Preface

It is customary in speaking of a scientific conference to designated by an acronym of its full name. It is a measure of the standing of such an event how well the general public understands the meaning of the acronym. Acronyms like SOFSEM, MFCS, ICALP are very well known in the IT community. ITAT is not so lucky - it is an acronym of a new workshop and it is because of this that we start by providing some basic information. In addition to expanding the acronym to the full name Information Technologies - Applications and Theory it seems appropriate to provide more details. This is best done - especialy when strating a new tradition - by answering the five classic questions: WHY, WHO, HOW, WHERE and WHAT?

WHY did we organize ITAT 2001?
More precisely - what did we expect the new workshop to look like? To be quite brief, we wanted to create an opportunity for researchers from the Slovakia, Czech Republic and Poland to meet in a nice place where they would be free from ordinary distractions and could discuss their work on - hopefully - the leading edge of computer science. This should promote closer contacts and cooperation in future.

WHO was invited to participate in the workshop?
Instead of precise formal selection criteria the organizers invited people with whom and whose work they were already acquainted. One might say that participants of this initial workshop were selected with a view to forming the program committee for future years.

HOW was the whole event organized?
Since it was people rather then papers who were invited to the conference there was no formal refereeing mechanism. The program itself consisted of lectures and short contributions where the length was suited to the subject. The presentations took up mornings and evenings while the middle part of each day was devoted to getting acquainted with the beautiful surrounding mountains. The selection of papers for the present proceedings volume was made by mutual agreement during the conference. One drawback of this mechanism was that it left no room for language editing; the language is therefore fully the responsibility of the authors.

WHERE did the workshop take place?

The venue of the workshop was the village Zuberec in the foothills of the Rohace mountain range. The place was selected with a view to opportunities for rest and recreation and also because the weather there is usually nice in September. We hope that the participants will agree that it was a suitable choice for this as well as the coming workshops.

WHAT were the subjects dealt with at the workshop?

It is not suprising - considering the extreme variety of subjects addressed by theoretical computer science - that the subjects of presented papers fall into quite a lot of subject categories.

In formal languages and automata there were two papers: M. Platek et al. discussed the relaxations and restrictions of word order in dependency grammars and D. Prusa devoted his paper to a generalization of context-free grammars suitable for defining picture languages.

Two papers were devoted to what is now generally known as data mining: I. Mrázová and F. Mráz surveyed data mining techniques based on neural networks while R. Lencses described (in the only paper written in Slovak) various techniques used in information retrieval.

One paper only by J. Hric was devoted to logic programming, namely to introducing monads into logic programming and Prolog. In contrast, three papers dealt with subjects from the more general surroundings of information technology: T. Pitner discussed the impact of information technologies on sustainable development, J. Kohoutková described the use of directed graphs for presenting objects and their relationships while Z. Fabián devoted his paper to relationships between probability measure and metric in the sample space. It should be mentioned that in addition to the published papers several lectures were delivered: P. Vojtáš presented some results concerning vague and ambiguous data, G. Andrejková enlarged on her favourite subject of approximation of functions by neural networks and S. Krajči delivered a lecture on the XML language.

On the whole we feel that this "number zero" workshop was a success and hope that it will lead to a whole series of annual ITAT workshops in the coming years. We also hope that the workshop will then be well known to such an extent that long forewords like the present one will become superfluous.

Košice 2001 G. Andrejková, J. Vinař

Content

Word-Order Relaxations and Restrictions within a Dependency Grammar[1]

Martin Plátek, Tomáš Holan, Vladislav Kuboň

Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic,

platek@ksi.ms.mff.cuni.cz, holan@ksvi.ms.mff.cuni.cz, vk@ufal.mff.cuni.cz

Karel Oliva

Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010, Wien, Austria

karel@ai.univie.ac.at

Abstract

This paper studies (formally) certain combinations of relaxations and restrictions of word-order, using a straightforward and intuitive formalization of dependency grammars (DG's). In particular, we first relax the relation between dependency and word order, and then study different possibilites of relaxing and restricting this relation. We introduce a new concept of *proper DG's* based on the notion of projectivity. By defining various types of word order (projectivity) relaxations, and by introducing restrictions on the relaxation within each such type, we obtain (two) infinite scales of classes of languages. Finally, we discuss the parsing-complexity of the individual classes (of languages) of the presented scales.

Valui poenam fortis in ipse meam
I-was disadvantage brave to I my
"I was brave, to my disadvantage"
Ovidius, Ars amandi 1.7.26

[1]Supported by the Grant Agency of the Czech Republic, Grant-No. GA ČR 201/02/1456

1 Introduction

We have chosen the above motto for this paper not so much because of its content, but rather in order to remind the reader that in languages typologically different from English, high degree of word order freedom, giving rise to heavy nonprojectivity (discontinuous constituency, non-configurationality), can still result in an acceptable sentence. The idea that word order freedom is not at all marginal in many languages, and hence that, in spite of the current anglocentrism, it should be taken seriously also in computational linguistics, constitutes the starting point of the research described in this paper.

Thus, even though this paper may seem to be mainly technically oriented, it is definitely motivated by linguistic considerations. It focuses on the phenomenon of word order freedom and on the implications it has on parsing. The main topic of the paper is the endeavor to provide an adequate formal description of a special class of dependency grammars called *proper dependency grammars*. This class exhibits properties which may be used to a considerable advantage in the process of parsing natural languages allowing a high number of nonprojective constructions inside one sentence.

For this purpose, we define a formal tool allowing for separating the description of syntactic dependency from the description of word order. Based on this, we introduce and study such notions as *degrees of relaxation* and *degrees of restriction* of word order, and correlate them with the notions and results in the mainstream of theory of formal languages and parsing.

Our investigation of the role of nonconfigurationality of natural languages does not start with this paper. In [9] we have introduced the *Free-Order Dependency Grammars (FODG's)* as a formal system suitable for dependency-based parsing of natural languages, in particular of those displaying a high degree of free word order. The design of this formal system was based on the experience acquired during our work on the development of a grammar-checker for Czech, cf. [8].

Hence, the current paper constitutes a possible next step towards a formal basis of a complete dependency-based syntactic analysis of Czech which was pioneered by the framework of *Functional Generative Description* (*FGD*, cf. [19]). In comparison with *FGD* and other common types of formal systems describing the syntax of natural languages, e.g. Tree-Adjoining Grammars (cf. [12]), the crucial novelty brought in by *FODG's* is that they take the phenomenon of word order freedom more seriously. In particular, in [9] we have introduced two types of measures of word-order freedom based on *FODG's*. The study of one of them is further developed in [18]. In [18] and in the current article, particular stress is given on the phenomenon of word order relaxations and restrictions. Word order relaxation in a related (but much more limited) sense was studied also in [17]. The other (above mentioned) type of measure was used to compare the complexity of word order of Czech and English in [10]. In [18] we have deliberately pointed out the fact that the (general, ordinary) formalizations of DG's cover also certain very 'nonliguistic' classes of DG's. In fact, we used some quite 'nonliguistic' examples to show the technical results there. Here we introduce the class of *proper* DG's, which (in our opinion) creates a better approximation of 'linguistically motivated' DG's. With this class of grammars we achieve similar results as in [18]. It means that also in this paper the concept of the (degrees of) relax-ability of word order remains the same as in [18], and keeps its original meaning.

In particular, the notion of word order relaxation within a DG means that besides the usual (projective) interpretations of one type of (ordinary) dependency grammars, cf. [2], other (relaxed) types of interpretations are also considered. The *scattered context grammars* ([16]) are related to the ordinary context-sensitive grammars in a similar way as relaxed DG's are related to ordinary (projective) dependency grammars. However, scattered context grammars do not directly retain the notion of a derivation (or syntactic) tree.

Two types of syntactic structures, namely *DR-trees* (delete-rewrite trees), and *D-trees* (dependency trees, deleting trees), are used in this paper. We consider DR-trees rather for a (special) drawing (visualization) of trees than for trees from the (pure) graph theory. Advantages of this approach were significantly used in [11] and [13]. A notion similar to DR-trees has been used already in [4] and [3] in order to show basic equivalencies between context-free and dependency grammars. Any DR-tree can be transformed into a D-tree in an easy and uniform way. DR-trees are used in this paper as a rather technical notion. On the other hand the (sets of) D-trees serve usually as a formal representation of dependency (syntactic) analysis (see e.g. [14], [17], [9]). The formulation of the results of this paper is based on the notion of a gap in DR-trees. The notion of proper grammar is based on the notion of a gap in DR-trees and on the notion of a gap in D-trees as well.

2 Basic notions

The basic notion we work with is dependency grammar (DG'). In the sequel the DG's are analytic (recognition) grammars, and we will use different interpretations of the rules of DG's, as usual. Even the next definition is more general then the corresponding one from the previous section. It allows to use the rules with terminals on their left-hand side. This property has some advantages for grammar-checking (cf. [8]) and it is close to the original linguistic idea of dependencies.

Definition 2.1 A dependency grammar (DG) is a tuple $G = (T, N, S_t, P)$, where the union of N and T is denoted as V, T is the set of terminals, N is the set of nonterminals, $S_t \subseteq V$ is the set of root-symbols (starting symbols), and P is the set of rewriting rules of the following forms:

a) $A \rightarrow_X BC$, where $A \in V$, $B, C \in V$, $X \in \{L, R\}$.

b) $A \rightarrow B$, where $A \in V$, $B \in V$.

The letter L (R) in the subscripts of the rules of the type a) means that the first (second) symbol on the right-hand side of the rule is considered *dominant*, and the other *dependent*.

If a rule has only one symbol on its right-hand side, we consider this symbol to be *dominant*.

Informally, a rule is applied (for a reduction) in the following way: The dependent symbol is deleted (if there is one on the right-hand side of the rule), and the dominant one is rewritten (replaced) by the symbol standing on the left-hand side of the rule. The rules $A \rightarrow_L BC$, $A \rightarrow_R BC$ can be applied for a reduction of a string z for any of the occurrences of symbols B, C in z, where B precedes C in z, in the general case not necessarily immediately.

The basic technical notion of this paper is the notion of a *DR-tree* (delete-rewrite-tree) according to G. This notion allows to introduce several different interpretations of a dependency grammar, i.e. different types of languages and different types of dependency analyses. A *DR-tree* maps the essential part of history of deleting dependent symbols and rewriting dominant symbols, performed by the rules applied.

Put informally, we consider a *DR-tree* (according to a DG G) to be a rooted tree on the one hand and to be at the same time a drawing in the plain of the tree on the other hand, cf. Fig.1a. A DR-tree here is in fact a drawing of a finite, binary tree with a root and with the following types of edges:

a) *vertical* (V-edges): these edges represent the rewriting of the dominant symbol by the symbol which is on the left-hand side of the rule (of G) used. The vertical edge leads (is oriented) bottom-up from the node containing the original dominant symbol to the node containing the symbol from the left-hand side of the rule used.

b) *oblique*: these edges represent the deletion of a dependent symbol. Any such edge is oriented from the node with the dependent deleted symbol to the node containing the symbol from the left-hand side of the rule used. There are two types of oblique edges:
i) directed bottom up and from the left to the right (R-edges),
ii) directed bottom up and from the right to the left (L-edges).

Let us now proceed more formally. The following technical definition of DR-tree allows to describe a DR-tree using its set of nodes only, allows to define a corresponding dependency tree from a DR-tree, allows to formulate several (practical) restrictions of a topological (or even geometrical) nature. Some of them are studied in this paper, some others are proposed and used in [11], [13].

Definition 2.2 A tree $Tr = (Nod, Ed, Rt)$ is called *DR-tree* induced by a DG $G = (T, N, S_t, P)$ (where Nod means the set of nodes, Ed the set of edges, and Rt means the root node), if the following points hold for any $U \in Nod$:

a) To any node of Tr lead at most two edges (if at least one edge leads to a node then exactly one of them is a vertical edge).

b) U is a 4-tuple of the form $[A, i, j, e]$, where $A \in V$ (terminal or nonterminal of G), $i, j \in Nat$, e is either equal to 0 or it has the shape (k, p), where $k, p \in Nat$. The A is called the *symbol of* U, the number i is called the *horizontal index of* U, j is called the *vertical index*, e is called the *domination index*. The horizontal index expresses the correspondence of U with the i-th input symbol. The vertical index corresponds to the length of the longest path leading (bottom-up) from a leaf to U increased by 1. The domination index either represents the fact that no edge starts in U ($e = 0$) or it represents the final node of the edge starting in U ($e = (k, p)$, cf. also the point f) below). If two nodes of the shape $[A, i, j, e]$, $[B, k, l, f]$ occur in Tr, then $i \neq k$ or $j \neq l$ (any node has its own position in the plain).

c) Let $U = [A, i, j, e]$ and $j > 1$. Then there exists exactly one node U_1 of the form $[B, i, k, (i, j)]$ in Tr, such that $1 \leq k < j$, the pair (U_1, U) creates a V-edge of Tr, and there is a rule in G with A on its left-hand side, and with B in the role of the dominant symbol of its right-hand side.

d) Let $U = [A, i, j, e]$. Then U is a leaf if and only if $A \in T$ (terminal symbol of G), and $j = 1$.

e) Let $U = [A, i, j, e]$. $U = Rt$ iff it is the single node with the domination index (e) equal to 0.

f) Let $U = [A, i, j, e]$. If $e = (k, p)$ and $k < i$ (or $k > i$), then an L-edge (R-edge) leads from U (dependent node) to its mother node U_m with the horizontal index k and vertical index p. Further a vertical edge leads from some node U_s to U_m. Let C be the symbol from U_m, B the symbol from U_s, then P contains a rule $C \rightarrow_L BA$ (or $C \rightarrow_R AB$).

g) Let $U = [A, i, v, e]$. If $e = (k, p)$, and $k = i$, then a vertical edge leads (bottom up) from U to its mother node $U_m = [B, i, p, e_m]$ (for some B and e_m), and P contains a rule with symbol B on its left-hand side and with symbol A in the role of the dominant symbol of the right-hand side, i.e., if U_m has only a single daughter, then there exists a rule in P of the shape $B \rightarrow A$, if U_m has two daughters etc.

Let us denote the set of DR-trees induced by G as $CrT(G)$.

We will say that a *DR-tree* Tr is *complete* if for any of its leaves $U = [A, i, 1, e]$, where $i > 1$, it holds that there exists (exactly) one leaf with the horizontal index $i - 1$ in Tr.

Remark. The notion of DR-tree corresponds to the fact that we consider dependency grammars for (bottom-up) analytic grammars, cf. the points c) and d) in the previous definition. It is not hard to transform at this point the dependency grammars (DR-trees) more close to the generative principles, but we are afraid that in that case our following consideration would become less transparent. Also the linguistic notion of dependency is derived from the principle of reduction of sentences.

Definition 2.3 Let $G = (T, N, S_t, P)$ be a DG. $FT(G)$ denotes the set of all complete DR-trees rooted in a symbol from S_t, generated by G. If $Tr \in FT(G)$, we say that Tr is (freely) *parsed* by G.

We can see that $FT(G) \subseteq CrT(G)$ for any DG G.

Let $w = a_1 a_2 \ldots a_n$, $w \in T^*$, $Tr \in FT(G)$, Tr has exactly n leaves, and let the i-th leaf of Tr have the form $[a_i, i, 1, e_i]$ for $i = 1, \ldots, n$. In such a case we say that the string w *is parsed into* Tr by G.

We shall write $FT(w, G) = \{Tr; w \textit{ is parsed into } Tr \textit{ by } G\}$.

Definition 2.4 Let $Tr \in FT(w, G)$ (w is parsed into Tr by G), where $w = a_1 a_2 ... a_n$. The *contraction* of Tr into a *dependency tree* $dT(Tr) =$

$(dNode, dE, dR)$ is defined as follows: The set of nodes $dNode$ is the set of 3-tuples $[a_i, i, k_i]$ (note that a_i is the i-th symbol of w). We call a_i the *symbol* of the node, i the *horizontal index* of the node, and k_i the *domination index* of the node. $k_i = 0$ if and only if the root of Tr has the horizontal index i. $k_i \in Nat$ if and only if an oblique edge of Tr leads from some node with the horizontal index i to some node with the horizontal index k_i. The edges of dE correspond (one to one) to the oblique edges of Tr and they are fully represented by the second and the third slots of nodes of $dT(Tr)$.

Let us denote $dFT(w, G) = \{dT(T)|T \in FT(w, G)\}$, $dFT(G) = \{dT(T)|T \in FT(G)\}$.

If $dT \in dFT(G)$ we say that dT is parsed by G.

We say that $dFT(w, G)$ is the dependency analysis of w by G, and $dFT(G)$ is the dependency analysis by G.

Example 2.5 This example illustrates the notion of DG. We define the following grammar G_2 (we denote this grammar by G_2 because we use a sequence of grammars G_i in the next section and G_2 will be the second member of this sequence). $G_2 = (T_2, N_2, \{A_1\}, P_2)$, $T_2 = \{a_1, a_2, b_1, b_2\}$, $N_2 = \{A_1, A_2, B_1, B_2\}$, $P_2 = \{A_1 \rightarrow_R a_1B_1, B_1 \rightarrow_L b_1A_2, A_2 \rightarrow_R a_2B_2, B_2 \rightarrow_L b_2A_1|b_2\}$.

The left part of Fig.1. displays a DR-tree $Tr_1 \in FT(a_1a_1a_2a_2b_1b_2b_1b_2, G_2)$, i.e. a DR-tree parsed by G_2 for the input sentence $a_1a_1a_2a_2b_1b_2b_1b_2$.

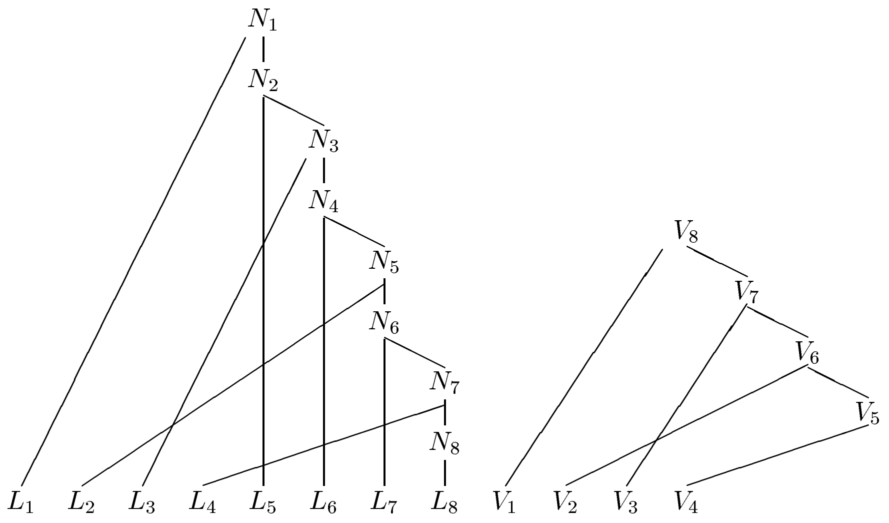
The leaves of Tr_1 are $L_1 = [a_1, 1, 1, (5, 9)]$, $L_2 = [a_1, 2, 1, (7, 5)]$, $L_3 = [a_2, 3, 1, (6, 7)]$, $L_4 = [a_2, 4, 1, (8, 3)]$, $L_5 = [b_1, 5, 1, (5, 8)]$, $L_6 = [b_2, 6, 1, (6, 6)]$, $L_7 = [b_1, 7, 1, (7, 4)]$, $L_8 = [b_2, 8, 1, (8, 2)]$.

The remaining nodes of Tr_1 are $N_1 = [A_1, 5, 9, 0]$, $N_2 = [B_1, 5, 8, (5, 9)]$, $N_3 = [A_2, 6, 7, (5, 8)]$, $N_4 = [B_2, 6, 6, (6, 7)]$, $N_5 = [A_1, 7, 5, (6, 6)]$, $N_6 = [B_1, 7, 4, (7, 5)]$, $N_7 = [A_2, 8, 3, (7, 4)]$, $N_8 = [B_2, 8, 2, (8, 3)]$.

The right part of Fig. 1. displays the D-tree $dTr_1 = dT(Tr_1)$, where the nodes of dTr_1 are:
$V_1 = [a_1, 1, 5], V_2 = [a_1, 2, 7], V_3 = [a_2, 3, 6], V_4 = [a_2, 4, 8], V_8 = [b_1, 5, 0], V_7 = [b_2, 6, 5], V_6 = [b_1, 7, 6], V_5 = [b_2, 8, 7]$.

We will introduce the notion of *DR-equivalence* between two DR-trees. Informally speaking, two DR-equivalent DR-trees express the same structure of dependency relations. Thus, two DR-equivalent DR-trees potentially differ in the information about horizontal positions of nodes only.

