$$

3.4. **Generalization.**

Definition 3.7 (Generalization)**.** *The child class extends the functionality of the parent class, creates more general instances.*

Like the construction, but we can tell absolutly nothing about substitutability. It depends on particular cases.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V_1', V_2', \ldots, V_{i'}'\}, \quad M' = \{M_1', M_2', \ldots, M_{j'}'\},$$
$$V \subseteq V',\ M \subseteq M',\ i \geq i',\ j \geq j',$$
$$S = V \cup M,\ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v_1' : V_1', \ldots, v_{i'}' : V_{i'}', m_1' : M_1', \ldots, m_{j'}' : M_{j'}'\ |\} \in *$$
$$\Rightarrow \Gamma \vdash S \leq ? < T$$

3.5. **Extension.**

Definition 3.8 (Extension)**.** *The child class adds further functionalities to the parent class, but does not change any inherited behavior.*

The principle of substitutability is guaranted.

$$V = \{V_1', V_2', \ldots, V_i'\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V_1', V_2', \ldots, V_i'\}, \quad M' = \{M_1', M_2', \ldots, M_j'\},$$
$$V = V',\ M \subseteq M',$$
$$S = V \cup M,\ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1', \ldots, v_i : V_i', m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v_1' : V_1', \ldots, v_i' : V_{i'}', m_1' : M_1', \ldots, m_{j'}' : M_j'\ |\} \in *$$
$$\Rightarrow \Gamma \vdash S \leq T$$

3.6. **Limitation.**

Definition 3.9 (Limitation)**.** *The child class restricts the use of some of the behaviors inherited from the parent class.*

The principle of substitutability is not guaranteed.

$$V = \{V'_1, V'_2, \ldots, V'_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V'_1, V'_2, \ldots, V'_i\}, \quad M' = \{M'_1, M'_2, \ldots, M'_j\},$$
$$V = V', \ M \subseteq M',$$
$$S = V \cup M, \ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V'_1, \ldots, v_i : V'_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_i : V'_i, m'_1 : M'_1, \ldots, m'_j : M'_j\ |\} \in *$$
$$\Rightarrow \Gamma \vdash S < T$$

3.7. **Variance.**

Definition 3.10 (Variance)**.** *The parent and the child class are variant of each other, the class – subclass relationship is arbitrary.*

The principle of substitutability is not guaranteed.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V'_1, V'_2, \ldots, V'_i\}, \quad M' = \{M'_1, M'_2, \ldots, M'_j\},$$
$$V \subseteq V', \ M \subseteq M',$$
$$S = V \cup M, \ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_i : V'_i, m'_1 : M'_1, \ldots, m'_j : M'_j\ |\} \in *$$
$$\Rightarrow \Gamma \vdash S < T$$

3.8. **Combination.**

Definition 3.11 (Combination)**.** *The child class inherits features from two or more parent classes. It is commonly called multiple inheritance.*

Changes the old variables, methods, brings in new variables, methods. We can tell absolutly nothing about substitutability. It depends on particular cases.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V'_1, V'_2, \ldots, V'_{i'}\}, \quad M' = \{M'_1, M'_2, \ldots, M'_{j'}\},$$
$$V \subseteq V', \ M \subseteq M', \ i \geq i', \ j \geq j',$$
$$S = V \cup M, \ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_{i'} : V'_{i'}, m'_1 : M'_1, \ldots, m'_{j'} : M'_{j'}\ |\} \in *$$
$$\Rightarrow \Gamma \vdash S \leq ? < T$$

4. Summary - Inheritance

The following table contains a general view of object-oriented inheritance kinds:

Kind	*New variables*	*New methods*	*Substi-tutability*	*Changes old variables*	*Changes old methods*
specialization	Yes	Yes	Yes	No	No
specification	No	No	Yes	Yes	No
construction	Yes	Yes	No	Yes	Yes
generalization	Yes	Yes	?(Yes,No)	Yes	Yes
extension	No	Yes	Yes	No	No
limitation	No	No	No	No	Yes
variance	No	No	No	Yes	Yes
combination	Yes	Yes	?(Yes,No)	Yes	Yes

5. The object-oriented polymorphism

According to the object-oriented polymorphism, one message (the same message) can be different interpreted by the objects (Methods with the same names, signatures and with different bodies). Polymorphism is a natural characteristic of object-oriented languages based on the principles of message passing, inheritance and substitutability [6].

For a better formalization we must give the subtyping rule for each kind of polymorphism.

5.1. **Overloading.**

Definition 5.1 (Overloading)**.** *A function name denotes more than one possible statement sequence.*

Overloading extends the syntax of the programming language. For example, overloading the '+' operator for *Complex* numbers (*Complex* is a class). Subtyping rule: see Overriding.