Definition 2.6 Let us suppose that T_1, T_2 are two DR-trees with n leaves each, and with the same number of nodes. We shall say that T_1 and T_2 are *DR-equivalent* if there exists a permutation π of the sequence $(1, ..., n)$ with the



N_1
N_2
N_3
N_4
N_5
N_6
N_7
N_8
V_8
V_7
V_6
V_5
L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8 V_1 V_2 V_3 V_4

Figure 1: DR-, D-tree on $a_1a_1a_2a_2b_1b_2b_1b_2$

following properties: To any node N_1 of T_1 of the form $N_1 = [A, i, j, e_1]$ there exists exactly one node N_2 of T_2 of the form $N_2 = [A, \pi(i), j, e_2]$, where:

if $e_1 = 0$ then $e_2 = 0$; in the case that e_1 has the form $e_1 = (k, p)$ then $e_2 = (\pi(k), p)$, at the same time if $k < i$ then $\pi(k) < \pi(i)$, or if $k > i$ then $\pi(k) > \pi(i)$.

Let us remind some notions introduced informally already in the previous section.

Definition 2.7 Let Tr be a DR-tree (D-tree). Let u be a node of Tr. As $Cov(u, Tr)$ we denote the set of horizontal indices of nodes from which a path (bottom up) leads to u. $Cov(u, Tr)$ obligatorily contains the horizontal index of u. We say that $Cov(u, Tr)$ is the *coverage* of u (according to Tr).

Let there be a node u of a DR-tree (D-tree) Tr such that $Cov(u, Tr) = \{i_1, i_2, \ldots, i_n\}$, $i_1 < i_2 \ldots i_{n-1} < i_n$, $1 \leq j < n$ and $i_{j+1} - i_j > 1$. We say that the pair (i_j, i_{j+1}) forms a gap in the $Cov(u, Tr)$ (or that the DR-tree Tr contains the gap (i_j, i_{j+1})).

Let T be a DR-tree (D-tree). We will say that T_s is a *covering (induced) subtree* of T if the following holds: if r_s is the root of T_s then T_s contains all the nodes from T which have their horizontal indices from the set $Cov(r_s, T)$. In other words, T_s contains all the nodes from T which are on some path leading (bottom up) to r_s. We say that T is *DR-projective* (*D-projective*) if T does not contain any gap.

Definition 2.8 Let us define several technical notions useful in the following text. As a matter of fact, these definitions are inspired by [12].

Let G be a DG, and $T \in FT(G)$. We shall say that T_s is an *A-adjoined subtree* of T if T_s is a covering subtree of T and if the following holds:

Let R_s be the root node of T_s, let A be the symbol of R_s. Then there exist exactly one node N_s of T_s such that N_s is not equal to R_s and the symbol of N_s is A.

The covering subtree B_p of T_s with the root N_s will be called the *bottom part of T_s*.

The subtree A_p of T_s which results by removing B_p from T_s will be called *adjoining part of T_s*.

We can see that $B_p \in CrT(G)$, while A_p is not necessarily from $CrT(G)$.

We shall say that T_s is a *simply-adjoined subtree* of T if T_s is an A-adjoined subtree of T for some symbol A, and if the following does not hold:

There is a node N_d of T_s, different from the root of T_s, such that N_d is a root of a B-adjoined subtree of T for some B.

Let us say that a D-grammar G is *acyclic* if for any $T \in FT(G)$ it holds that the adjoining part of any its simply-adjoined subtree contains at least one leaf which is also a leaf of T.

It is not hard to see that those D-grammars which are not acyclic can derive infinite many DR-trees with the same D-tree. This makes such grammars inadequate for linguistic modelling.

3 Restrictions and proper DG's

Let us suppose from now on that all D-grammars we work with are acyclic. We will reintroduce also in this section the (complexity) measure of non-projectivity which was outlined in the section 2. We introduce it for DR-trees and for D-trees as well.

Definition 3.1 Let $Tr \in CrT(G)$ for some G, i.e., Tr is a DR-tree, u be a node of Tr, and $Cov(u, Tr)$ its coverage. The symbol *DR-$Ng(u, Tr)$* represents the number of gaps in $Cov(u, Tr)$. *DR-$Ng(Tr)$* (*D-$Ng(Tr)$*) denotes the maximum of $\{$*DR-$Ng(u, Tr)$*; $u \in Tr\}$. We say that *DR-$Ng(Tr)$* is the *node-gaps complexity of Tr*.

In the same way we introduce the measure for D-trees. Let $Tr \in dFT(G)$, i.e., Tr is a D-tree, let u be a node of Tr, and let $Cov(u, Tr)$ be its coverage. The symbol $D\text{-}Ng(u, Tr)$ represents the number of gaps in $Cov(u, Tr)$. $D\text{-}Ng(Tr)$ denotes the maximum of $\{D\text{-}Ng(u, Tr);\ u \in Tr\}$. We say that $D\text{-}Ng(Tr)$ is the *node-gaps complexity of* Tr.

Example 3.2 *We stick to the* DR-tree Tr_1 *from the Fig. 1. The following coverages contain gaps:*

$Cov(N_3, Tr_1) = \{2, 3, 4, 6, 7, 8\}$	*has one gap (4,6),*
$Cov(N_4, Tr_1) = \{2, 4, 6, 7, 8\}$	*has two gaps (2,4), (4,6),*
$Cov(N_5, Tr_1) = \{2, 4, 7, 8\}$	*has two gaps (2,4) and (4,7),*
$Cov(N_6, Tr_1) = \{4, 7, 8\}$	*has one gap (4,7),*
$Cov(N_7, Tr_1) = \{4, 8\}$	*has one gap (4,8).*

We can see that $DR\text{-}Ng(Tr_1) = 2$.

Definition 3.3 Let $G = (V_T, V_N, P, S)$ be a DG. We shall say that G is a *proper DG* if the following holds:

If $dT \in dFT(G)$ and $D\text{-}Ng(dT) = 0$ (dT is D-projective), then there exists $T \in FT(G)$ such that $dT = dT(T)$ and $DR\text{-}Ng(T) = 0$ (T is DR-projective).

Example 3.4 Let us choose a DG G_{Ll} in the following way: $G_{Ll} = (T, N, \{C\}, P)$, where $T = \{a, b, c, d\}$, $N = \{A, B, C\}$, $P = \{A \rightarrow_L Bb, B \rightarrow_L Cc, C \rightarrow_L Aa, C \rightarrow d\}$.

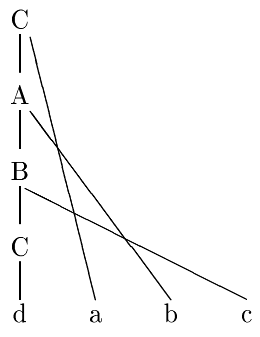
We can see from Fig.2 that G_{Ll} is not a proper grammar. Namely it generates only projective (very simple) D-trees of depth 1, and on the other hand it generates not only a nonempty, but even an infinite set of strings (D-trees) corresponding to nonprojective DR-trees.

Definition 3.5 Let $G = (V_T, V_N, P, S)$ be a DG, and Cs be a set of *gap restrictors* , i.e. pairs of the shape $[A, i]$, where $A \in V_T \cup V_N$ and $i \in Nat^+$. We say that the pair $G_{Cs} = (G, Cs)$ is a *restricted DG* (*RsD-grammar, RsDG*), and if G is a proper grammar we say that $G_{Cs} = (G, Cs)$ is a *restricted proper DG* (*prop-RsDG*).

Let us define the set of DR-trees $CsT(G_{Cs})$ for a RsDG $G_{Cs} = (G, Cs)$.

$CsT(G_{Cs}) = \{T \in FT(G)|$ *if* T_A *is a covering subtree of* T *with the root symbol* A*, and* $[A, j] \in Cs$*, then* $DR\text{-}Ng(T_A) \leq j\}$.

Let $i \in (Nat^+ \cup \{*\})$ *and let us consider that* $*$ *is greater than any natural number, and let* $G_{Cs} = (G, Cs)$ *is a RSDG. Then we define the following:*



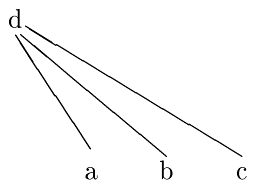


Figure 2: DR-, D-tree on the string *dabc* by G_{Ll}

- *DR-$T(w, G_{Cs}, i)$ is the set of DR-trees such that for any of its member T the following three conditions are met:*

  a) $T \in FT(w, G)$

  b) DR-Ng$(T) \leq i$,

  c) $T \in CsT(G_{Cs})$.

  For $i = *$ only the two following conditions are taken in account:

  a) $T \in FT(w, G)$,

  c) $T \in CsT(G_{Cs})$.

- *DR-$L(G_{Cs}, i) = \{w|$ DR-$T(w, G_{Cs}, i) \neq \emptyset\}$.*

- *DR-$T(G_{Cs}, i)$ denotes the union of all DR-$T(w, G_{Cs}, i)$ over all $w \in$ DR-$L(G_{Cs}, i)$.*

- *DR-$\mathcal{L}(i)$ denotes the class of languages DR-$L(G_s, i)$, for all RsDG's G_s.*

- *For $X \in \{DR, DR{-}prop\}$ and $Y \in \{T(w, G_{CS}, i), T(G_{CS}, i), L(G_{CS}, i), \mathcal{L}$ (i), RsDG$\}$ we shall use the combination of denotations of the form X-Y, with the straightforward meaning (the ordinary DG's can be substituted by proper DG's in the previously introduced notions) E.g., DR-prop-$\mathcal{L}(i)$*

denotes the class of languages DR-prop-L(G_s, i), for all proper RsDG's G_s.

Let us recall an obvious proposition from [18].

Proposition 3.6 *Let $GS = (G, Cs)$ be a RsDG, $n \in Nat$. Then*

$$DR\text{-}T(GS,0) \subseteq DR\text{-}T(GS,1) \subseteq ... \subseteq DR\text{-}T(GS,n)... \subseteq DR\text{-}T(GS,*),$$

$$DR\text{-}L(GS,0) \subseteq DR\text{-}L(GS,1) \subseteq ... \subseteq DR\text{-}L(GS,n)... \subseteq DR\text{-}L(GS,*).$$

If the set of restrictions Cs is empty, then also DR-T$(GS,*) = FT(G)$.

Moreover, for any DR-tree $T \in$ DR-T(GS,i) $i \in \{0,1,2,...,*\}$, there exists a DR-equivalent DR-tree $T_p \in$ DR-T(GS,0), and therefore for any string $w \in$ DR-L(GS,i) there exists a permutation $\pi(w)$ of w such that $\pi(w) \in$ DR-L(GS,0).

Remark 3.7 The previous proposition presents in a formal way the stepwise relaxation of the word order described by a *RsD*-grammar. A complete scale of formal languages according to a RsD-grammar is obtained in this way. On the other hand, for any relaxed DR-tree the grammar determines its projective DR-equivalent variant (representation).

The following paragraphs focus on results concerning classes of languages which arise through a certain degree of relaxation. The next proposition resembles a claim from [18].

Proposition 3.8 $DR\text{-}\mathcal{L}(0) \subseteq DR\text{-}\mathcal{L}(1) \subseteq ... \subseteq DR\text{-}\mathcal{L}(n)... \subseteq DR\text{-}\mathcal{L}(*)$,

$$DR\text{-}prop\text{-}\mathcal{L}(0) \subseteq DR\text{-}prop\text{-}\mathcal{L}(1) \subseteq ... \subseteq DR\text{-}prop\text{-}\mathcal{L}(n)... \subseteq DR\text{-}prop\text{-}\mathcal{L}(*).$$

Proof: We can see that for any $L_1 \in$*DR-L*(i), where $i \in Nat^+$, there exists an RsD-grammar $G_1 = (G, Cs_1)$ such that $L_1 \in$*DR-L*(G_1, i).
We construct a new RsD-grammar $G_2 = (G, Cs_2)$ from G_1 by keeping the unrestricted dependency grammar G and changing the constrains Cs_1 into Cs_2. Let $Cs_1 = \{[A_1, j_1], ..., [A_k, j_k]\}$, and let the symbols $B_1, ..., B_p$ be the unrestricted symbols of G. Let us define $Cs_2 = \{[B_1, i], ..., [B_p, i], [A_1, m_1], ..., [A_k, m_k]\}$, where for any $n \in \{1, ..., k\}$ such that $j_n < i$ holds $m_n = j_n$, and for any $n \in \{1, ..., k\}$ such that $j_n \geq i$ holds $m_n = i$.
It is not difficult to see that for any $j \in \{i, i+1, ..., *\}$ the equality $L(G_1, i)$= $L(G_2, j)$ holds. That proves directly the first sequence of (improper) inclusions.

Following the fact that only the restrictions Cs_1 are changed in the above construction, we can see that the proper grammars are transformed into the proper grammars. **q.e.d.**

Remark 3.9 The improper inclusions in the previous proposition can be replaced by proper inclusions. The following considerations are made in order to show the property of the proper inclusion using the word-order relaxation ability (relax-ability) of proper dependency grammars only. We shall sometimes use the fact that any DG Gs for which it holds that to any node of any $dT \in$ *dFT(Gs)* leads at most one L-edge and at most one R-edge, is a proper DG.

Let us remind an obvious propositition from [18].

Proposition 3.10 *DR-$\mathcal{L}$(0) = CF^+*

Definition 3.11 [Degrees of relax-ability (due to RsDG's)] Let $k \in Nat$. We shall say that a grammar GS has the *degree of DR-relax-ability equal to k ($DRS(GS) = k$)* if the following two conditions are fulfilled:

a) *DR-L*$(GS, i) \notin$ *DR-$\mathcal{L}$*$(i-1)$, for $i \in \{1, 2, ..., k\}$, and

b) *DR-L*$(GS, k) =$ *DR-L*$(GS, k+j)$ for any $j \in Nat$.

Denotation 3.12 Let us denote as G_i the following DG for any $i \in Nat$:
$G_i = (T, N, \{A_1\}, P)$, where $T = \{a_1, a_2, \ldots, a_i, b_1, b_2, \ldots, b_i\}$, $N = \{A_1, A_2, \ldots, A_i, B_1, \ldots, B_i\}$, $P = \{A_1 \rightarrow_R a_1B_1, B_1 \rightarrow_L b_1A_2, A_2 \rightarrow_R a_2B_2, \ldots, A_i \rightarrow_R a_iB_i, B_i \rightarrow_L b_iA_1, B_i \rightarrow b_i\}$.
Let us denote $w_{(i,j,n)} = a_1^n a_2^n ... a_j^n (b_1 b_2 ... b_j a_{j+1} b_{j+1} ... a_i b_i)^n$ for any natural $n, i > 0$, $i \geq j \geq 0$. Let us take $W(i,j) = \{w_{(i,j,n)} | n \in Nat\}$ for $i > 0$,$i \geq j \geq 0$.
For any even $i > 1$, we shall denote the RsD-grammar $(G_i, [A_1, i/2])$ as GS_i.
For any even $i > 0$, $j \geq 0$ we shall denote *DR-L*(GS_i, j) as *DR-L*(i, j).

Proposition 3.13 *Let dT $\in$ D-T$(GS_i, 0)$ for some even i and dT = dT(T) for some T $\in$DR-T$(GS_i, *)$. Then T $\in$DR-T$(GS_i, 0)$. It means that that for any even i GS_i is a proper RsDG's.*

Proof: We can see that at most one L-edge and at most one R-edge lead to any node of $dT \in dFT(G_i)$. Now we can easily see that G_i is a proper DG. Therefore the GS_i is a proper RsDG. **q.e.d.**

Proposition 3.14 *Let us suppose that $i \geq j \geq 0$, $i > 0$ is an even number, j is an even number. Then $W(i,j) \subseteq$DR-L$(i, j/2)$.*

Proof: Let $0 \leq j \leq i$, j is an even number, $i > 0$ is an even number. Let us show that $w_{(i,j,n)} = a_1^n a_2^n ... a_j^n (b_1 b_2 ... b_j a_{j+1} b_{j+1} ... a_i b_i)^n \in L(GS_i, j/2)$ for any natural $n > 0$.
First let us outline the rough idea: We shall choose a sequence of reductions by the grammar G_i of a word of the form $w_{(i,j,n)}$ in which certain cycles are repeated. We shall see in the more detailed part that in the first cycle an incomplete DR-tree with $j/2$ gaps will be constructed. We shall show that by proceeding in similar cycles, it is possible to obtain a DR-tree T_{an} from *DR-T*$(w_{(i,j,n)}, GS_i, j/2)$.
Now we describe the procedure of reduction. We can see that $w_{(i,j,n)}$ has the length equal to $2 \cdot n \cdot i$. Let us denote the s-th position ($1 \leq s \leq 2 \cdot n \cdot i$) in $w_{(i,j,n)}$ as p_s. We can see (for example) that the leftmost b_1 occurs on the $p_{n \cdot j+1}$ of any $w_{(i,j,n)}$.
The sequence of reductions starts by the application of the rule $B_i \rightarrow b_i$ to the symbol b_i in the rightmost position $p_{2 \cdot n \cdot i}$. The symbol b_i will be rewritten by the symbol B_i. The second step will be the application of the rule $A_i \rightarrow_R a_i B_i$. If $i > j$ then the rule is applied to the rightmost occurrence of a_i, i.e. to the position $p_{2 \cdot n \cdot i-1}$ and to the single occurrence of B_i. The symbol a_i will be deleted and the symbol B_i will be rewritten to A_i. This application of the rule does not create any gap.
If $i = j$ then the rule is applied to the leftmost occurrence of a_i, i.e. to the position $p_{n \cdot (j-1)+1}$ and to the single occurrence of B_i. This application of the rule creates the first gap.
The third step will be the application of the rule $B_{i-1} \rightarrow_L b_{i-1} A_i$ to the leftmost occurrence of the symbol b_{i-1} and to the single occurrence of the symbol A_i. Next we shall use the rule $A_{i-1} \rightarrow_R a_{i-1} B_{i-1}$. The rule is applied to the rightmost occurrence of a_{i-1}, and to the single occurrence of B_{i-1}. This application of the rule does not create a new gap.
The sequence of reductions continues by the application of the rule $B_{i-2} \rightarrow b_{i-2} A_{i-1}$ to the rightmost symbol b_{i-2} and to the single occurrence of A_{i-1}. The next step will be the application of the rule $A_{i-2} \rightarrow_R a_{i-2} B_{i-2}$. If $i - 2 > j$ the rule is applied to the rightmost occurrence of a_{i-2}, and to the single occurrence of B_{i-2}. This application of the rule does not create any gap.
If $i - 2 \leq j$ then the rule is applied to the current leftmost occurrence of a_{i-2}, and to the single occurrence of B_{i-2}. This application of the rule creates a new gap (if $n > 1$).

The next step will be the application of the rule $B_{i-3} \rightarrow_L b_{i-3}A_{i-2}$, to the leftmost occurrence of the symbol b_{i-3} and to the single occurrence of the symbol A_{i-2}. This application of the rule does not create a new gap.
We can continue in this way until we obtain for the first time the nonterminal A_1 in the reduced string. The reduced string has the form
$w_{(i,j,n-1)}A_1 = a_1^{n-1}a_2^{n-1}...a_j^{n-1}(b_1b_2...b_ja_{j+1}b_{j+1}...a_ib_i)^{n-1}A_1$.
It is not hard to see that in the course of obtaining $w_{(i,j,n-1)}A_1$ as described, we create an incomplete DR-tree T_{a1} with *DR-Ng*$(T_{a1}) \leq j/2$ which contains exactly $j/2$ gaps for $n > 1$.
We have reduced $w_{(i,j,n)}$ into $w_{(i,j,n-1)}A_1$ during the construction of T_{a1}. If $n > 1$ we can use almost the same sequence of rules (by substituting the starting one) in order to obtain a DR-tree T_{a2} from T_{a1} in such a way that *DR-Ng*$(T_{a2}) = j/2$. At the same time the string $w_{(i,j,n-1)}A_1$ is reduced into the string $w_{(i,j,n-2)}A_1$. In this way we can stepwise obtain the complete DR-tree T_{an} such that $T_{an} \in$*DR-T*$(w_{(i,j,n)}, GS_i, j/2)$. **q.e.d.**

Proposition 3.15 *For any even i and even j, $i \geq j > 1$, DR-$L_{(i,j)} \notin$ DR-$\mathcal{L}(0)$.*

Proof: Let us suppose that *DR-*$L_{(i,j)} \in$ DR-$\mathcal{L}(0)$. Because *DR-*$\mathcal{L}(0)$= CF^+, the language *DR-*$L_{(i,j)}$ must be a context-free language. Any word from *DR-*$L_{(i,j)}$ must contain an equal number of the symbols a_k (and b_k) for all $1 \leq k \leq i$. We can see by the pumping lemma for context-free languages that $W(i,k) = \{w_{(i,k,n)} | n \in Nat\}$, where $w_{(i,j,n)} = a_1^n a_2^n...a_j^n(b_1b_2...b_ja_{j+1}b_{j+1}...a_ib_i)^n$, cannot be a subset of any context-free language with the above mentioned equal-number-property for *DR-L*(i,j). This constitutes a contradiction with the presupposition. **q.e.d.**
The separation results for the other classes of the considered hierarchies are based on similar observation as the previous separation result. The formulation of this observations is slightly more complex than in the previous case.

Proposition 3.16 *Let us suppose that GS is a RsDG such that DR-$L_{(i,k)}$ = DR-$L(GS,j)$, for some i,j,k, $i \geq 2k \geq 0$, $i \geq j \geq 0$. Let $T \in$ DR-$T(GS,j)$, and T_s is a simply-adjoined subtree of T. Then there is $n_1 > 0$ such that the adjoining part A_s of T_s contains n_1 leaves with the symbol a_m and the same number n_1 of leaves with the symbol b_m for any $m \in \{1,...,i\}$.*

Proof: Let us suppose that $T \in$*DR-T*(GS,j), and T_s is a simply-adjoined subtree of T and let $GS = (G, Cs)$. Let us recall that we suppose that G is a lexicalized DG. Via proposition 3.6 we can see that there is a DR-tree $T_0 \in$*DR-T*$(GS,0)$ which is DR-equivalent with T. Further, it is easy to see that T_0

contains a simply-adjoined subtree T_{s0} which differs from T_s in the order of nodes only (in the horizontal and domination indices of the nodes only). We can see that the adjoining part of T_{s0} can be adjoined (due to lexicalization of G really "pumped", possibly several times) in order to obtain again a DR-tree from $DR\text{-}T(GS, 0)$, and therefore also from $DR\text{-}T(GS, j)$. Using this observation we can easily find a contradiction. **q.e.d.**

Proposition 3.17 *Let us suppose that $GS = (G, Cs)$ is a RsDG such that r is the number of rules of G, $DR\text{-}L_{(i,k)} = DR\text{-}L(GS, j)$, for some i, j, k, where $i > 0$, $i \geq 2k \geq 0$, $i \geq j \geq 0$. Let $w = w(i, 2k, n)$, $n \geq 4^{r+1}$, and let $T_w \in DR\text{-}T(w, GS, j)$. Then $DR\text{-}Ng(T_w) \geq k$, and therefore also $j \geq k$.*

Proof: Let us note at first that the $n \geq 4^{r+1}$ of the proposition is chosen in such a way that it can be shown that any simply-adjoined subtree T_s of T_w contains at least k gaps. In this way we shall show that $DR\text{-}Ng(T_s) \geq k$. We can see that the whole T_s contains at most 2^r leaves, because of the fact that only the symbol of the root of T_s is repeated exactly once somewhere inside T_s. This means that any path within T_s is shorter than $r + 2$. We can see from the proposition 3.16 that each symbol from the set $\{a_1, ..., a_i, b_1, ..., b_i\}$ must occur at least once among the leaves of the adjoining part of T_s.
Let r_s be the root of T_s. We shall show that the coverage $Cov(r_s, T_w)$ contains at least k gaps.
Let us consider that $i \geq 2$, $w = w_{(i,2k,n)} = {a_1}^n ... a_{2k}^n (b_1 ... a_{2 \cdot k+1} b_{2 \cdot k+1} ... a_i b_i)^n$, where $n \geq 4^{r+1}$. We can see that $r > i$ and T_s contains at most $(2^{r+1})/i$ symbols a_1 (at least one) and also at most $(2^{r+1})/i$ occurrences of each of symbols a_2,...a_{2k+1}, ..., a_i,...b_i and at least one of each of them. We can see that w has such a shape that $Cov(r_s, T)$ contains at least k gaps, because any gap-free string of leaves of T_s contains occurrences of at most two symbols from the set $\{a_1, a_2, ..., a_{2k}\}$ and these two symbols must be of the form a_s, a_{s+1} for any $s \in \{1, ..., 2k - 1\}$. This proves this proposition. **q.e.d.**

Proposition 3.18 *For any even $i > 0$, $i \geq 2k$, $k > j \geq 0$ there does not exist any RsD-grammar GS such that $DR\text{-}L_{(i,k)} = DR\text{-}L(GS, j)$.*

Proof: Let us suppose that there is such a grammar GS. We can see a contradiction using Proposition 3.17. **q.e.d.**

Proposition 3.19 *For any even $i > 0$ it holds that $DRS(GS_i) = i/2$, i.e. the degree of relax-ability of the grammar GS_i is equal to $i/2$.*

Proof: This proposition is a direct consequence of the previous proposition. **q.e.d.**

The following theorem is also a direct consequence of the previous propositions.

Theorem 3.20 *For any $j \in Nat$ there is a prop-RsDG Gp_j such that $j = DRS(Gp_j)$.*

Theorem 3.21

$DR\text{-}\mathcal{L}(0) \subset DR\text{-}\mathcal{L}(1) \subset ... \subset DR\text{-}\mathcal{L}(5)... \subset DR\text{-}\mathcal{L}(*)$

$DR\text{-}prop\text{-}\mathcal{L}(0) \subset DR\text{-}prop\text{-}\mathcal{L}(1) \subset ... \subset DR\text{-}prop\text{-}\mathcal{L}(n)... \subset DR\text{-}prop\text{-}\mathcal{L}(*)$.

Proof: It follows from the previous theorem that for any $i \in Nat$ there exists a language $Lr_i \in DR\text{-}prop\text{-}\mathcal{L}(i)$ such that $Lr_i \notin DR\text{-}\mathcal{L}(i-1)$. The rest is a consequence of Proposition 3.6. **q.e.d.**

Remark 3.22 *In fact it holds that $DR\text{-}prop\text{-}\mathcal{L}(0) = CF^+$. We will not prove this equality in this paper.*

4 Remarks on parsing

The previous notions and results are implicitly connected with parsing considerations through the concept of coverage. The next propositions illustrate this fact.

Proposition 4.1 *To any RsDG GS there exists a (sequential) algorithm Am computing for any string w an $i \in Nat^+$, such that i is the smallest element of Nat^+ for which $w \in DR\text{-}L(GS, i)$, or, if such an i does not exist, returning a message about the fact that $w \notin L(GS, *)$. Moreover, for a given $i \in Nat^+$ Am recognizes the membership $w \in L(GS, i)$ in a polynomial time (compared with the length of w and the size of GS), where the degree of the polynomial increases with i.*

Proof: We outline the main idea only. Am is a bottom-up (CYK-like) parsing algorithm based on a stepwise computation of pairs (items) of the shape (U, Cv), where U means a node of a DR-tree by GS and Cv means its coverage related to the given input-word. At first the items (coverages) without gaps are computed,

second items with one gap, third items with two gaps, etc., until such an item (R, Ci) is obtained, where R contains a root-symbol and Ci covers completely the input string w.
With the *DR-Ng* limited by a constant k (*DR-Ng* can be interpreted as the maximum of the number of gaps in the coverages during a computation of the parser), the number of items depends polynomially on the size of the input, and on the number of symbols (terminals, nonterminals) of the grammar GS. The degree of the polynomial depends on k. The assertion of the claim can be derived from this observation. **q.e.d.**

Corollary 4.2 *There exists a sequential algorithm such that for any $i \in Nat^+$ and any $L \in$ DR-$\mathcal{L}(i)$, the algorithm recognizes L in a polynomial time, where the degree of the polynomial increases with i.*

Remark 4.3 *We believe that there exists an $i \in Nat^+$ for which there does not exist an (sequential) algorithm recognizing every language from D-$\mathcal{L}(i)$ in a polynomial time. We have even the suspicion, due to the results from [18], that this i can be equal to 0. This remark introduces the next corollary.*

Corollary 4.4 *There exists a sequential algorithm recognizing any $L \in$D-prop-$\mathcal{L}(0)$ in a polynomial time.*

Conjecture. We believe that any $L \in$*D-prop-*$\mathcal{L}(i)$ for any $i \in Nat^+$ is recognizable in a polynomial time.

5 Conclusions

The main aim of this paper is to lay theoretical foundations for a formal framework able to serve as a linguistically adequate tool for expressing word order regularities in natural languages with various word order properties, with a particular stress on the description of relaxations and restrictions of word-order within languages with a high degree of word order freedom. This framework supports also a formal presentations of properties of an implemented parser (environment for parsers-development) and of an implemented grammar for Czech presented in [11] and [13].
An important novel contribution is to be seen particularly in the fact that the approach to word-order studied here is in its spirit substantially different from the tools used for strengthening the power of (CF-)grammars, namely from the often studied principle of (pure) regulation, cf. [12]. In particular, the approach

of (pure) relaxation proposed here adds massively new word-order interpretations of the ('context-free') rules used. To the contrary, the principle of (pure) regulation (e.g., of tree-adjoining or vector grammars) directly decreases the number of such interpretations. We believe strongly that in this point, the approach advocated in this paper is more adequate linguistically, and hence also it is a new adequate tool for linguistic description. A strong evidence for this claim seems to be fact that, unlike many other approaches, the notions studied here are intrinsically connected to formal tools for measuring (and hence comparing) degrees of word order freedom in different natural languages. Thus, the approach allows for expressing formally (even: quantitatively) the well-established empirical observation that some natural languages display more word-order freedom than others - in fact, the tools proposed indeed generate a whole scale of degrees of word-order freedom. The fact that this is relevant not only for linguistic aims, but that these differences in word order freedom have also important consequences for parsing complexity, which is studied in the final sections of the article, shows another aspect of the relevance of the approach proposed.
As for research direction in the next future, we shall try to perform a more thorough comparison between the degrees of nonprojectivity based on DR-trees on the one hand and the degrees of nonprojectivity based on D-trees on the other hand. In the paper [18] we have used deliberately such a sequence of 'witness' grammars that it has an increasing degree of DR-nonprojectivity (relax-ability) but by which all the parsed D-trees remain D-projective. We consider this property very 'nonliguistic', compare, e.g., [14]. In this paper we worked in a similar way with some more appropriate classes of DG's. We have also outlined that in this way we can obtain more satisfactory parsing-complexity estimations for dependency-based parsing.

Acknowledgement

This work has been supported by grant of GAČR No. 201/99/0236 to Martin Plátek, Tomáš Holan and Vladislav Kuboň. The work of Karel Oliva has been sponsored by the *Fonds zur Foerderung der wissenschaftlichen Forschung (FWF), Grant No. P13224*. The *Austrian Research Institute for Artificial Intelligence* is supported by the *Austrian Federal Ministry of Education, Science and Culture.*

References

[1] J.Dassow, G.Păun: Regulated Rewriting in Formal Language Theory, Akademie-Verlag Berlin, 1989 *Handbook of Formal Languages*, G. Rosenberg and A. Salomaa (eds.), Berlin and Heidelberg: Springer, 1997.

[2] A.Dikovskij, L.Modina : Dependencies on the Other Side of the Curtain, Traitement Automatique des Langues (T.A.L, guest editor S.Kahane), Vol. 41, No. 1, Hermes, pp.79-111.

[3] S. J. Fitialov: On the equivalence of ps-grammars and dependency grammars. In: Problems of Structural Linguistics. "Nauka", 1968, pp. 100 -114

[4] H.Gaifman: Dependency system and Phrase -Structure System, Information and Control, Vol 8, 1965, pp. 304-337

[5] A.V. Gladkij: Formal'nye grammatiki i jazyki, Iz.: NAUKA, Moskva, 1973

[6] S.A. Greibach, J.E. Hopcroft: Scattered context grammars, JCSS 3 (1969) 233-247.

[7] T.Holan, V.Kuboň, M.Plátek: An Implementation of Syntactic Analysis of Czech, in: Proceedings of IWPT' 95, Charles University Prague, 1995, pp. 126-135

[8] T.Holan, V.Kuboň, M.Plátek : A Prototype of a Grammar Checker for Czech, Proceedings of the Fifth Conference on Applied Natural Language Processing, Association for Computational Linguistics, Washington, March 1997, pp.147-154

[9] T. Holan, V.Kuboň, K.Oliva, M.Plátek: "Two Useful Measures of Word Order Complexity", in *Proceedings of the Coling '98 Workshop "Processing of Dependency-Based Grammars"*, A. Polguere and S. Kahane (eds.), University of Montreal, Montreal, 1998

[10] T.Holan, V.Kuboň, K.Oliva, M.Plátek. *On Complexity of word-order.* In: Traitement automatique des langues (T.A.L., guest editor S.Kahane), Vol. 41, No 1, 2000, pp. 243-267.

[11] T.Holan: Nástroj pro vývoj závislostních analyzátorů přirozených jazyků s volným slovosledem, (A Software Environment for the Development Of Dependency Parsers for Natural Languages with Free Word Order), in Czech, PhD thesis, Charles University, Prague, 2001

[12] A.K. Joshi, and Y.Shabes: Tree-Adjoining Grammars, in *Handbook of Formal Languages*, Vol 3, G. Rosenberg and A. Salomaa (eds.), Berlin and Heidelberg: Springer, 1997, pp. 69-123

[13] V.Kuboň: Problems of Robust Parsing of Czech, PhD thesis, Charles University, Prague, 2001

[14] J.Kunze : *Abhängigskeitsgrammatik*, Berlin: Akademie-Verlag, 1975

[15] S.Marcus: "Sur la notion de projectivité", in *Zeitschrift fuer mathematische Logik und Grundlagen der Mathematik XI*, 1965, pp. 181-192.

[16] A. Meduna: Syntactic Complexity of Scattered Context Grammars, *Acta Informatica 32* (1995) 285-298.

[17] A. Nasr: A Formalism and a Parser for Lexicalized Dependency Grammars, in: Proceedings of IWPT' 95, Charles University Prague, 1995, pp. 186-195

[18] M.Plátek, T.Holan, V.Kuboň. *On Relax-ability of Word-Order by D-grammars.* In: Combinatorics, Computability and Logic. C.S. Calude and M.J. Deneen (eds.), Springer Verlag, Berlin, 2001, pp. 159-174.

[19] P.Sgall, E.Hajičová, J.Panevová: *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*, Dordrecht: Reidel and Prague: Academia, 1986

[20] K. Sikkel, A. Nijholt : "Parsing of Context-Free Languages", in *Handbook of Formal Languages*, Vol 2, G. Rosenberg and A. Salomaa (eds.), Berlin and Heidelberg: Springer, 1997, pp. 61-100

Two-dimensional Context-free Grammars[2]

Daniel Průša

Department of Theoretical Computer Science
Charles University, Faculty of Mathematics and Physics
118 00 Praha 1, Malostranské nám. 25
Czech Republic

prusa@barbora.ms.mff.cuni.cz

Abstract.
We present a generalization of context-free grammars to two dimensions and define picture languages generated by these grammars. We examine some properties of the formed class and we describe how these languages can be recognized by two-dimensional forgetting automata.

1. Introduction

The goal of this paper is to present one of the possible generalizations of concepts of context-free grammars and languages to two dimensions. Informally, a two-dimensional string (called a picture) is defined as a rectangular array of symbols from a finite alphabet. A picture language is a set of pictures.

Some proposals of two-dimensional context-free languages already exist ([4], [2]), however a complete theory has not been formed yet. It is a difficult task. The situation is rather complicated even in case of regular languages. We emphasize that our ambitions are not to claim what two-dimensional context-free languages should be. We generalize concepts of context-free grammars in a natural way only and study the formed class of picture languages. However, in the text, we use terms like two-dimensional context-free grammar, resp. language to refer to them.

Our generalized grammars have productions whose left sides are non-terminals and the right sides are matrixes of terminals and non-terminals. This

[2]Supported by the Grant Agency of the Czech Republic, Grant-No. GA ČR 201/02/1456

idea is not original. It can be found for example in [7], where productions with the right side restricted to one row or column are considered only. The other example is in [8]. Some basic facts, that we extend, are mentioned there.

Our results on the class of context-free languages include the facts that not all languages recognized by (two-dimensional) finite state automata are context-free and that the restricted grammars from [7] are weaker than the presented grammars. In addition, we describe how context-free languages can be recognized by two-dimensional forgetting automata. This construction is based on results in [8] and [1].

2. Picture Languages

We assume that the reader is familiar with the theory of one-dimensional languages as can be found for example in [3]. We extend some basic definitions from the one-dimensional theory now. More details can be found in [4].

Definition 1 *A* picture *over a finite alphabet Σ is a two-dimensional rectangular array (matrix) of elements of Σ. Σ^{**} denotes the set of all pictures over Σ. A* picture language *over Σ is a subset of Σ^{**}.*

Let $O \in \Sigma^{**}$ be a picture. $rows(O)$, resp. $cols(O)$ denotes the number of rows, resp. columns of O. The pair $rows(O) \times cols(O)$ is called the *size* of O. We say that O is a square picture of the size n if $rows(O) = cols(O) = n$. The *empty picture* Λ is the only picture of the size 0×0. For integers i, j such that $1 \le i \le rows(O)$, $1 \le j \le cols(O)$, $O(i,j)$ denotes the symbol in O at the coordinate (i,j). A *sub-picture* of O is a sub-matrix of it.

We use $[a_{ij}]_{m,n}$ to denote the matrix

$$\begin{array}{ccc} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{array}$$

We define two binary operations – the row and the column concatenation. Let $A = [a_{ij}]_{k,l}$ and $B = [b_{ij}]_{m,n}$ be non-empty pictures over Σ. The column concatenation $A \varnothing B$ is defined iff $k = m$ and the row concatenation $A \ominus B$ iff

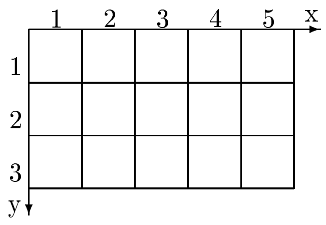


Figure 3: The system of coordinates used in our picture descriptions.

$l = n$. The results of the operations are given by the following schemes:

$$A \Phi B = \begin{array}{cccccc} a_{11} & \dots & a_{1l} & b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{kl} & b_{m1} & \dots & b_{mn} \end{array} \qquad A \ominus B = \begin{array}{ccc} a_{11} & \dots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{kl} \\ b_{11} & \dots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{array}$$

Moreover, the column and the row concatenation of A and Λ is always defined and Λ is the neutral element for both operations.

The unary operation $\bigoplus$ is defined on a set of matrixes whose elements are pictures over a fixed alphabet. Let P_{ij}, $i = 1, \dots, m$, $j = 1, \dots, n$ be pictures over Σ such that $\forall i \in \{1, \dots, m\}\, rows(P_{i1}) = rows(P_{i2}) = \dots = rows(P_{in})$ and $\forall j \in \{1, \dots, n\}\, cols(P_{1j}) = cols(P_{2j}) = \dots = cols(P_{mj})$. $\bigoplus[P_{ij}]_{m,n}$ is defined as $P_1 \ominus P_2 \ominus \dots \ominus P_m$, where $P_k = P_{k1} \Phi P_{k2} \Phi \dots \Phi P_{kn}$.

In our descriptions we use the system of coordinates in a picture depicted in Fig. 3. Speaking about a position of a specific field, we use words like up, down, right, left, first row, last row etc. with respect to this scheme.

3. Two-dimensional Automata

The two-dimensional Turing machine works on a two-dimensional tape. It can move its head left, right, up and down. We give its formal definition only. Terms like configuration, computation, accepting, recognized language, etc. are defined in a natural way. Details can be found in [4].

Definition 2 Two-dimensional Turing machine *is a tuple* $(Q, \Sigma, \Sigma_0, q_0, \delta, Q_F)$, *where* Q *is a finite set of states,* Σ *is a tape alphabet,* $\Sigma_0 \subset \Sigma$ *is an input alphabet,* $q_0 \in Q$ *is the initial state,* $Q_F \subseteq Q$ *is a set of final states and* $\delta : \Sigma \times Q \rightarrow 2^{\Sigma \times Q \times \mathcal{M}}$ *is a transition relation.* $\mathcal{M} = \{L, R, U, D, N\}$ *is the set of automaton movements (left, rigth, up, down, no movement). We always assume that there is a distinguished symbol* $\# \in \Sigma \setminus \Sigma_0$ *called the background symbol.*

A two-dimesional Turing machine is *bounded* iff the head does not leave an input during the computation (when it encounteres # it returns in the next step and does not rewrite this symbol). We consider bounded machines in the following text only. A two-dimensional finite state automaton is a two-dimensional Turing machine that does not rewrite any symbol during its computation. We abbreviate it as FSA, deterministic FSA as $DFSA$.