5.2. **Polymorphic parameters.**

Definition 5.2 (Polymorphic parameters)**.** *One method (function or procedure) can be called with different type of arguments.*

A method can be redeclared with a different parameter signature from its ancestor, it overloads the inherited method without hiding it. Calling the method in a descendant class activates whichever implementation matches the parameters in the call.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$

$$V' = \{V'_1, V'_2, \ldots, V'_{i'}\}, \quad M' = \{M'_1, M'_2, \ldots, M'_{j'}\},$$
$$V \subseteq V',\ M \subseteq M',\ i \geq i',\ j \geq j',$$
$$S = V \cup M,\ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_{i'} : V'_{i'}, m'_1 : M'_1, \ldots, m'_{j'} : M'_{j'}\ |\} \in *$$
$$\exists k \in \{1, \ldots, j\} : name(m_k) = name(m'_k) \wedge signature(m_k) \neq signature(m'_k)$$
$$\Rightarrow \Gamma \vdash S \leq T$$

Where *name(m)* is the name of the method, and *signature(m)* is the signature (name and parameter list) of the method.

5.3. **Deferring.**

Definition 5.3 (Deffering)**.** *The parent class declares only the method, the child class implements it.*

Commonly called abstract polymorphism.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$

- no method bodies in M, just the declarations.

$$V' = \{V'_1, V'_2, \ldots, V'_{i'}\}, \quad M' = \{M'_1, M'_2, \ldots, M'_{j'}\},$$
$$V \subseteq V',\ M \subseteq M',\ i \geq i',\ j \geq j',$$
$$S = V \cup M,\ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_{i'} : V'_{i'}, m'_1 : M'_1, \ldots, m'_{j'} : M'_{j'}\ |\} \in *$$
$$\forall k \in \{1, \ldots, j\} : signature(m_k) = signature(m'_k)$$
$$\Rightarrow \Gamma \vdash S \leq T$$

5.4. **Overriding.**

Definition 5.4 (Overriding)**.** *A child class changes the meaning of a method, which was defined in the parent class.*

Overriding a method means extending or refining it, rather than replacing it. A descendant class can override any of its inherited virtual methods. It is the common and the most used case of polymorphism.

$$V = \{V_1, V_2, \ldots, V_i\}, \quad M = \{M_1, M_2, \ldots, M_j\},$$
$$V' = \{V'_1, V'_2, \ldots, V'_{i'}\}, \quad M' = \{M'_1, M'_2, \ldots, M'_{j'}\},$$
$$V \subseteq V',\ M \subseteq M',\ i \geq i',\ j \geq j',$$
$$S = V \cup M,\ T = V' \cup M',$$
$$\Gamma \vdash \{|\ v_1 : V_1, \ldots, v_i : V_i, m_1 : M_1, \ldots, m_j : M_j\ |\} \in *,$$
$$\Gamma \vdash \{|\ v'_1 : V'_1, \ldots, v'_{i'} : V'_{i'}, m'_1 : M'_1, \ldots, m'_{j'} : M'_{j'}\ |\} \in *$$

$$\exists k \in \{1, \ldots, j\} : signature(m_k) = signature(m'_k)$$
$$\Rightarrow \Gamma \vdash S \leq T$$

6. Summary - Polymorphism

The following table contains a general view of object-oriented polymorphism kinds:

Kind	*Same names*	*Same bodies*	*Same signatures*	*Abstract methods*	*Extends the syntax*
overloading	Yes	No	Yes	No	Yes
parameters	Yes	No	No	No	No
deferring	Yes	No	Yes	Yes	No
overriding	Yes	No	Yes	No	No

References

[1] Atsuchi Igarashi, Benjamin C. Pierce, *Foundations for Virtual Types*, ECOOP'99, LNCS 1628, 1999, 161–185.
[2] Martin Abadi, Luca Cardelli, *A Theory of Objects*, Springer-Verlag, 1996.
[3] Luca Cardelli, *Notes about* $F^{\omega}_{\leq}$, http://citeseer.nj.nec.com/cs, Unpublished manuscript, October 1990
[4] Kathleen Fisher, John C. Mitchell, *The Development of Type Systems for Object-Oriented Languages*, Stanford University, STAN-CS-TN-96-30
[5] Benjamin C. Pierce, *Type Systems*, Draft, 2000.
[6] Luca Cardelli, Peter Wagner, *On Understanding Types, Data Abstraction and Polymorphism*, ACM Computing Surveys, 17 (4), 1995, 471–522

Department of Computer Systems, Babeş-Bolyai University, Cluj-Napoca
E-mail address: klehel@cs.ubbcluj.ro

John von Neumann Faculty of Informatics, Budapest Polytechnic, Budapest
E-mail address: legradi@nik.bmf.hu

Department of General Computer Science, Eötvös Loránd University, Budapest
E-mail address: csz@inf.elte.hu