4. Two-dimensional Context-free Grammars

Definition 3 A two-dimensional context-free grammar G *is a tuple* $(V_N, V_T, S_0, \mathcal{P})$, *where* V_N *is a finite set of non-terminals,* V_T *is a finite set of terminals,* $S_0 \in V_N$ *is the initial non-terminal and* $\mathcal{P}$ *is a finite set of productions of the form* $N \rightarrow W$, *where* $N \in V_N$ *and* $W \in (V_N \cup V_T)^{**} \setminus \{\Lambda\}$. *In addition,* $\mathcal{P}$ *can contain* $S_0 \rightarrow \Lambda$. *In such a case,* S_0 *is not a part of any right side of productions.*

Definition 4 *Let* $G = (V_N, V_T, S_0, \mathcal{P})$ *be a two-dimensional context-free grammar. We define a picture language* $\mathcal{L}(G, N)$ *over* V_T *for every* $N \in V_N$. *The definition is given by the following recursive description:*

A) If $N \rightarrow W$ *is a production in* $\mathcal{P}$ *and* $W \in {V_T}^{**}$, *then* W *is in* $\mathcal{L}(G, N)$.

B) Let $N \rightarrow [A_{ij}]_{m,n}$ *be a production in* $\mathcal{P}$ *(not* $S_0 \rightarrow \Lambda$*) and* P_{ij} *(*$i = 1, \ldots, m$ $j = 1, \ldots, n$*) pictures such that: For every pair of indexes i,j, if* A_{ij} *is a terminal then* P_{ij} *is the picture of the size* 1×1 *whose only field contains the symbol* A_{ij}. *If* A_{ij} *is non-terminal then* $P_{ij} \in \mathcal{L}(G, A_{ij})$. *In addition,* $\bigoplus[P_{ij}]_{m,n}$ *is defined. Then* $\bigoplus[P_{ij}]_{m,n}$ *is an element of* $\mathcal{L}(G, N)$.

The set $\mathcal{L}(G, N)$ *contains just all pictures that can be obtained by applying a finite sequence of rules A) and B). The language* $\mathcal{L}(G)$ *generated by the grammar* G *is defined as the language* $\mathcal{L}(G, S_0)$.

We abbreviate a two-dimensional context-free grammar as CFG. $\mathcal{L}(CFG)$ is the class of all two-dimensional context-free languages. CF stands for context-free. An equivalent definition of a language generated by a context-free grammar is based on a generalization of derivation trees.

Definition 5 *Let $G = (V_N, V_T, S_0, \mathcal{P})$ be a CFG. A derivation tree for G is every tree T satisfying:*

- *T has at least two vertices.*
- *Each vertex v of T is labeled by a pair $(a, k \times l)$. If v is a leaf then $a \in V_T$, $k = l = 1$ else $a \in V_N$, $k, l \geq 1$ are integers.*
- *Edges are labeled by pairs (i, j). Let us denote the set of labels of all edges connecting v with its descendant as $I(v)$. It holds that $I(v) = \{1, \ldots, m\} \times \{1, \ldots, n\}$ and $m.n$ is the number of descendants of v.*
- *Let v be a vertex of T labeled $(N, k \times l)$, where $I(v) = \{1, \ldots, m\} \times \{1, \ldots, n\}$. Let the edge labeled (i, j) connect v and its descendant v_{ij} labeled $(A_{ij}, k_i \times l_j)$. Then $\sum_{i=1}^{m} k_i = k$, $\sum_{j=1}^{n} l_j = l$ and $N \to [A_{ij}]_{m,n}$ is a production in $\mathcal{P}$.*

If $S_0 \to \Lambda \in \mathcal{P}$ then the tree T_Λ with two vertices – the root labeled $(S_0, 0 \times 0)$ and the leaf labeled $(\Lambda, 0 \times 0)$ is a derivation tree for G too.

Let T be a derivation tree for a CF grammar $G = (V_N, V_T, S, \mathcal{P})$, V set of its vertices. We assign a picture to each vertex of T by defining a function $p : V \to {V_T}^{**}$: if $v \in V$ is a leaf labeled $(a, 1 \times 1)$ then $p(v) = a$ else $p(v) = \bigoplus [P_{ij}]_{m,n}$, where $I(v) = \{1, \ldots, m\} \times \{1, \ldots, n\}$, $P_{ij} = p(v_{ij})$, v_{ij} is a descendant of v connected by the edge labeled (i, j). $p(T)$ is defined as $p(r)$, where r is the root of T. $p(T_\Lambda) = \Lambda$. Observation: if $v \in V$ is labeled $(N, k \times l)$ then $rows(p(v)) = k$, $cols(p(v)) = l$.

Lemma 6 *Let $G = (V_N, V_T, S, \mathcal{P})$ be a CF grammar and $N \in V_N$.*

1. *Let T be a derivation tree for G having its root labeled $(N, k \times l)$. Then $p(T) \in \mathcal{L}(G, N)$.*
2. *Let O be a picture in $\mathcal{L}(G, N)$. There exists a derivation tree for G with root labeled $(N, k \times l)$ such that $rows(O) = k$, $cols(O) = l$ and $p(T) = O$.*

Proof: The lemma follows directly from the previous definitions. $\square$

Example 7 *Let us define the picture language L over $\Sigma = \{a, b\}$.*

$$L = \{O \mid O \in \{a,b\}^{**} \wedge \exists i, j \in \mathbf{N} : \ 1 < i < rows(O) \ \wedge \ 1 < j < cols(O) \ \wedge$$

$$\forall x \in \{1, \dots, rows(O)\}, y \in \{1, \dots, cols(O)\} : \ O(x,y) = a \Leftrightarrow x \neq i \ \wedge \ y \neq j\}$$

L is context-free. It is generated by the CF grammar $G = (V_N, \Sigma, S, \mathcal{P}, S)$, where $V_N = \{S, A, V, H, M\}$ and the set $\mathcal{P}$ consists of the following productions:

$$S \to \begin{matrix} A & V & A \\ H & b & H \\ A & V & A \end{matrix} \qquad A \to M \qquad A \to \ A \quad M \qquad M \to a \qquad M \to \begin{matrix} a \\ M \end{matrix}$$

$$V \to b \qquad V \to \begin{matrix} b \\ V \end{matrix} \qquad H \to b \qquad H \to \ b \quad H$$

The non-terminal A generates the language $\{a\}^{**} \backslash \{\Lambda\}$, M generates one-column pictures of a's, V generates one-column pictures of b's and finally H generates one-row pictures of b's.

Let us consider CF grammars with productions of the form $N \to a$, $N \to [A_{1j}]_{1,2}$ and $N \to [A_{i1}]_{2,1}$, where a is a terminal and A_{ij} are non-terminals. These grammars are presented in [7]. Let us denote them as $CFG2$. We proof that their generative power is less than the generative power of CFG's.

Theorem 8 *$\mathcal{L}(CFG2)$ is a proper subset of $\mathcal{L}(CFG)$.*

Proof: By a contradiction. Let $G = (V_N, V_T, S, \mathcal{P})$ be a $CFG2$ generating the language L in Example 7. Let us consider an integer $n \geq 3$. We denote the set of all square pictures of the size n in L as L_1. n can be chosen sufficiently large so that no picture in L_1 equals the right side of any production in $\mathcal{P}$. L_1 consists of $(n-2)^2$ pictures. At least $\lceil \frac{(n-2)^2}{|\mathcal{P}|} \rceil$ pictures are derived from S using the same production. Without loss of generality, let this production be $S \to A\,B$. If n is sufficiently large there exist two pictures with different indices of the row of b's (maximally $n-2$ pictures in L_1 can have the same index of the row of b's). Let us denote these pictures as O and $\overline{O}$. It holds $O = O_1 \Phi O_2$, $\overline{O} = \overline{O}_1 \Phi \overline{O}_2$, where $O_1, \overline{O}_1 \in \mathcal{L}(G, A)$ and $O_2, \overline{O}_2 \in \mathcal{L}(G, B)$. It implies $O = O_1 \Phi \overline{O}_2 \in \mathcal{L}(G)$. It is a contradiction, O contains b in the first and in the last column, but these b's are not in the same row. $\square$

Example 9 *Let us define the language L over the alphabet $\Sigma = \{0, 1, x\}$ consisting just of all pictures $O \in \Sigma^{**}$ satisfying:* 1) *O is a square picture of an odd size,* 2) *$O(i,j) = x \Leftrightarrow i, j$ are odd indexes,* 3) *if $O(i,j) = 1$ then the i-th row or the j-th column (at least one of them) consists of 1's*

Lemma 10 *L can be recognized by a $DFSA$.*

Proof: $DFSA$ automaton T recognizing L can be constructed as follows. T checks if an input picture is a square picture of an odd size. It can be done moving the head diagonally. The computation continues by scanning row by row and checking if symbols x are just in all fields with both indexes odd, in case of other fields containing 1 if the field and its four neighbours form one of the possible configurations as follows:

$$\begin{array}{ccc} & 1 & \\ x & 1 & x \\ & 1 & \end{array} \quad \begin{array}{ccc} & x & \\ 1 & 1 & 1 \\ & x & \end{array} \quad \begin{array}{ccc} & 1 & \\ 0 & 1 & 0 \\ & 1 & \end{array} \quad \begin{array}{ccc} & 0 & \\ 1 & 1 & 1 \\ & 0 & \end{array} \quad \begin{array}{ccc} & 1 & \\ 1 & 1 & 1 \\ & 1 & \end{array} \quad \begin{array}{ccc} & x & \\ \# & 1 & 1 \\ & x & \end{array}$$

$$\begin{array}{ccc} & \# & \\ x & 1 & x \\ & 1 & \end{array} \quad \begin{array}{ccc} & x & \\ 1 & 1 & \# \\ & x & \end{array} \quad \begin{array}{ccc} & 1 & \\ x & 1 & x \\ & \# & \end{array}$$

□

Theorem 11 *$\mathcal{L}(DFSA)$ is not a subset of $\mathcal{L}(CFG)$.*

Proof: By contradiction, let $G = (V_N, V_T, S, \mathcal{P})$ be a CFG such that $\mathcal{L}(G) = L$, where L is the language in Example 9. Without loss of generality, $\mathcal{P}$ does not contain any production of the form $A \rightarrow B$, where A, B are non-terminals. We take an odd integer $n = 2 \cdot k + 1$. Let L_1 be the set of all pictures in L of the size n. n is chosen sufficiently large so that no picture in L_1 equals the right side of any production. We have $|L_1| = 2^k \cdot 2^k = 2^{n-1}$ (there are k columns and k rows, for each we can choose if the row, resp. column consists of 1's or not). There are at least $\frac{2^{n-1}}{|P|}$ pictures in L_1 that are derived from S using the same production. Let it be the production $S \rightarrow [A_{ij}]_{p,q}$. Let the set of the given pictures be L_2. Without loss of generality, we assume that $p \geq q$. In addition, $p \geq 2$ (otherwise the production is of the form $A \rightarrow B$).

The goal is to show that there are two pictures $U, V \in L_2$ such that $U = \bigoplus[U_{ij}]_{p,q}$, $V = \bigoplus[V_{ij}]_{p,q}$, $U_{ij}, V_{ij} \in \mathcal{L}(G, A_{ij})$ (property (1)) and next that the first row of U does not equal the first row of V (property (2) – in other words, it means U and V differ in one of the columns with respect to the symbols 1). The number of all possible sequences

$$cols(U_{1,1}), cols(U_{1,2}), \ldots, cols(U_{1q}), rows(U_{1,1}), rows(U_{2,1}), \ldots, rows(U_{p1})$$

is bounded by n^{p+q}. There exists a set $L_3 \subseteq L_2$, $|L_3| \geq \frac{2^{n-1}}{|\mathcal{P}| \cdot n^{p+q}}$ and each pair of pictures in L_3 has the property (1). L_2 contains a subset of $2^k = 2^{\frac{n-1}{2}}$ pictures, where each pair satisfies the property (2). It implies that the pair U, V exists in L_3 for some sufficiently large n. If we replace the sub-pictures $U_{1,1}, \dots, U_{1q}$ in U by the sub-pictures $V_{1,1}, \dots, V_{1q}$ (U_{1i} replaced by V_{1i}) we get the picture O that is in L again. But it is a contradiction, because O does not have all properties of pictures in L. $\Box$

Note that $DFSA$ cannot recognize, for example, the CF language $L = \{a^n b^n \mid n \in \mathbf{N}\}$ containing one-row pictures only. It means $\mathcal{L}(DFSA)$ and $\mathcal{L}(CFG)$ are incomparable.

5. Two-dimensional Forgetting Automata

Forgetting automata are bounded Turing machines that can rewrite the content of a field by the special symbol @ only (we say, they erase it). It is possible to characterize (one-dimensional) context-free languages using forgetting automata as it is shown in [5]. We extend some of these ideas – we show that two-dimensional context-free languages can be recognized by two-dimensional forgetting automata. The proof is strongly based on a technique of storing information in blocks that has been presented in [1], where relations between two-dimensional $NFSA$ and two-dimensional forgetting automata are studied.

Definition 12 Two-dimensional forgetting automaton *(NFA) is a two-dimensional bounded Turing machine* $(Q, \Sigma, \Sigma_0, q_0, \delta, Q_F)$*, where* $\Sigma = \Sigma_0 \cup \{\#, @\}$. $@ \notin \Sigma_0$ *is a special symbol called the erase symbol. In addition, if* $(a, q) \to (\overline{a}, \overline{q}, d)$ *is an element of transition relation given by* δ*, then* $a = \overline{a}$ *or* $\overline{a} = @$.

First of all, we sketch an idea of how a deterministic forgetting automaton (DFA) can store and retrieve information by erasing some symbols on the tape so that the entry picture can be still reconstructed (we follow the description presented in [1]).

Let $A = (Q, \Sigma, \Sigma_0, q_0, \delta, Q_F)$ be a DFA. Let $\Sigma_0 = \Sigma \setminus \{@, \#\}$. A performs a computation on a picture O. Let M be the set of tape fields that O contains. Let $O(f)$ denote a symbol contained in the field $f \in M$. Then for each $G \subseteq M$ there is $s \in \Sigma_0$ such that $|\{f \in G, O(f) = s\}| \geq \frac{|G|}{|\Sigma_0|}$. Let the automaton A erase fields of G containing the symbol s only. Each such field can therefore store 1 bit of information: the field is either erased or not erased. It is thus ensured that G can hold at least $\frac{|G|}{|\Sigma_0|}$ bits of information. Furthermore, the original symbol of all erased fields in G is known – it is s.

Let us consider M to be splitted into rectangular blocks of the size $k \times l$, where $n \leq k < 2n$, $n \leq l < 2n$ for some n. The minimum value for n will be determined in the following paragraphs. (In the case of just one dimension of the picture being lower than n, the blocks will be only as high – or wide – as the picture. In the case of both dimensions of the picture being lower than n, an automaton processing such a picture can decide whether to accept it or reject it on the basis of enumeration of finitely many cases.)

If both width and height of M are at least n, all blocks contain $n \times n$ fields, except for the blocks neighbouring with the lower boundary of M, which can be higher, and the blocks neighbouring with the right boundary of M, which can be wider. Nevertheless, both dimensions of each block are at most $2n - 1$.

Each block $B_i \subseteq M$ is divided into two parts – F_i and G_i. F_i consists of the first $|\Sigma_0|$ fields of B_i. We can choose the size of the blocks arbitrarily, so a block will always contain at least $|\Sigma_0|$ fields. G_i contains the remaining fields of B_i. Let $s_r \in \Sigma_0$ be a symbol for which $|\{f \in G_i, O(f) = s_r\}| \geq \frac{|G_i|}{|\Sigma_0|}$. The role of F_i is to store s_r: if s_r is the r-th symbol of Σ_0 then A stores it by erasing the r-th field of F_i. Now A is able to determine s_r, but it needs to store somewhere the information about the symbol originally stored in the erased fields in F_i. A uses the first $|\Sigma_0|$ bits of information that can be stored in G_i. If the erased symbol in F_i was the q-th symbol of Σ_0 then the q-th occurrence of s_r in G_i is erased, allowing A to determine the erased symbol in F_i. This way a maximum of $|\Sigma_0|$ bits of available information storable in G_i will be lost. For any block B_i containing m fields this method allows A to store at least $\frac{m-|\Sigma_0|}{|\Sigma_0|} - |\Sigma_0|$ bits of information in B_i.

In the following text, a region is every rectangular sub-array of tape fields. We can consider such a region to be the picture as well. $\bigoplus[R_{ij}]_{m,n}$ (if defined) is used to denote the region that is the union of R_{ij}'s, where indexes of rows, resp. columns in R_{ij} are less than indexes of rows in $R_{i+1,j}$, resp. columns in $R_{i,j+1}$.

Theorem 13 $\mathcal{L}(CFG) \subset \mathcal{L}(NFA)$

Proof: Let us consider a context-free grammar $G = (V_N, V_T, S_0, \mathcal{P})$, $\mathcal{L}(G) = L$. We describe how to construct a forgetting automaton A that recognizes L. We define Σ as $V_T \cup \{\#, @\}$. Let O be an input picture. The idea of the computation of A is to try to construct a derivation tree T for G such that its root is labeled $(S_0, rows(O) \times cols(O))$ and $p(T) = O$. During the computation, O (more precisely, the region containing the input) will be split into disjunct regions, each labeled by an element of $V_T \cup V_N$. We distinguish two kinds of regions: Regions consisting of one field (t-regions) – each labeled by a terminal given by the original content of the field, and regions consisting of more than one field (N-regions) – labeled by a non-terminal. Some of the possible regions are *derived.* A derived region is *represented* if there is information determining its position, size and label stored on the tape. We explain later how A derives and represents regions. We consider a bijection between derived regions and vertices of T. Let $m(R)$ denote the vertex corresponding to a region R and $m^{-1}(v)$ the region corresponding to a vertex v.

At the beginning stage, we consider O to be split into $rows(O) \cdot cols(O)$ t-regions. Each region is derived, represented and corresponds to a leaf of T. These regions are the only derived regions at the beginning. A works in cycles. A cycle includes steps. In a step, A derives a new region. Roughly said, it non-deterministically chooses a (not derived yet) region $R = \bigoplus[R_{ij}]_{s,t}$, where R_{ij} are represented regions or regions derived in the current cycle, R_{ij} labeled A_{ij}, and a production $N \rightarrow [A_{ij}]_{s,t} \in \mathcal{P}$ (if such a production does not exists, the computational branch does no accept). R is derived and labeled N. As for the tree T, $m(R)$ is labeled $(N, cols(R) \times rows(R))$ and $p(v(R)) = R$, $m(R_{ij})$ is a descendant of $m(R)$, the edge connecting the vertices is labeled (i, j). A uses the technique of storing information in blocks. We consider O to be divided into rectangular blocks B_i such that $n \leq rows(B_i), cols(B_i) < 2n$, where n is a constant that we derive later. We assume $rows(O), cols(O) \geq n$. The other inputs will be discussed as a special case. Let us describe how to represent regions during the computation. A does not represent any N-region that is a subset of a block – we denote this requirement as (1). We distinguish two types of the remaining N-regions. Let us consider a block B of the size $k \times l$ and a N-region R. Let us denote the four fields neighbouring with the corners of B as the C-fields of B (see Fig. 4). We say that B is a *border* block of R iff $R \cap B \neq \emptyset$ and R does not contain all 4 C-fields of B. A represents a region in its border blocks.

We consider the bits available to store information in B to be organized into groups. Each group has a *usage flag* consisting of two bits determining if the group is *not used* (information has not been stored in the group yet), *used* (stored information is current) or *deleted* (stored information is not relevant anymore). The first state is indicated by two non-erased symbols, the second one by one symbol erased and finally the third one by two erased symbols. One group of bits represents the intersection between a region and a block. Information in a group includes coordinates in the block (one coordinate requires $\lfloor log(2 \cdot n) \rfloor$ bits), labels (a non-terminal is represented unary using $|V_N|$ bits) and various "flags" that we describe in the following paragraphs.

Let B be a border block of R. We say that the intersection between R and B is of the *first* type if R contains one or two C-fields of B and of the *second* type

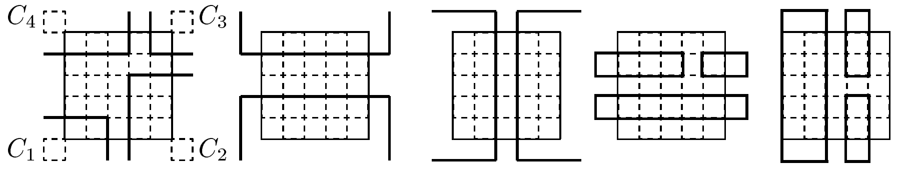


Figure 4: A block and its C-cells; eight types of intersection between the block and a N_1-region; the horizontal and vertical types of N_2-regions.

if R does not contain any C-field. It is obvious that if R has the intersection of the first, resp. second type with a border block then it has the intersection of the same type with all its border blocks. It means we can denote every N-region having the intersection of the first, resp. second type with its border blocks as N_1-region, resp. N_2-region.

There can be 8 (Fig. 4) different types of the intersection between a N_1-region R and a block B with respect to which C-fields of B are included in R. It means the intersection can be represented using 3 bits determining the type and one or two coordinates. Let us solve the question how many different intersections with N_1-regions A need to represent during the computation in B. B can be a border block of 4 different represented N_1-regions after performing a sequence of cycles. The border (coordinates in B) of one N_1-region can be changed maximally $k+l-2$ times (before it completely leaves B), because every change increases, resp. decreases at least one of the coordinates. It means it is sufficient if B can represent $8 \cdot n$ intersections. Note that more than $8 \cdot n$ N_1-regions having the non-empty intersection with the border block B can be represented, however, some of them have the same intersection with B. In addition, if A knows coordinates of R in B it can determine which group of bits represents R in neighbour blocks. The label of R is represented in one of its border blocks – the correspondent usage flag is of the value "used". If two labels are reserved in each group of bits then there is always at least one not used label in a border block of a new represented region.

We consider N_2-regions to be vertical or horizontal (Fig. 4). Let R be a horizontal, reps. vertical N_2-region. There are three types of intersection between R and B, thus A represents the intersection using tree bits determining if the region is vertical or horizontal and the type of the intersection, two coordinates of the first and the last row of R and, in addition, twice two coordinates with the usage flag that are reserved to represent positions of the leftmost and the rightmost column, resp. the first and the last row of R in correspondent border blocks eventually (the same idea of the representation as in the case of labels). It holds that there is just one border block of R, where one pair of coordinates is marked as used. Let the width of R be $min(cols(R), rows(R))$. We add one more requirement, denoted (2), on the representation of regions: If R is a represented N_2-region then there is not any different represented N_2-region R' of the same width having the same border blocks. Under this assumption, A needs to represent maximally $2 \cdot 4 \cdot 2 \cdot n = 16 \cdot n$ intersections of the second type in a block during the computation (the number of N_2-regions of the width 1 is bounded by $4 \cdot max(k,l) \leq 4 \cdot 2 \cdot n$, a N_2-region of a greater width is created as the concatenation of several N_2-regions of less widths, every concatenation decreases the number of represented N_2-region at least by 1).

We complete and summarize what information should be stored in a block during the computation. One bit determines if B is a subset of some derived N-region or not – this information is changed during the computation one time exactly. According to this bit, A determines if a field of a block that is not a border block of any N-region is t-region or not. $8 \cdot n$ groups of bits are reserved to represent intersections of the first type. Each group consists of the usage flag, 3 bits determining the type of intersection, two coordinates and two labels, each with the usage flag. Similarly, $16 \cdot n$ groups of bits are reserved to represent intersections of the second type. These groups contain one additional information – twice two coordinates and the usage flag. It means we need $O(n \cdot log(n))$ bits per block, while a block can store $\Omega(n^2)$ bits. It implies that there exists a suitable constant n.

We can describe cycles and steps in more details now. Let d be the maximal number of elements of the right side of productions in $\mathcal{P}$. In a cycle, A non-deterministically chooses a non-represented region R consisting of the set of regions $\mathcal{R} = \{R_1, \ldots, R_s\}$ that are all represented, and a sequence of productions $P_1, \ldots, P_t$, where $s, t \leq d \cdot 4n^2 + 2 \cdot 4n^2$. A chooses R as follows. It starts with its head placed in the upper left corner of O. It scans row by row from left to right, proceeding from top to bottom and non-deterministically chooses the upper left corner of R (it has to be the upper left corner of an already represented region). Once the corner is chosen, T moves its head to the right and chooses the upper right corner. While moving, when T detects a region R_i first time, it scans its borders and remembers in states the neighbour regions of R_i including their order and the label of R_i as well. When the upper right corner is "claimed", A continues by scanning next rows of R until it chooses the last one. Every time T enters a new represented region (not visited yet), it detects its neighbours. Thanks to mapping of neighbouring relation among R_i's, A is able to move its head from one region to any other desired "mapped" region.

A continues by deriving regions. The first region is derived according to P_1. A chooses $S_1 = \bigoplus[S_{ij}]_{s_1,t_1}$, where each S_{ij} is one one of R_{ij}'s and checks if S_1 can be derived. Let us consider all S_{ij}'s to be deleted from $\mathcal{R}$ and S_1 to be added. A performs the second step on the modified set $\mathcal{R}$ using P_2, etc. In the last step A derives R. All these derivations are performed in states of A only. After that, A records changes on the tape. If the region corresponding to O labeled S_0 is derived then T has been constructed and $O \in L$. On the other hand, let T be a derivation tree for G having its root labeled $(S_0, rows(O) \times cols(O))$ and $p(T) = O$. A can construct T despite the requirements (1) and (2). If R is a region derived using some regions that are subsets of blocks, A derives such regions in one cycle. It requires to choose $d.4n^2$ regions maximally. If R and R' are N_2-regions to be derived having the same border blocks and $R \subseteq R'$, A derives both in one cycle and represents R'. Note that $|R' \setminus R| \leq 2.4n^2$, thus A can choose all needed regions to derive R' as well.

Let us discuss the remaining special case when one of the sizes (e.g. $cols(O) = m$) of O is less than n (if both sizes are less than n then A scans all symbols and accepts or rejects O immediately). In this case, A needs to represent horizontal N_2-regions in a block only. In addition, maximally $4.m$ different intersections have to be represented in a block (estimated similarly as in the previous case). $O(m \cdot log(n))$ bits are required, while $\Omega(m \cdot n)$ bits can be stored, thus a suitable constant n exists again. □

6. Conclusions

We proved that $\mathcal{L}(CFG)$ does not include all languages in $\mathcal{L}(DFSA)$. It indicates that the presented class is not probably a suitable candidate for the class of "real" two-dimensional context-free languages. However, this is the problem with all other proposals as well. In our opinion, the class $\mathcal{L}(CFG)$ deserves an attention, because it is based on a natural generalization of CF grammars. We showed how forgetting automata can recognize languages in $\mathcal{L}(CFG)$. Thus the question arises whether forgetting automata restricted in some way can characterize it exactly.

References

[1] P.Jiřička, V.Král: Deterministic Forgetting Planar Automata, in proceedings of the 4-th Int. Conference: Developments in Language Theory, Aachen, Germany, 1999.

[2] D.Giammarresi, A.Restivo: Two-dimensional Languages, In A.Salomaa and G.Rozenberg, editors, Handbook of Formal Languages, volume 3, Beyond Words, pages 215–267. Springer-Verlag, Berlin, 1997.

[3] J.Hopcroft, J.Ullman: Formal Languages and Their Relation to Automata, Addison-Wesley, 1969.

[4] A.Rosenfeld: Picture Languages - Formal Models of Picture Recognition, Academic Press, New York, 1979.

[5] P.Jančar, F.Mráz, M.Plátek: Characterization of Context-Free Languages by Erasing Automata, Proceedings of MFCS'92, LNCS 629, Springer, 1992, pp. 305–314.

[6] P.Jančar, F.Mráz, M.Plátek: Forgetting automata and context-free languages, Acta Informatica 33, Springer-Verlag, 1996, pp. 409–420.

[7] M.I.Schlesinger, V.Hlaváč: Deset Kapitol z Teorie Statistickeho a Strukturniho Rozpoznavani, CVUT, Prague, 1999, (in Czech).

[8] P.Jiřička: Grammars and automata with two dimensional lists (grids). Master thesis, Faculty of Mathematics and Physics, Charles U., Prague, 1997, (in Czech).

Data Mining and Neural Networks[3]

Iveta Mrázová

Department of Software Engineering, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic

František Mráz

Department of Software and Computer Science Education, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract: Especially the emerging technologies of artificial neural networks, fuzzy logic and evolutionary programming provide essential tools for designing intelligent data mining systems. In this article, we present a brief summary of various data mining techniques with the emphasis on techniques based on artificial neural networks. In general, it is relatively complicated to explain and to visualize what and why the neural networks are doing. From this perspective, we will compare several recent models of neural networks which could help us to extract and to interpret a set of clear and simple rules providing insight into how is a particular model working.

1. Introduction

Recently, data base technologies are the well-established means for storing and exploring large volumes of data. Statistics and many techniques from artificial intelligence represent time proven tools for modeling the data and analyzing their mutual dependencies. For many years, these techniques were developed independently one on the other. But due to a number of reasons that occurred in 1990s they were put together in data mining. The main factors were the production and warehousing of large amounts of data, available computing power and a high competitive pressure requiring optimization of decision-making based on automated methods for learning from the past.

[3]This research was supported by the grant No. 157/1999/A INF/MFF of the GA UK and by the grant No. 201/99/0236 of the GA ČR.

Data mining means in principle exploring and analyzing large quantities of data with the aim to discover their mutual relationships. Many data mining techniques started few years or decades ago, mostly in the field of artificial intelligence. But their widespread availability and acceptance is only just beginning. Data mining algorithms typically require multiple passes over large volumes of data and many of them are computationally intensive. Anyway, several trends are increasing the necessity of powerful data mining tools - an increasingly service-based economy, the advent of mass customization and the competitive importance of information.

Data mining is an interactive and iterative process consisting of several steps: selection of data, its pre-processing, transformation, adequate "mining" and interpretation of obtained results. The area of data mining brings together ideas and techniques from a variety of fields - economics, artificial intelligence (AI), databases and statistics. To the tasks well suited for data mining belong classification, estimation, prediction, affinity grouping, clustering and description. The goal of a data mining task might be e.g. to allow a corporation to improve its marketing, sales, and customer support operations through better understanding of its customers. Further, data mining techniques can help e.g. to spot the most valuable customer but also to pick out those that should be e.g. turned down for a loan.

Data mining proceeds following one - or both - basic styles:
a) Hypothesis testing, is a top-down approach that attempts to verify or disprove preconceived ideas concerning relationships present in the data.
b) Knowledge extraction, is a bottom-up approach that starts in the data and tries to "discover" something new that we did not know before. There are known two basic approaches to knowledge extraction - directed and undirected. The directed one is characterized by the presence of a target field whose value is to be predicted. The undirected one attempts to find patterns or similarities among groups of data records without the knowledge of particular target fields.

In the following text, we will concentrate on knowledge extraction. The most common data mining techniques applicable for this purpose are: cluster detection, memory-based reasoning, market basket analysis, decision trees, link analysis, artificial neural networks (ANN), fuzzy logic, support vector machines, and genetic algorithms. First, we will briefly state the tasks which can be solved by means of data mining. Next, we will characterize the above-mentioned data mining techniques. In Section 4, we will focus on the so-called BP-networks already used in data mining. Further, we will discus various directions for using some recent models of ANN which, at least from our point of view, could have better knowledge extraction abilities than standard models of ANNs.

2. Data mining tasks

In practice, the most important data mining tasks include: classification, estimation, prediction, affinity grouping, clustering and description. *Classification* consists of examining the features of a newly presented object and assigning it to one of a predefined set of classes. The classification is characterized by a well-defined definition of a limited number of the classes, and a training set consisting of pre-classified examples. The objective is to build a model that can be applied to unclassified data in order to classify it, i.e. to assign the data to the one or other class. Decision trees and memory-based reasoning are techniques well suited to classification. Link analysis is also useful for classification under certain circumstances.

While classification deals with discrete outputs, *estimation* deals with continuously valued outputs. In practice, estimation is often used to perform a classification task. E.g. instead of classifying objects into two classes, we can estimate the probability that the given object belongs to the first class. Now, the classification task now comes down to establishing a threshold score. Objects with a score greater than or equal to the threshold are classified as belonging to the first class and objects with a lower score are considered to be from the second class. Neural networks are well suited to estimation tasks.

The nature of *prediction* is the same as in the case of classification or estimation except that the records are classified according to some predicted future behavior or estimated future value. In a prediction task, the only way to check the accuracy of the classification is to wait and see. Anyone of the techniques used for classification and estimation can be adapted for use in prediction by using historical data for training examples such that the value of the variable to be predicted will be already known. When this model is applied to current inputs, the result is a prediction of future behavior.

Affinity grouping helps to determine which things go together. A prototypical example is to determine what things go together in a shopping cart in a supermarket. For this reason, the techniques used for affinity grouping are often called market basket analysis. *Clustering* is the task of segmenting a heterogeneous population of data into a number of more homogeneous subgroups or clusters. Compared with classification, clustering does not rely on predefined classes - there are no predefined classes and no examples with predicted or estimated future values. The data records are grouped together on the basis of self-similarity. It is up to the user to determine what meaning, if any, to attach to the resulting clusters. Clustering is often performed as a kind of prelude to some other form of data mining or modeling.

The purpose of *description* is to describe what is going on in a complicated database. Its main goal is to increase the understanding of the people, products, or processes that produced the data in the processed database. An adequate description of a behavior often suggests also its explanation. Some of the data mining tools, e.g. the market basket analysis, are purely descriptive. Others, like neural networks in general, provide very little with regard to nothing description.

3. Data mining techniques

There can be found many approaches in the literature to solve the above-sketched data mining tasks. In this section, we provide a brief overview of the main data mining techniques. Most of them are treated in [5], for the rest of them we give explicit references. *Cluster Detection* is inherently an undirected data mining technique. The goal of cluster detection consists in finding previously unknown similarities in the data with techniques including geometric methods, statistical methods, and neural networks. Cluster detection provides a very good starting point for any further analysis of the data. *Market Basket Analysis* is a form of clustering used for finding groups of items that tend to occur together in a transaction (or market basket). The resulting information can be used e.g. for planning store layouts, limiting specials to one of the products in a set that tend to occur together.

Memory-Based Reasoning (MBR) is a directed data mining technique that uses known instances as model to make predictions about unknown instances. MBR looks for the nearest neighbors in the given data set and combines their values to assign the classification or prediction values to unknown patterns. The distance to the neighbors gives a measure of correctness of the results. A major advantage of MBR represents its ability to learn new classifications merely by introducing their new instances into the database. For instance, if we want to determine whether a new customer warrants to pay off installments, we would find similar customers - neighbors - in the data set and make the "give-a-credit", "investigate-further" or "refuse-immediately" decision based on the status of the neighbors.

A powerful model used for classification represent also the so-called *Decision Trees.* They divide the patterns in the training set into pairwise disjoint subsets, each of which is described by a simple rule on one or more pattern features. This allows evaluating the results, identifying their key characteristics. *Link Analysis* is an application of graph theory constructs to data mining. It follows relationships between the particular data patterns. Relationships between customers are becoming increasingly important, especially as marketing groups focus more on customers, households, and economic marketing units instead of specific accounts. The few tools available focus on visualizing the links, rather than analyzing the patterns.

Artificial Neural Networks ([3], [7]) represent probably the most common data mining technique, perhaps a synonym with data mining to some people from the area of economics. These challenging models being originally inspired by their biological counterparts are still exploited for mathematical description of brain functioning, and, at a higher level, for modeling of mental processes in cognitive computing. On the other hand of this research effort there are engineers who adopt these neural models for solving important practical tasks in AI-applications where standard analytical solutions are not always adequate and/or encounter difficulties.

Neural networks learn in general from a training set, generalizing the knowledge incorporated somehow inside it in order to classify or predict future data. Neural networks can also be applied to undirected data mining (in the form of self-organizing feature maps and related structures) and time-series prediction. To the most often used neural network paradigms belong the so-called feed-forward neural networks of the back-propagation type (BP-networks; [13]) based on supervised training and the so-called self-organizing feature maps (SOFMs; [8]) using self-organization for their adjustment.

Neural networks have two major drawbacks. The first refers to poor understanding of models they produce. But in many applications an interpretation of the network function is necessary or at least desirable. In such a case, methods of *Fuzzy Logic* can be used. Fuzzy logic ([4]) can be conceptualized as a generalization of classical logic. Modern fuzzy logic was developed by Lofti Zadeh in the mid-1960s to model those problems in which imprecise data must be used or in which the rules of inference are formulated in a very general way making use of diffuse categories. The second drawback of neural networks consists in their sometimes high sensitivity to the form of incoming data. Different data transformations can lead to different results; therefore, setting up the data is a significant part of their successful application.

Support Vector Machines (SVM) introduced by Vapnik and co-workers are learning systems that use linear discriminant functions in a transformed high dimensional feature space. This paradigm employs the so-called supporting vectors, which are patterns situated most closely to the separating hyperplane. In the few years since its introduction, this method has already outperformed many other systems in a wide variety of applications ([6]). SVM can be applied to similar tasks like neural networks.

Genetic Algorithms ([6]) employ genetics and natural selection to a search for optimal sets of parameters that correspond to a predictive function. By means of selection, crossover, and mutation operators they evolve successive generations of solutions. As the generations evolve, only the most predictive individuals should survive, until the fitness function converges to an optimal solution. Genetic algorithms have also been used to enhance other data mining models, like e.g. MBR and neural networks ([14], [15]).

4. Neural networks in data mining

This section is focused on BP-networks and some of their numerous modifications designed with the aim to improve their characteristics inevitable for data mining. An artificial neural network consists of basic units called neurons. Each neuron has several inputs (with associated weights) that it combines into a single output value. The neurons can be mutually inter-connected such that the outputs of some neurons can be used as inputs of other neurons. In a feed-forward neural network, the neurons are grouped together in a sequence of layers. Each neuron in the first layer (called the input layer) is connected to exactly one input. In the following layers, each neuron receives as inputs the outputs of all the neurons in the preceding layer. The last layer (called the output layer) is connected to the output of the network. All the layers between the input layer and the output layer are called hidden layers.

In the standard BP-network, each neuron calculates its output by multiplying the value on each input by its corresponding weight, summing these up, and applying the sigmoidal transfer function to the computed sum. A neural network can have any number of hidden layers, but for most practical tasks, one or two hidden layers are sufficient. The wider the layer the greater the capacity of the network to recognize patterns. Anyway, for too large layers, the neural network can recognize patterns-of-one by memorizing each of the training examples (from the training set). Therefore, this problem is sometimes denoted as "overtraining". Obviously, we want the networks to generalize the training set, not to memorize it.

The aim of the standard Back-Propagation training algorithm was to find a set of weights that ensures that for each input vector contained in the training set the actual output vector produced by the network equals the desired output vector. This requirement has the form of the so-called objective function. Actual or desired states of hidden neurons are not specified by the task and the learning procedure has to decide under what circumstances the hidden neurons should be active in order to help achieve the desired input-output behaviour. The Back-Propagation training algorithm involves a forward pass through the network to estimate the error at the output layer, and then a backward pass to determine also the error at the hidden neurons. Then, the synaptic weights are modified with the aim to decrease the error at the output layer and to achieve the desired input-output behaviour.

More details can be found e.g. in [3]. Solving practical tasks, there are often trained many networks with different parameters like learning step, initial weight values, architecture etc. Afterwards, the network with the best wanted properties is selected and actually applied in the solution. The standard BP-training algorithm has a lot of advantages. However, we have also to deal with several limitations:

a) Relatively slow convergence of the standard BP-training algorithm - moreover, sometimes the learning process does not end with a network working satisfactorily even on the training set, or the training results in an "overtrained" network.

b) An almost "unclear" inner network structure with poor generalization - the training algorithm neither specifies explicitly, what hidden neuron outputs are appropriate for processing an input pattern in a proper way, nor specifies the number of hidden neurons. Moreover, it is extremely difficult to "guess" the real meaning of every particular hidden neuron for a proper network output.

c) Poor robustness of trained networks - intuitively, for an input patterns with only a small deviation from a certain training patterns the network should give approximately the same output. However, this can be a difficult problem especially for those input patterns lying in the border area between pattern clusters.

d) Poor reusability of already trained networks in the solution of other tasks - even "slight" modifications of the given task (represented e.g. by another dimension of the particular input/output vector) or by a modification of the training set (e.g. by adding and/or removing some training patterns) may often lead to a renewed training process.

The standard BP-training algorithm has relatively good approximating and classification properties and using it we can often form an internal structure that seems to be appropriate for a particular task domain. The outputs of all neurons in a hidden layer computed for a given input pattern is called internal representation of the input pattern. In [10], we proposed a training algorithm for forcing the so-called condensed internal representation. The aim of this algorithm is to group the outputs of the hidden neurons around three possible values - 1, 0 and 0.5. In the AI-terminology, this corresponds to creating rules of an expert system, where active neuron states indicate "yes", passive states "no" and the so-called silent states stand for those cases where "no decision is possible".

In order to prevent the network from forming very similar internal representations for different output patterns already during the training process, the new objective function will consist of the standard BP-error function enlarged by the representation error function and an ambiguity criterion ([11]). The internal knowledge representation is forced to form rather in the hidden neurons than exclusively by means of large weight values typical for "overtrained" networks. In this way, forcing the unambiguous condensed internal representation can help to overcome the danger of "overlearning". Such a "transparent" internal representation seems to be very important for the design of the optimal network architecture.

With regard to the reusability of already trained networks, the elimination of unnecessary hidden and input neurons seems to be an important aspect. In many cases, a reduced number of hidden neurons improves generalization capabilities of the trained network at the expense of its accuracy. On the other hand, an insufficient number of hidden neurons may result in an inadequate capability to separate the concerned pattern clusters. There are known many attempts to determine the right number of hidden neurons - almost all based on the empirical experience. However, a satisfactory general rule for the selection of the right number of hidden neurons has not been developed yet.

Some approaches initialize the training process with more hidden neurons than needed. Later on, some neurons may be discovered to be unnecessary and can be removed from the network. Those hidden neurons yielding a redundant information are often characterized by constant output values for all input patterns from a given set or by the same or "opposite" output values than those of another neuron from the same layer - and this for all considered input patterns. A BP-network having no such redundant neurons is said to be reduced. Further, it can be shown that for every BP-network and a set of input patterns there exists a reduced BP-network yielding the same output vector as the original BP-network. Alternative approaches based on incremental approximation start the training process with a small number of hidden neurons (maybe 1). During training, new hidden neurons are added and their weights are updated ([1], [2]).

Solving practical tasks, there are usually trained many networks with different parameters like learning step, initial weights, architecture etc. But training large BP-networks with a complicated structure can be difficult and time-consuming. Moreover, even "slight" modifications of the given task represented for example by another dimension of the particular input/output vector or by a modification of the training set (given e.g. by adding and/or removing some training patterns) may often lead to a renewed training process.

In such a case, although having fast training algorithms, it would be more efficient to extract or to get a kind of a "module library" and to "incorporate" these modules into the solutions of other and possibly even more complex tasks ([11]). Such smaller parts of the entire neural network can be treated easier (e.g. with respect to the desired internal representation or robustness). In this context, especially the recognition of superfluous pattern features and pruning of unnecessary hidden and/or input neurons seems to play an important role in solving practical problems in data mining.

For the standard BP-training algorithm, there are usually no restrictions concerning the number of hidden or even inevitable input neurons needed for a "correct solution" of a given task. Therefore, it is very difficult to "guess" the really meaning of every particular hidden neuron for a proper network output. Good training accuracy can be achieved by forming very complex decision boundaries, which in turn requires a large network size. However, a good training accuracy does not necessarily guarantee a satisfactory robustness and/or generalization capabilities of the trained network. Actually, for a trained network we can find an "ε-equivalent" network with better robustness ([11]). Outputs of the ε-equivalent network can differ at most by a fixed (arbitrarily small) ε from the outputs of the original network.

5. Conclusions and further research

Another very important issue in data mining is represented by the ability to detect significant (novel) input patterns and to identify their characteristic features. This property could be used both for training and optimizing the structure of the data mining model at hand and for improving its characteristics - generalization abilities, robustness, storage capacity, etc. A suitable means for the detection of significant input patterns and features seems to be the internal knowledge representation. Most probably, the internal representation of significant input patterns will differ "somehow" from the known and "not too exceptional" ones. This could help to gain insight into the structure of the processed data, or to detect irregularities and errors in it.

Therefore, our further research will be focused on the mechanisms of creating and visualizing an appropriate internal knowledge representation in modular and hybrid data mining systems. The extracted knowledge will be applied for finding significant input patterns and their characteristics with a stressed attention to selected economical problems. In this framework, we plan to integrate various techniques for designing intelligent data mining systems. The models under consideration include the so-called generalized relief error networks (GREN-networks; [12], [13]) acting like an advisor for other models based on error-correction learning (usually BP-networks). The major advantages of GREN-networks consist in that they do not require "well-balanced" training sets for training other networks. Further, they are able (to a certain extent) to "simulate" the evolution of presented input patterns and evaluate to corresponding error.

The detection of significant input patterns and pattern features already before or during the training process can save computational time and means. The reason for this expectation consists in the same observation like e.g. in SVMs - that the most important training patterns lie close to separating hypersurfaces (on the border of the respective pattern clusters). Anyway, other patterns should be considered for training too - however rather according to the extent of their current significance, instead of simply omitting them. We expect that such a strategy could speed up the training process significantly. Further, we hope that the identification of the most significant pattern features could reduce drastically the number of necessary input neurons and hence lead to a simpler architecture of the chosen model as well.

On the other hand, GREN-networks still lack an illustrative kind of internal representation of "correct" and "suspicious" input patterns and their mutual topological relationship in the underlying feature space present e.g. in SOFMs or in Fuzzy ART-Maps. A transparent and visualized representation of significant pattern features, patterns and their mutual relationships among the data set could namely indicate e.g. "suspicious" positive outliers. Such input patterns lying far from separating hypersurfaces might satisfy the prescribed criteria but at the same time they can be situated far from the respective pattern cluster. This could indicate a "suspicious" input pattern but also an extremely good customer.

If we would be able to visualize the topological relationship among the data, we could e.g. tailor our loan products for each respective customer. Such a kind of financial engineering would allow to suggest and to adjust the proposed financial products such that they would better suit to the intended group of customers. Moreover, we expect that the application of GREN-networks would allow (at least to a certain extent) to predict the "evolution" of considered customers and to "identify" their most significant features that should or could be improved together with the amount of the suggested change.

References

[1] G. Andrejková: Application of the Approximation Theory by Neural Networks, in: Neural Network World, No. 5, Vol. 10, 2000, pp. 787-795.

[2] G. Andrejková: Incremental Approximation by Layer Neural Networks. Proceedings ISCI'2000, Physica-Verlag, (2000), pp. 15-20.

[3] M. Anthony, P. L. Bartlett: Neural Network Learning: Theoretical Foundations, (1999), Cambridge University Press.

[4] C. H. Chen (Ed.): Fuzzy Logic and Neural Network Handbook, (1996), McGraw-Hill, New York.

[5] M. J. A. Berry, G. Linoff: Data Mining Techniques For Marketing, Sales, and Customer Support, (1997), John Wiley & Sons, New York.

[6] N. Christianin, J. Shawe-Taylor: An Introduction to Support Vector Machines and other kernel-based learning methods, (2000), Cambridge University Press.

[7] Z. Michalewicz: Genetic Algorithms + Data Structures = Evolution Programs, (1999), Springer-Verlag, Berlin.

[8] S. Haykin: Neural Networks: A Comprehensive Foundation, (1999), Prentice Hall, Upper Saddle River, N. J.

[9] T. Kohonen: Self-Organizing Maps, (1995), Springer-Verlag, Berlin.

[10] I.Mrázová: Condensed Knowledge Representation in BP-networks, in: Proc. of the ISE'94, Hamburg, IEE, (5 - 9 Sept. 1994), pp. 118-123.

[11] I. Mrázová: Internal Representation in BP-networks, PhD-Thesis, 1997, ÚIVT AV ČR, Prague.

[12] I. Mrázová: Generalized Relief Error Networks and Patterns with Lower Errors, accepted for publication in the International Journal of Smart Engineering System Design, (January 2001).

[13] I. Mrázová: Controlled Learning of GREN-Networks, accepted for publication in: Proc. of the ANNIE 2001, St. Louis, (4 - 7 Nov. 2001).

[14] D. E. Rumelhart, G. E. Hinton, R. J. Williams: Learning Representations by Back-Propagating Errors, in: Nature, (323) (1986), pp. 533-536.

[15] D. J. Schaffer: Combinations of Genetic Algorithms with Neural Networks or Fuzzy Systems, in: J. M. Zurada, R. J. Marks II, C. J. Robinson (Eds.): Computational Intelligence Imitating Life, (1994), IEEE Press, pp. 371-382.

[16] H. J. Zimmerman: Hybrid Approaches for Fuzzy Data Analysis and Configuration Using Genetic Algorithms and Evolutionary Methods, in: J. M. Zurada, R. J. Marks II, C. J. Robinson (Eds.): Computational Intelligence Imitating Life, (1994), IEEE Press, pp. 364-370.

Algoritmy v oblasti získavania informácií

Rastislav Lencses

Department of Computer Science
Faculty of Science
University of P. J. Šafárik,
Jesenná 5, 041 54 Košice, Slovakia

lencses@duro.science.upjs.sk

Abstrakt.
V tomto príspevku podávame prehľad algoritmov týkajúcich sa information retrieval spolu s dosiahnutými vlastnými výsledkami. Je tu popísaný vektorový model, rozšírené booleovské vyhľadávanie a fuzzy vyhľadávanie, taktiež spätná väzba, tezaurus, sémantické siete, zhlukovanie, parsovanie a rozdeľovanie dokumentov.

1. Úvod

Tento príspevok sa zaoberá prehľadom základných metód používaných pri získavaní informácií (information retrieval). Typická situácia - užívateľ zadá otázku, kde sú špecifikované jeho požiadavky, systém otázku spracuje a vráti užívateľovi zoznam dokumentov zodpovedajúcich otázke.

Základ práce systému pre získavanie informácií (IRS - information retrieval system) spočíva vo vytvorení indexu zo množiny statických dokumentov (ešte pred prvou užívateľskou otázkou), spracovaní užívateľskej otázky a nájdení (a utriedení) dokumentov relevantných ku otázke. Pre IRS je typické, že ťažisko práce spočíva vo vyhľadávaní informácií, pridávanie a tobôž úprava alebo mazanie dokumentov je v omnoho menšej miere, čím sa tieto systémy líšia od tradičných databázových informačných systémov. Ďalší podstatný rozdiel je vo využívaní neštruktúrovanej formy dát, i keď tu sa odlišnosti stále viac stierajú. Po položení otázky IRS vráti množinu dokumentov. Tieto označíme ako získané (retrieved). V množine všetkých dokumentov, v ktorých sa vyhľadáva, existuje podmnožina dokumentov relevantných (relevant) ku otázke. Cieľom je, aby sa množina relevantných dokumentov rovnala množine získaných dokumentov, v praxi však je dobrým výsledkom, ak majú veľký neprázdny prienik. Tento prienik označíme ako relevantné získané dokumenty. Potom môžeme definovať dve čísla, presnosť (precision) a úplnosť (recall) , ktoré vyjadrujú kvalitu IRS:

$$presnost = \frac{pocet\ \ relevantnych\ \ ziskanych\ \ dokumentov}{pocet\ \ ziskanych\ \ dokumentov}$$

$$uplnost = \frac{pocet\ \ relevantnych\ \ ziskanych\ \ dokumentov}{pocet\ \ relevantnych\ \ dokumentov}$$

Čím väčšiu úplnosť požadujeme, tým sa presnosť znižuje.

2. Používané modely a algoritmy

IRS určuje relevantnosť dokumentov ku užívateľskej otázke podľa jedného (alebo kombinácie viacerých) modelov. Doteraz známe modely sú :

1. Vektorový model (Vector Space Model) - otázka i dokumenty sú považované za vektory vo vektorovom priestore termov. Relevantnosť dokumentu a otázky sa vypočíta na základe podobnosti dvoch vektorov (zvyčajne ich uhol alebo vzdialenosť).

2. (Rozšírené) booleovské vyhľadávanie - otázka je reprezentovaná v booleovskej forme s logickými spojkami a treba nájsť množinu dokumentov, ktoré spĺňajú otázku. Pri rozšírenom vyhľadávaní sa ráta aj s váhami termov.

3. Fuzzy vyhľadávanie - dokumentu je priradená fuzzy množina. Termom dokumentu sa priradia čísla vyjadrujúce váhu ich dôležitosti. Potom sa použije prienik, zjednotenie a doplnok fuzzy množín otázky a dokumentov, čím sa získa výsledok.

4. Pravdepodobnostné vyhľadávanie - pre každý term v množine dokumentov sa vypočíta pravdepodobnosť toho, že sa term vyskytne v relevantnom dokumente. Miera podobnosti otázky a dokumentu je potom vypočítaná ako kombinácia pravdepodobností každého zo zodpovedajúcich si termov z otázky a dokumentu.

5. Genetické algoritmy - optimálna otázka (na nájdenie relevantných dokumentov) vznikne genetickým vývojom. Prvotná otázka obsahuje náhodné alebo heuristické váhy. Nové otázky vzniknú modifikáciou týchto váh;

prežívajú len tie, ktoré najlepšie zodpovedajú známej množine relevantných dokumentov.

6. Neurónové siete - neuróny reprezentujú termy a dokumenty, väzby medzi neurónami popisujú príslušnosť termov ku dokumentu a koeficient podobnosti medzi termami otázky a dokumentami. Sieť môže byť pomocou spätnej väzby trénovaná zmenou váh väzieb.

7. Latent semantic indexing - výskyt termov v dokumentoch je reprezentovaný maticou obsahujúcou počet termov v dokumentoch. Táto matica je potom redukovaná pomocou Singular Value Decomposition (SVD), čím sa dosiahne to, že dokumenty s podobným významom sa budú vo viacdimenziálnom priestore nachádzať blízko seba.

V ďalšom si bližšie popíšeme prvé tri typy algoritmov.

3. Vektorový model

Model je založený na myšlienke, že význam dokumentu závisí od slov, ktoré sa v ňom nachádzajú. V tomto modeli je otázka i množina dokumentov reprezentovaná vektormi. Relevantnosť otázky ku dokumentom je potom spočítaná na základe podobnosti vektora otázky a vektorov dokumentov.

Vektory sa konštruujú nasledovne [1]. Pre celú množinu dokumentov sa určí postupnosť významných termov, podľa ktorých sa potom bude môcť vyhľadávať (napr. spojky a častice sú zvyčajne vylúčené). Potom sa pre každý dokument vytvorí vektor, ktorý obsahuje na $i - tom$ mieste 1 alebo 0, podľa toho, či sa v ňom vyskytuje $i - ty$ významný term. Táto binárna reprezentácia sa dá zlepšiť použitím informácie nielen o tom, či sa významný term v dokumente nachádza, ale aj koľkokrát sa v dokumente nachádza. Potom sa na $i - tom$ mieste bude vyskytovať počet výskytov $i - teho$ významného termu v dokumente. Ďalšie vylepšenie možno dosiahnuť uplatnením nasledovnej myšlienky: termy, ktoré sa vyskytujú v menšom množstve dokumentov, by mali získať väčšiu dôležitosť, než tie, ktoré sa vyskytujú častejšie. Konštrukcia takého vektora je formálne popísaná nižšie. Analogicky, otázka je tiež považovaná za dokument a vytvára sa rovnako. V prípade, že užívateľ chce preferovať niektoré termy, vo vektore otázky sa zvýši váha príslušného termu.

Formálne definujme:
n = počet významných termov v množine dokumentov
tf_{ij} = počet výskytov termu t_j v dokumente D_i (term ferquency)

df_i = počet dokumentov, ktoré obsahujú term t_j
$idf_j = log(\frac{M}{df_j})$, kde M je veľkosť množiny dokumentov (idf sa často označuje ako inverse document frequency)

Pre každý dokument D_i bude vytvorený vektor d dažky n, ktorý bude obsahovať na $j - tom$ mieste váhu $j - teho$ termu v postupnosti významných termov:

$$d_{ij} \; = \; tf_{ij} \; . \; idf_j$$

Tento vzťah je základný. Neskôr boli nájdené (a experimentálne overené) zložitejšie vzťahy, napríklad [2]:

$$d_{ij} = \frac{(log \; tf_{ij} + 1.0) \; . \; idf_j}{\sum_{j=1}^{n}((log \; tf_{ij} + 1.0) \; . \; idf_j)^2}$$

Najdôležitejšou zmenou je tu prekonanie príliš veľkého vplyvu termu s vysokou term frequency (tomu zabráni sa pomocou použitia $log \; tf$).

Miera podobnosti medzi dokumentom D_i a otázkou Q je založená na skalárnom súčine oboch vektorov:

$$Sim(Q, D_i) = \sum_{j=1}^{n} w_{qj} \; . \; d_{ij}$$

kde w_{qj} je vektor otázky. Takto definovaná miera podobnosti je opäť len základná. Medzi prepracovanejšie patrí kosínus uhla dvoch vektorov a miery založené na ňom:

$$Sim(Q, D_i) = \frac{\sum_{j=1}^{n} \; w_{qj} \; . \; d_{ij}}{\sqrt{\sum_{j=1}^{n} \; (d_{ij})^2 \; . \; \sum_{j=1}^{n}(w_{qj})^2}}$$

Zaujímavý model založený na vektorovom modeli je distributional semantics [3]. V tomto modeli sa predpokladá, že dva termy sú si podobné, ak ich kontexty sú podobné. Analogicky, dva dokumenty sú si podobné, ak ich kontexty sú podobné. Kontext je definovaný ako postupnosť termov v nejakom okolí daného termu (veta, odstavec alebo nejaké delta-okolie). Pre term t sa vytvorí vektor:

$$c^t \; = \; (c_j^t, \dots \; , c_M^t),$$

kde c_j^t je počet výskytov $j - teho$ termu z indexovacej množiny I v kontexte termu c^t. Množina I s kardinalitou M obsahuje vybrané termy, o ktorých sa predpokladá, že budú dobrými diskriminátormi. Celý dokument potom bude reprezentovaný pomocou

$$d_{ij} \;=\; tf_{ij}\;.\;idf_j\;.\;c^j$$

Tento model sme v [4] obohatili myšlienkou, že kontext nachádzajúci sa bližšie ku termu je dôležitejší. Ko-frekvenciu dvoch termov potom definujeme ako usporiadanú dvojicu (n, s), kde n je počet ich výskytov v nejakej textovej jednotke a s je ich vzdialenosť v tejto jednotke:

$$c(t_1, t_2) \;=\; (n(t_1\;,\;t_2)\;,\;s(t_1\;,\;t_2))$$

Túto dvojicu môžeme agregovať do jedného čísla, aby sme mohli spočítať podobnosť:

$$c(t_1, t_2) \;=\; \frac{n(t_1, t_2)}{s(t_1, t_2) + 1}$$

Ďalší postup je už rovnaký, ako vo vektorovom modeli.

Pri implementácii vektorového modelu sa namiesto sekvenčného prehľadávania množiny dokumentov používajú invertované indexy dokumentov [5]. Indexy sa však používajú aj v iných modeloch. Tento index je vybudovaný pred položením prvej otázky. Index sa skladá z usporiadanej postupnosti významných termov. Pre každý term je v indexe určený zoznam dokumentov, v ktorých sa vyskytuje. Vylepšená verzia obsahuje aj počty výskytov v danom dokumente. Najlepšia (ale aj najviac zaťažujúca miesto) verzia eviduje aj miesto výskytu daného termu v dokumente. V indexe je aj informácia o idf daného termu a jeho váha v príslušnom dokumente. Na zníženie veľkosti indexu sa využívajú rôzne komprimačné metódy vyvinuté na komprimáciu textu. Zaujímavá je technika znižovania pamäťového miesta pre identifikátory dokumentov [6]. Tieto sú zvyčajne čísla, ktorým môžeme prideľovať miesto podľa skutočnej potreby (malé čísla vyžadujú menší počet bytov než väčšie). V prípade veľkej množiny dokumentov a konštantnej veľkosti miesta pre identifikátor by sme zbytočne použili pre malé čísla veľa pamäťového miesta. Riešením je využiť pomocný identifikátor, v ktorom sa špecifikuje, koľko bytov treba použiť pre samotný identifikátor dokumentu. Ďalšia metóda spočíva v použití signatúr dokumentov [7]. Tu sa dokument reprezentuje nie cez index, ale pomocou signatúry - dokument je zakódovaný často len pomocou niekoľkých bitov (bytov). Je to podobné hašovacej funkcii. Termy obsiahnuté v otázke sa transformujú podobne a potom sa pomocou pattern matching testuje ich prítomnosť v signatúre. Samozrejme, signatúry môžu zodpovedať viacerým dokumentom, preto po nájdení vyhovujúcich signatúr je nutné sekvenčne prehľadať originálne dokumenty.

4. Rozšírené booleovské vyhľadávanie

Štandardné booleovské vyhľadávanie neumožňuje utriedenie získaných dokumentov podľa relevantnosti. Otázka je v tvare booleovskej formuly (použitie logických spojok) a nájdené dokumenty musia zodpovedať otázke rovnako.

V rozšírenom booleovskom vyhľadávaní [8] pridáme ku termom dokumentu aj váhu (napríklad počet výskytov termu v dokumente). Dokument s vyšším počtom termov s väčšou váhou (zodpovedajúce termom z otázky) bude viac releventný ku otázke. V prípade jednoduchej otázky ($t_1\ OR\ t_2$) budeme pre vektory obsahujúce váhy nájdených dokumentov ($w_{i1},\ w_{i2}$) počítať vzdialenosť od bodu $(0,0)$:

$$Sim(Q,d_i) \;=\; \frac{\sqrt{w_{i1}^2\ +\ w_{i2}^2}}{\sqrt{2}}$$

V prípade otázky ($t_1\ AND\ t_2$) budeme pre vektory obsahujúce váhy nájdených dokumentov (w_{i1}, w_{i2}) počítať vzdialenosť od bodu $(1,1)$:

$$Sim(Q,d_i) \;=\; 1 \;-\; \frac{\sqrt{(1-w_{i1}^2)+(1-w_{i2}^2)}}{\sqrt{2}}$$

Ak budeme uvažovať aj váhy termov otázky $(q_1,\ldots,q_n)$ obsahujúcej ľubovoľné množstvo termov a využijeme zovšeobecnenú euklidovskú vzdialenosť, dostaneme:

$$Sim(Q_{\bigvee_{j=1}^{n} q_j}\ ,\ d_i) \;=\; \left[\frac{\sum_{j=1}^{n}(q_j w_{ij})^p}{\sum_{j=1}^{n} q_j^p}\right]^{\frac{1}{P}}$$

$$Sim(Q_{\bigwedge_{j=1}^{n} q_j}\ ,\ d_i) \;=\; 1-\left[\frac{\sum_{j=1}^{n}(q_j(1-w_{ij}))^p}{\sum_{j=1}^{n}(q_j)^p}\right]^{\frac{1}{P}}$$

kde p je parameter slúžiaci na spresňovanie modelu

5. Fuzzy booleovské vyhľadávanie

Podstata tohto modelu je možnosť definovať, "o čom všetkom" dokument je a s akou mierou. Dokument reprezentovaný pomocou fuzzy množiny [8]. Jej prvky majú definovaný stupeň členstva (confidence factor) v množine pomocou členskej funkcie (zobrazenie z množiny termov dokumentu do intervalu $[0,1]$). Dokument je teda vyjadrený:

$$D = \{ (c_1, f_{D(c1)}), \dots , (c_n, f_{D(cn)}) \}$$

kde $C = c_1, \dots , c_n$ je množina konceptov a $f_D : C \rightarrow [0,1]$ je členská funkcia. Pre term t existuje množina D_t veľkosti m, ktorá obsahuje dokumenty (v ktorých sa nachádza term t) a stupne členstva pre tieto dokumenty:

$$D_t = \{ (D_1, cf_t, 1), \dots (D_m, cf_{t,\ m}) \}$$

Otázka je pozostáva z termov pospájaných fuzzy spojkami. Pre tieto termy sa skonštruujú D_t množiny a vykonajú sa príslušné operácie (napr. pre prienik fuzzy množín sa použije minimum ich členských funkcií a pre zjednotenie maximum). Výsledkom teda bude fuzzy množina dokumentov a ich stupeň členstva pre položenú otázku.

6. Pomocné algoritmy

Pri základných algoritmoch v oblasti IR sa často využívajú pomocné algoritmy, ktoré možno zvyčajne použiť pri rôznych vyhľadávacích stratégiách a kombinovať ich [9]:

spätná väzba (relevance feedback): pri získaní dokumentov zodpovedajúcich prvotnej otázke sa prvých n najrelevantnejších dokumentov skontroluje (manuálne alebo algoritmom) a niektoré termy z nich sa pridajú do otázky. Tento proces iteruje.

rozdeľovanie dokumentov: dokument je rozdelený na časti, ktoré sú považované za samostatné dokumenty, keďže často iba časť dokumentu je relevantná voči otázke.

zhlukovanie (clustering): dokumenty alebo termy sú zhlukované do skupín. Otázka je potom kladená iba na tie skupiny, ktoré sú relevantné ku otázke.

parsovanie: spracovanie fráz, použitie stemmingu, odrezávanie prípon a predpon

tezaurus: slovník obsahujúci podobné slová a synonymá používaný na expandovanie otázky alebo dokumentu

n **- gramy**: termy sú rozdelené do n-gramov (sekvencie n znakov), ktoré sa potom využívajú na zistenie relevantnosti otázky a dokumentu. Ich použitím sa obmedzia problémy s chybne zapísanými termami

sémantické siete: hierarchie konceptov (koncepty sú spájané väzbami so stanovenou mierou asociácie). Tieto siete sa používajú na expandovanie otázky alebo dokumentov

7. Spätná väzba (relevance feedback)

Tento algoritmus je založený na iteratívnej reakcii užívateľa, ktorý z postupnosti nájdených dokumentov označí tie, ktoré sú podľa neho najrelevantnejšie a podľa nich sa upraví otázka, ktorá je znova položená IRS. Základný algoritmus (pre vektorový model) zmeny vektora otázky je podľa nasledovnej rovnice [10]:

$$Q^{'} = \alpha Q + \beta \sum_{i=1}^{n_1} \frac{R_i}{n_1} - \gamma \sum_{i=1}^{n_2} \frac{S_i}{n_2}$$

kde R a S je postupnosť dokumentov označených ako relevantných a irelevantných, n_1 a n_2 je ich počet, α, β, γ sú Rocchiove váhy. Ich hodnota sa určuje experimentálne (y je zvyčajne 0). Relevantnosť dokumentov sa určí manuálne, alebo sa použije prvých n dokumentov usporiadaných podľa relevantnosti. V novej otázke sa nemusia použiť všetky termy z nájdených relevantných dokumentov. Stačí vybrať len najdôležitejšie a tie pridať do novej otázky. Dôležitosť termov je možné stanoviť viacerými mierami, napríklad [11]:

$$n_k = \sum_{i=1} N \frac{tf_{i\ k}}{f_k} \log_2(f_k tf_{i\ k})$$

kde $tf_{i\ k}$ je počet výskytov termu k v dokumente i, f_k je počet výskytov termu k v celej množine dokumentov a N je počet termov v celej množine dokumentov.

8. Zhlukovanie (clustering)

Dokumenty sa zaraďujú do skupín (zhlukov, kategórií), avšak tieto nie sú definované vopred (tým sa zaoberá document routing alebo filtering) , ale vznikajú v procese zhlukovania. Zhlukovanie sa používa na zníženie priestoru prehľadávania (pri hľadaní relevantných dokumentov) a na sprehľadnenie zobrazenia výsledných relevantných dokumentov (nezobrazia sa všetky, ale iba zástupcovia zhlukov). Dokumenty v zhluku sú charakterizované maximálnou podobnosťou navzájom a minimálnou podobnosťou voči dokumentom z iných zhlukov. Zhluky tvorí automaticky systém na základe definovanej miery podobnosti medzi dokumentmi, alebo podľa vlastností vzniknutých skupín. Existujú dve základné skupiny algoritmov vytvárajúcich zhluky [12]. Division (prerozdeľujúce) algoritmy vytvoria z množiny dokumentov nejakým spôsobom zhluky (náhodne, alebo podľa trénujúcej vzorky) a potom ich prerozdeľujú podľa definovaného pravidla. Dokumenty sa teda v priebehu iterácií môže zaradiť postupne do rôznych zhlukov. Koná sa to až do nájdenia globálneho, prípadne lokálneho optima. Najznámejší je $k - means$, algoritmus hľadajúci k stredov. Vytvára daný počet zhlukov a dáta zhromažďuje okolo ich stredov (zhluky teda majú sférický tvar vo vzťahu ku podobnostnej metrike). Jeho zložitosť je $O(nkT)$, kde k je počet stredov a T počet iterácií.

Hierarchical (hierarchické) algoritmy rozdelia množinu dokumentov na niekoľko zhlukov. Každý zhluk sa potom rozdeľuje ďalej, až kým sa nenájde optimum. Týmto spôsobom rozdeľujú algoritmy typu "zhora nadol", analogicky spájajú algoritmy typu "zdola nahor". Hierarchické algoritmy sa líšia podľa definície podobnosti zhlukov:

1. single-link a complete-link (jednoduché a úplné spojenie) - algoritmy, ktoré definujú podobnosť zhlukov na základe maximálnej a minimálnej podobnosti každej dvojice dokumentov z dvoch zhlukov.

2. group average (priemer skupiny) - tento algoritmus definuje podobnosť zhlukov na základe priemeru (alebo inej agregácie) zhluku.

Zastavovacích kritérií bolo vytvorených pomerne veľa, v praxi sa však využívajú iba najjednoduchšie (napríklad zastav, keď je počet zhlukov n).

9. Rozdeľovanie dokumentov (passage based retrieval)

Základná idea je založená na tom, že najmä dlhšie dokumenty majú voči otázke relevantné len niektoré svoje časti. Ostatné časti takého dokumentu potom prispievajú ku zníženiu jeho relevantnosti. Preto sa dokument rozdelí na časti (manuálne alebo automaticky) a pre tieto časti sa vypočíta koeficient podobnosti voči otázke [13]. Dokumenty sú zväčša už prirodzene rozdelené na odstavce a vety, prípadne môžu byť rozdelené do častí autorom. Koeficienty relevantnosti častí dokumentu sa pomocou nejakej agregačnej funkcie zlúčia a vypočíta sa koeficient relevantnosti celého dokumentu. Automatické generovanie častí dokumentu je závislé na otázke. Pre term z otázky sa označí časť dokumentu (okolie termu stanovenej veľkosti). Ďalšie časti môžu nasledovať po (a pred) označenej časti tak, že budú na seba súvislo nadväzovať, alebo sa budú prekrývať.

10. Parsovanie

Každý IRS musí mať schopnosť identifikovať a upravovať termy, ktoré tvoria dokumenty. Jednotlivým termom sa v IRS transformujú veľké písmená na malé (napr. začiatok vety). Termy sa zbavujú častých prípon a predpon (stemming), takže index dokumentu bude menší a úplnosť (recall) vyhľadávania sa zvýši. Stop list je zoznam veľmi často používaných slov bez informačného významu (spojky a pod.), ktoré sa do indexu nezapíšu a v otázke sa neuvažujú. Frázy sú spojenia termov (bez stop words), ktoré sa často v texte používajú. Indexovaním fráz (a ich používaním v otázkach) sa zvýši presnosť aj úplnosť vyhľadávania. Extrakcia informácií sa zaoberá vyhľadávaním štrukturovaných dát v textoch. Typické je identifikovanie mien, dátumov, miest, hodnôt a pod. Používané algoritmy sú založené na štatistike alebo pravidlách.

11. Tezaurus

Tezaurus podporuje zvýšenie presnosti a úplnosti vyhľadávania v IRS pomocou využitia slovníka synoným [14]. Dokument môže byť relevantný voči otázke a nemusí obsahovať termy z otázky. Vtedy tezaurus môže byť použitý na doplnenie synoným do otázky alebo na priradenie všeobecného pojmu pre všetky synonymá termu. Tezaurus môže byť vybudovaný ručne (veľmi pracné) alebo automaticky. Metódy automatickej výstavby tezaura sú založené zväčša na porovnávaní vektorov termov (pre term t sa vytvorí vektor vt , ktorý obsahuje napr. počty výskytov termu v dokumentoch kolekcie). V [15] sme sa zaoberali využitím fuzzy tezaura. Tento môže obsahovať ekvivalencie, synonymá, hierarchické hypernymá a hyponymá (kategórie) a prípadne asociatívne vzťahy. Všetky relácie sú ohodnotené váhou. Expandovanie otázky obsahujúcej termy a ich váhy spočíva v pridaní termov z tezaura, ktoré sú v relácii s termami otázky. Nové váhy pre tieto termy sa zväčša určujú heuristicky, my sme navrhli využiť existujúcu teóriu fuzzy logiky na určenie nových váh. Pomocu viachodnotového modus ponens dostaneme:

$$w^{'} = C(w,\ I(b)),$$

kde w je váha termu z otázky, ktorý expandujeme, $w^{'}$ je nová váha pre term, ktorý je doplnený do otázky, C je nejaká konjunkcia (možná t-norma) a $I(b)$ je váha relácie, v ktorej je originálny a dopĺňaný term.

12. Sémantické siete

Sémantické siete sú založené na myšlienke, že znalosť môže byť reprezentovaná konceptami, ktoré sú pospájané linkami vyjadrujúce rôzne typy vzťahov medzi konceptami [16]. Sémantická sieť je potom množina uzlov a hrán. Hrany medzi uzlami sú pomenované názvom vzťahu (napr. "je", "je synonymum", "týka sa"). Uzly obsahujú informácie o koncepte spätom s uzlom, jeho bližšie charakteristiky, ktoré môže nadobúdať (farba, veľkosť a pod.) Sémantické siete sa používajú na vyriešenie problému, keď dokument neobsahuje termy nachádzajúce sa v otázke a napriek tomu je relevantný. Vtedy by v sémantickej sieti mala existovať cesta, ktorou sú spojené termy z otázky a dokumentu. Táto cesta má nejakú cenu (sémantická vzdialenosť), ktorá vyjadruje podobnosť termov. Sémantické siete fungujú podobne ako tezaurus, pomocou nej však možno modelovať bohatšiu sieť vzťahov. Známy príklad vybudovanej sémantickej siete je WordNet.

Zoznam literatúry

1 *Salton, G., Yang, C., Wong, A.*, **A vector-space model for automatic indexing**. Communication of ACM 18 (1975) 613–620

2 *Salton, G., Buckley, C.*, **Term-weighting approaches in automatic text retrieval**. Information Processing and Management 24 (1988) 513–523

3 *Besancon R, Rajtman M., Chappelier J.C.*, **Textual Similarities based on a Distributional Approach**. International Workshop on Similarity Search (IWOSS99), Firenze, Italy, Sept 1-2 (1999)

4 *Lencses, R.*, **Document Clustering with Neural Network**. Proceeding of the European Symposium of Computational Intelligence. "Physica Verlag A Springer - Verlag Company", Heidelberg 2000, 296–301

5 *Bleir, R.*, **Treating hierarchical data structures in the SCD time-shared data management system**. Proceedings, 22nd ACM National Conference 1967, 41–49

6 *Bell. T, Cleary, J., Witten, I.*, **Text compression**. Prentice Hall 1990

7 *Lee, D., Ren, L.*, **Document ranking on weight-partitioned signature files**. ACM transactions on Information Systems 14 (1996) 109–137

8 *Salton, G.*, **Automatic Text Processing**. Addison-Wesley 1989

9 *Grossman, D. A., Frieder, O.*, **Information Retrieval: Algoritms and heuristics**. Kluwer Academic Publishers 2000

10 *Rocchio, J. J.*, **SMART Retrieval System - Experiments in Automatic Document Processing**. Prentice Hall 1971, 313–323

11 *Harman, D.*, **Towards interactive query expansion**. In 11th International ACM SIGIR Conference on Research and Development in Information Retrieval 1988, 321–331

12 *Lu, A., Ayoub, M., Dong, J.*, **Ad hoc experiment using EUREKA**. In Proceedings of the Fifth Text Retrieval Conference 1996, 229–239

13 *Callan, J.*, **Passage-level evidence in document retrieval**. In Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval 1994, 302–310

14 *Salton, G.*, **The SMART Retrieval System - Experiments in Automatic Document Processing**. Prentice Hall 1971, 324–336

15 *Lencses, R.*, **Query Expansion with fuzzy thesauri**. Proceedings of the Third Conference on Electrical Engineering and Information Technology for Ph.D. students, Slovak University of Technology, Bratislava 2000

16 *Minsky, M.*, **The Psychology of Computer Vision**. McGraw-Hill Book COmpany 1975, 211–277

Monads in Logic Programming

Jan Hric

Department of Theoretical Computer Science
Charles University, Faculty of Mathematics and Physics
118 00 Praha 1, Malostranské nám. 25
Czech Republic

Jan.Hric@mff.cuni.cz

phone: +420 2 2191 4246

Abstract

This paper describes implementation of monads in logic programming, particularly in Prolog. The idea of monads is taken from functional programming. During an implementation of monadic library we had to solve problems concerning higher-order code, free variables, and non-directionality of data flow. These problems are identified and solutions are suggested. A special syntax, the so-called do-notation, is introduced to allow more user-friendly code.

1. Introduction

Monads are a technique used in functional programming [4],[2], which allow better modularity and adaptability of a code to changes. We suggest to use them also in logic programming. In this paper we develop techniques and implementations of monads in logic programming, particularly in Prolog. Previous attempt to include monads into logic programming [1] used the language λProlog.

The structure of the paper is as follows. In section 2 we introduce monads in general and a special syntax, the so-called do-notation. In section 3 we introduce particular monads for errors, output, nondeterminism, state and context. Section 4 contain conclusions and possibility of futher work.

2. Monads

We describe an implementation of a few particular monads. An application of monads is shown on a simple interpreter of arithmetic expresions. This interpreter captures the general idea of recursive traversal through a data structure.

```
interp(con(N),val(N)).
interp(inc(E1),val(N)) :-
  interp(E1,val(N1)), N is N1+1.
interp(E1+E2,val(N)):-
  interp(E1,val(N1)),
  interp(E2,val(N2)), N is N1+N2.
```

The interpreter takes the expression in the first argument and returns the value in the second argument. The arithmetic expression can have functors $con/1$, $inc/1$, and $plus/2$ with a number, an expression, and two expressions as their arguments, respectively. The result is a number wrapped in functor $val/1$.

The interpreter has only basic functionality. As we introduce particular monads, we introduce also supplementary kind of terms, which enable to call special operations of monads.

2.1. Characterization of monads

A monad consists of data structure (type constructor[4]) and two operations *unit* and *bind*. Monads convert the computation between a and b into computation between a and $M\ b$, where M is type constructor. The operation *unit* projects the value into the monad. The operation *bind* composes computations in the following way. Given a data structure in the form of a monad, *bind* for each plain data in the structure computes the result and integrates them in the resulting structure.

Each monad enables to represent some additional information about computation. To include such information into a result each monad has specific predicates. So the monadic interpreter must be extended to enable a computation with extended terms and calls to these predicates.

Depending on the monad chosen, the same program in a monadic form can be used for different extensions. Monads, particularly their operations *unit* and *bind* serve as procedures with late binding. To show the basic idea of a transformation to the monadic style, we describe a simple version of the interpreter.

[4]As we work in Prolog, we use a concept of types informally.

The simple version covers only first-order monads that are represented using a data structure, as maybe, output and list monad. The full version will be introduced with do-notation in the following section, and will cover also monads represended using calls, as reader and state.

```
interp(T,con(N),MN):-unit(T, val(N),MN).
interp(T,inc(E1),MN0) :-
  interp(T,E1,MN1),
  bind(T, MN1, l2([val(N1),MN1V],[],
    (N is N1+1, unit(T,N2,MN2))),MN0).
interp(T,E1+E2,MN0):-
  interp(T,E1,MN1)),
  bind(T,MN1,l2([N1,MN1V],[],
    (interp(T,E2,MN2),
     bind(T, MN2, l2([N2,MN2V], [],
       (N is N1+N2, unit(T,val(N),MN2V))),
       MN1V))), MN0).
```

An implementation of monads in functional programming can take an advantage of the type system with overloading (type and constructor classes). As this is not available in Prolog, we added one parameter to the predicates *unit* and *bind* which implement monads. The parameter represents the type, and according to this parameter, the right operation for the particular monad is called.

The result in each clause is delivered using *unit*. It projects its second parameter, a plain value, to the monad according to the type parameter.

The *bind* predicate gets a monad in the second parameter. To each plain value in the monad, it starts the computation given by the third parameter, and all results are collected in the last parameter.

The third parameter represents a higher-order predicate. We choose such representation of higher-order predicates: functor $l2/3$, which has formal parameters in the first argument, a list of "global" variables in the second parameter and a goal in the last one. Each higher-order predicate must be standardized apart before calling. The list of the formal parameters allows to bind actual parameters given for particular call. The list of global variables enables to bind variables in a newly renamed term to the corresponding variables in the pattern term.

The suggested syntax is not too user-friendly. We suggest an improved so called do-notation in the following chapter.

2.2. Introduction of do-notation

The improved so called do-notation known in funcional programming can be adapted to our approach. It hides some technical details of implementation and enables to write more compact code. The type parameter is again present as the first argument. The second argument is the goal for computing a result which should be included into the resulting structure. The third parameter is a list of goals which correspond to a body. The fourth parameter is the result in monadic form.

The goals in the third parameter can be monadic or nonmonadic. The former are in the form $Value - Result <= Glob\hat{\ } Goal$ and are processed using the *bind* predicate. $Value$ is the input value for the rest of the computation and *Result* is the result of the current *Goal*. *Glob* is the list of (possibly) free variables, which are considered global and should not be standardized apart. The nonmonadic goals wrapped in the $r/1$ functor are processed directly. This notation enables to call directly goals which are not in monadic form. They cannot raise any special operations and are usually used to call built-in predicates.

```
do(T,GX,[],MX):- unit(T,GX,MX).
do(T,GX,[X1-MX1 <= Glob^Goal| Gs], M0):-
    Goal, bind(T, MX1, l2([X1,MX1V],Glob,do(T,GX,Gs,MX1V)), M0).
do(T,GX,[r(Goal)|Gs],M0):- Goal, do(T,GX,Gs,M0).
```

The part of the interpreter using do-notation follows:

```
interp2(T, con(N),MA):- do(T, l2([N0],[N],N0=val(N)),[], MA).
interp2(T, inc(E1),MN0):- do(T, l2([N2],[N1], inc(N1,N2)),
        [N1-MN1 <= [E1]^interp2(T, E1, MN1) ], MN0).
interp2(T, E1+E2,MN0):- do(T, l2([N0],[N1,N2], plus(N1,N2,N0)),
        [N1-MN1 <= [E1,E2]^interp2(T, E1, MN1),
         N2-MN2 <= [E2,N1]^interp2(T, E2, MN2) ], MN0).
```

We explain now the key ideas of the interpreter and the do-notation. There are the form of the goal in the second parameter and global variables in goals in the third parameter.

The goal, which computes the result, is explicitely expressed in the second parameter of the *do* predicate using the $l2$ functor. If the monadic result is a goal, then due to a data flow and a possible presence of a free variables, a call to the goal cannot be performed during computation in the monad. It must be postponed to a later stage, because the formal parameters of the monadic result are not known during computation in the monad.

The do-notation uses representation of higher-order predicates. So it must perform standardization apart explicitly. The goals are represented using the *l*2/3 predicate and new copies are created using *cpl*2 predicate which uses Prolog built-in *copy_term* predicate. But goals can have some parameters instantiated and we do not want to lose binding of variables. Variables, which should not lose bindings, are called global and are represended in the second parameter of the *l*2 structure in a list. After copying the goal structure *l*2, they are bound again. This parameter contains usually input data as well as results of previous monadic computations. In the current version of the library they must be stated explicitly. As this is annoying for a user, we are looking for a better notation and methods of description.

3. Particular monads

We describe now particular monads. There are *maybe*, *output*, *list*, *state* and *reader* monads for representing errors, output, nondeterministic computations, passing state, and passing context.

3.1. Monad maybe

The monad *maybe* enables to identify errors in computation. The result is the term $fail$ meaning error during computation or the term $just(X)$ when computation was successful. The proper data are in the latter case in the parameter X.

```
unit(maybe,XP, just(X)):-cpl2(XP,[X],[],G), call(G) .
bind(maybe,just(X),P,MY):- cpl2(P, [X,MY], [], G), call(G).
bind(maybe,fail,_P,fail).
```

The first argument in both predicates *unit* and *bind* is the atom *maybe* as an identification of the monad. The predicate *unit* wraps the result into the functor $just/1$. The predicate *bind* propagates the value $fail$ as the result. If the partial result is the value $just(X)$, then *bind* calls higher-order predicate P on the value X. The result MY of *bind* is also the result of the whole computation.

The standardization apart of the structure P is hidden in the predicate *cpl*2 (copy l2). It uses the predicate *copy_term* (of BinProlog) to get a term with fresh variables, then it binds arguments and global variables.

```
cpl2(P,Args,Glob,G):-
```

```
  P  = l2(_,V2,_),
  copy_term(P,P0),
  P0 = l2(Args,V2,G).
```

The monad *maybe* do not yet has any new functionality. So the interpreter does not have any means to raise an error. The predicate *zero*/2 enables to raise an error.

```
zero(maybe,fail).
```

We add the term *error* to the input terms and extend the interpreter by the following clause.

```
interp(T,error,MN):-zero(T,MN).
```

3.2. Monad output

The monad *output* is used for remembering the output created during a computation. The monad gives except the value also an output. The output is in our case a list of terms, which were sent to the output using the predicate *write*/3. The representation of the monadic values is the term $X - Out$, where X is the result and Out is the output.

```
unit(output,XP,X-[]):- cpl2(XP,[X],[],G), call(G) .
bind(output,X1-O1,P,X0-Out0):-
  cpl2(P,[X1,X0-O2],[],G),
  append(O1,O2,Out0), call(G).
write(output,Out,X, X-Out).
```

The predicate *unit* returns an empty output. The predicate *bind* uses the value for futher computation and concatenates the outputs of both parts. The predicate *write* sends the given value to output. The result is not interesting as it will not be used.

We extend the input expressions with terms in the form $out(Out, Exp)$, where Out is the (single) term sent to output before the expressions Exp is evaluated. We could analogically introduce expressions which send the output after evaluation of the expression. The called predicate *write* is the same for both cases.

```
interp(T,out(O,E1),MV0):-write(T,[O],MN1),
  bind(T,MN1,l2([_,MN1V],[MN1],
    interp(T,E1,MN1V) )).
```

Here for the first time we used a list of global variables in our representation of higher-order predicates with the $l2$ functor. If we want to maintain a correspondence among free variables in evaluated expression on the output, we must use some technique to protect variables from standardization apart. In the presented interpreter the value of the expression $write(t(X), write(t(X), con(1)))$ will be $val(1) - [t(X), t(X)]$. Without the global variables we get the value $val(1) - [t(X1), t(X2)]$ where the correspondence between two occurences of the input variable X is lost.

3.3. Monad list

The monad *list* can deal with nondeterministic expressions which return a list of answers.

```
unit(list,XP, [X]):- cpl2(XP,[X],[],G), call(G).
bind(list,[],_P,[]).
bind(list,[X|Xs],P,MY):- cpl2(P,[X,MX],[],G),
  call(G), bind(list,Xs,P,MXs), append(MX,MXs,MY).
zero(list,[]).
pluss(list,X,Y,Z):-append(X,Y,Z).
```

The predicate *unit* projects the value into the list. The operation *bind* evaluates successively the values in the second parameter using the predicate P. The specific operations are *zero* and *pluss*. The former denotes failing computation and the latter nondeterministic computation (in given order).

We extend the interpreted terms with the nondeterministic choice represented by the functor $amb/2$. The term *error* in the list monad means failing computation.

```
interp(T,amb(E1,E2),MN0):-
  interp(T,E1,MN1), interp(T,E2,MN2),
  pluss(T,MN1,MN2,MN0).
```

For example the evaluation of the term

$$amb(con(1), con(2)) + amb(con(30), con(40))$$

gives the result $[val(31), val(41), val(32), val(42)]$.

3.4. Monad state

The monad *state* is used for passing state through a computation. It enables to set and to get the value of the state. The result is a representation of a higher-order predicate, which expects two arguments. The first is the input value of the state and the second is the structure $Value - State$ with components the resulting value and the final state. The code computes the result and new state from the initial value of the state.

```
unit(state,XP,MN):- cpl2(XP,[N0],[],G),
  MN=l2([St,N0-St],[],G).
bind(state,MA,P,MB):-
  MA=l2([S1,X1-S2],Glob,G1),
  MB=l2([S1,X2-S3],Glob,G12 ),
  cpl2(P, [X1,l2([S2,X2-S3],Glob,G2)], [], G),
  call(G), G12 = (G1,G2).
set(state,S,l2([_,_-S],[], true)).
get(state, l2([S1,S1-S1],[],true) ).
```

The *unit* predicate creates a result in the form of the anonymous predicate with the $l2$ functor, which passes the state without a change. The *bind* predicate composes the representation of the previous computation $G1$ with the current one $G2$. The special predicates are $set/3$ and $get/3$. The *set* predicate besides the type gets a new value of the state S and sets it in the represented goal. The body of the goal in *set* is empty, as the new value is passed as a part of the parameter. The *get* predicate binds the result in the goal to the current value of the state and passes the state without change. We must extend the interpreter to be able to call the new predicates.

```
interp(T, set(R,E),MN0):-
  do( T, l2([N2],[N2],true),
     [_N1-MN1 <= [E]^set2(T,R,MN1),
      N2-MN2 <= [E,_N1]^interp(T,E,MN2) ], MN0).
interp(T, get,MN0):- get2(T,MN0).
```

The evaluation of the term

$$set(val(10), con(2) + get)$$

returns as the result the goal G:

```
      l2([X1,X2-val(10)],[], (X3=val(2), X2 is X3+val(10))).
```

Calling the goal G using $call(G, State, Result)$ we get $Result = val(12)-$ $-val(10)$. The initial value of *State* is not interesting, as the value is first set and then used.

The state monad passes only one state. But the value can be a structure, for instance a look-up dictionary indexed by names. So the state monad can actually pass several pieces of information.

3.5. Monad reader

The monad *reader* is used for passing context to a computation. It enables to fetch the passed value and to restore the value. Each monadic value is a representation of a higher-order predicate. It expects two parameters and computes the result in the second parameter using the initial value of the context given in the first parameter.

```
unit(reader,XP,MN):- cpl2(XP,[N0],[],G),
  MN=l2([_R,N0],[],G).
bind2R(reader,MA,P,MB):-
  MA=l2([R,X1],Glob,G1),
  MB=l2([R,X0],Glob, G12 ),
  cpl2(P, [X1,l2([R,X0],Glob,G2)], [], G),
  call(G), G12=(G1,G2).
fetch(reader,l2([R,R],[],true)).
restore(reader,R, l2([_,X],[],G),l2([_,X],[], G0) ):-
  cpl2(l2([_,X],[],G),[R,X],[],G0).
```

The *unit* predicate creates a result in the form of the anonymous predicate with the *l*2 functor, which do not use the context. The *bind* predicate composes the representation of the previous computation $G1$ with the current one $G2$. The special predicates are *restore*/4 and $fetch/2$. The $fetch$ predicate gets a value of the context R and returns it as the result in the represented goal. The body of the goal in $fetch$ is empty, as the value is passed through the parameters. The *restore* predicate returns the new goal in which the goal G is called in the context R. *Restore* renames G using *cpl*2 and creates new goal. In comparison to the state monad the new value of context is used only in the goal G and is not used outside.

We extend the interpreter and interpreted terms with expressions *restore* and $fetch$ to be able to call the new predicates.

```
interp(T, restore(R,E),MN0):-
```

```
  interp(T, E,MN1), restore(T,R,MN1,MN0).
interp(T, fetch,MN0):- fetch(T,MN0).
```

The evaluation of the term

$$restore(val(10), con(2) + fetch)$$

returns as the result the goal G:

```
          l2([X1,X2],[], (X3=val(10), X2 is X3+val(2) )).
```

Calling the goal G using $call(G, Context, Result)$ we get $Result = val(12) - Context$. The initial and final values of $Context$ are not relevant here and can be free, because the value of context is first set and then used.

As with the state monad, the reader monad passes only one context. But the value can be structure, so the reader monad can actually pass several pieces of information.

3.6. Representation of code

There are some variants of the representation of the higher-order predicates, which are results of monadic computations. The structure of the goal in the results is the same as the structure of the interpreted term. If we wish to compose the goals using comma only to the right side, we change a representation to difference goals. They are analogy to difference list and enable quick composing. The creation of the difference goal from the usual goal is hidden behind the interface consisting of predicates $mk\_goal/2$, $mk\_and/3$, and $calldg/1$ for a creation, a composition, and a call of the difference goal, respectively. During the composition we can also prune the *true* goals. Example of using the interface is the following code for $fetch3$.

The second possible change concerns the mode of the results in the second parameter. In some cases the value is known and is passed directly, as in $fetch1$. If we wish to generate code which passes the value, then we use the form of $fetch2$.

```
fetch1(reader,l2([R,R],[],true)).
fetch2(reader,l2([R,X],[],X=R)).
fetch3(reader,l2([R,X],[],G0)):- mk_goal(X=R,G0).
```

4. Conclusions

We have shown the implementation of a few monads and their integration to a small interpreter. We suggested a representation of higher-order predicates and introduced global variables to solve a problem with unwanted renaming. We have adapted do-notation known in functional programming to out representation.

Some problems were left. The first one is that the explicit notation for global variables is error-prone and annoying for a user. Some form of preprocessing is needed. The second problem which is not successfully solved in the theory is the composition of monads. The interpreter can be instantiated by a monad, but only by one. Combinations of monads can be very helpful, for instance using ideas from [3]. We were not interested in the efficiency of the solution up to now. How to use a partial evaluation for these specific interpreters is also left for future work.

Bibliography

1 Y. Bekkers, P. Tarau: *Monadic Constructs for Logic Programming*, ILPS '95, MIT Press, 1995, `http://clement.info.umoncton.ca/~tarau/`

2 E. Meijer, J. Jeuring: *Merging Monads and Folds for Functional Programming*, Advanced Functional Programming, Spring School, LNCS 925, Springer Verlag, Berlin, 1995

3 S. Liang, P. Hudak, M. Jones: *Monad Transformers and Modular Interpreters*, POPL'95, ACM Press, New York, 1995

4 [W92] P. Wadler: Essence of Functional Programming. *Annual Symposium of Principles of Programming Languages*, Albuquerque, New Mexico, 1992

Appendix: Additional monads

Monad Maybe

A generalization of the *maybe* monad is the *error* monad, which distinguishes a kind of raised error, usually as a string.

The predicate *pluss* is used in the maybe (and error) monad for catching exceptions.

Informační technologie a trvale udržitelný rozvoj[5]

Tomáš Pitner

Masarykova univerzita v Brně, Fakulta informatiky

Botanická 68a, 602 00 Brno, Česká republika

`tomp@fi.muni.cz`

Abstract: The aim of this paper is to present the impact of information technologies on the sustainable development.

1. Globální environmentální problémy dneška

Na úvod představme nejzásadnější globální environmentální problémy dneška, uveďme jejich příčiny, perspektivu a potenciální východiska. Pojem "environmentální problém" budeme přitom chápat v dnes obvyklém, tj. širším slova smyslu, zahrnujícím kromě problémů životního prostředí též otázky ekonomické, sociální, zdravotní a další. Pokud jde o závažnost jednotlivých problémů, je možné vycházet např. z výzkumu mezi 200 prestižními vědci z 50 zemí, prováděném při sestavování zprávy [GEO2000]. Více než polovina z nich považuje za problém č. 1 *globální klimatické změny*, následované s odstupem *nedostatkem pitné vody, odlesňováním, znečištěním vody, chudobou, ztrátou biodiverzity, populačním růstem a pohyby, změnou sociálních hodnot, hospodařením s vodou* a *znečištěním vzduchu.*

2. Co je trvale udržitelný rozvoj?

Se stále intenzivnějším globálním vlivem lidské činnosti na životního prostředí se ukázalo, že životní prostředí není výlučnou záležitostí ani individuálních osob, ani vlád jednotlivých zemí, ale dokonce celé mezinárodní komunity jako celku.

[5]Tento příspěvek vznikl za podpory grantu FRVŠ 0427/2001.

Izolované aktivity jednotlivých států v ochraně životního prostředí mají svá omezení a bylo třeba hledat mechanismy, jak identifikovat klíčové ekologické problémy celého lidstva, vytyčit strategii vztahu lidstva jako celku k životnímu prostředí a postupně tuto strategii realizovat.

OSN na sebe vzala úlohu světového koordinátora v otázkách životního prostředí a uspořádala roku 1992 v Rio de Janieru schůzku na nejvyšší (tedy ministerské) úrovni pod názvem *Earth Summit.* Tato konference ideově navazovala na první setkání ve Stockholmu roku 1972 a stala se jejím podstatným prohloubením. Závěrečné resumé tohoto setkání bylo zformulováno do několikasetstránkového dokumentu pod názvem *Agenda21* (viz [Ag21]). Jedná se o strategický plán, který identifikuje klíčové globální ekologické problémy lidstva a nabádá vlády států, jež pod Agendu připojily své podpisy, aby implementovaly myšlenky Agendy[6].

Celkové odpovědné chování člověka ve vztahu k jeho životnímu prostředí se označuje spojením *trvale udržitelný rozvoj (sustainable development).* Tím je míněno respektování přirozeného ekonomického růstu moderní civilizace a zároveň jeho usměrňování takovým způsobem, aby se růst neuskutečňoval na úkor budoucích generací.

3. Vliv IT na trvale udržitelný rozvoj

3.1. Pozitivní dopady IT

3.1.1. Změna modelů výroby

Díky masivnímu rozšíření a používání IT lze dosáhnout postupných změn v modelech výroby. Od moderní sériové průmyslové výroby maximálně unifikovaných výrobků "na sklad" se přechází k individuálně přizpůsobené výrobě:

a) *kdy je potřeba* (just-in-time)

b) *přesně podle individuálních potřeb* (just-for-you)

c) *přesné množství* (just-enough)

Lepší informovaností díky možnostem rychlé, levné a cílené komunikace lze také např. docílit úplnějšího využití nejrůznějších *rezerv* (přebytečných zdrojů) ať už jde o přebývající suroviny, materiály, energii, lidské zdroje atd.

[6]Ve 108 státech, které Agendu podepsaly, vzniklo na jejím základě 1800 *lokálních Agend21.*

Stejně jako přebytečné primární zdroje lze v globalizovaném světě vyměňovat i *znovupoužitelný materiál a suroviny* - tj. recyklovatelný odpad, chemikálie atd. nebo dokonce znovupoužívat celé části původního výrobku (rámy, pláště), pak se hovoří o tzv. *remanufacturingu*, viz [Lovins96].

Podstatných environmentálních zlepšení se můžeme dočkat s obecným prosazením automatizovaných systémů B2B elektronického obchodu, který skutečně může zredukovat manuální zpracování "papírové" administrativy a snížit tak její materiální náročnost, stejně jako posílit výše uvedené procesy racionalizace výroby.

3.1.2. Změna stylu práce

Klasickým příkladem pozitivního environmentálního dopadu využívání IT jsou *videokonference*, odstraňující nutnost cestování "abychom mohli partnerovi pohlédnout do tváře". Zatímco (viz [Lovins96]) environmentální zátěž průměrné zaoceánské cesty jedné osoby odpovídá 1 tuně, pak jedna šestihodinová videokonference "stojí" zhruba 10 kg na jednoho účastníka.

Celkový environmentální přínos zavádění těchto technologií je ovšem sporný a ne zcela prokazatelný - jednotlivou cestu je možné zredukovat, otázkou ovšem zůstává, zdali to (díky současné úspoře času a peněz) spíše nevyvolá další, jiné, cesty. Pozitivní efekt se ovšem jistě ukáže v okamžiku, kdy budou externality spojené s moderní dálkovou (zejména leteckou) dopravou internalizovány ve formě adekvátního environmentálního zdanění - pak se videokonference stanou nezbytností.

Obecně, fakt, že *teleworking*, *e-working, working from home* atd. prostřednictvím IT, je jednou zvýrazných charakteristik nastupující informační společnosti, je známý. Některé telekomunikační společnosti (např. British Telecom, viz `http://www.wfh.co.uk/wfh`) aktivně nabízejí programy na podporu tohoto způsobu práce.

Environmentální dopady e-workingu jsou obecně považovány za kladné, zejména pokud jde o úspory dopravních nákladů a tím nižší produkci skleníkových plynů a snížení lokálního znečištění především individuální dopravou. Pesimistické výhrady, že teleworking nakonec dopravní náklady nezredukuje (lidé se budou stejně chtít vidět) a energii neuspoří, se neukazují jako pravdivé - studie potvrzují (viz [INT2001]) asi 20% úspory.

Pokud jde o různé rozšíření e-workingu v různých zemích a regionech, zajímavé závěry přináší aktuální studie [Huws2001]: mezi nejvyspělejší země v tomto ohledu počítá USA, Německo, Austrálii, Francii, Velkou Británii, naproti tomu například Česká republika a Slovensko jsou (na rozdíl od Polska, Maďarska a

Slovinska) překvapivě počítány mezi "e-losers" bez velké perspektivy.

Poněkud jinak (optimističtěji) vyznívá statistika výzkumu mezi evropskými zaměstnavateli *eWork in Europe*, `http://www.emergence.nu/news/employer.html`, řadící Polsko, Maďarsko a ČR na první tři místa mezi dodavateli softwaru a softwarové podpory jak pro vlastní potřebu, tak pro EU.

Obecně platí, že "čistý" e-working přímo zdomácnosti se takřka nevyskytuje (dnes v EU cca 2%), populární jsou naopak různé kombinované formy, spočívající např. v práci v (i s dílených) vzdálených kancelářích, práci v *call-centru*, práci na více místech (*multi-locational working*) apod.

Často je e-working spojen s outsourcingem *(e-outsourcing)*, buďto klasickým, kdy partnerem je firma (v regionu či vzdálená), anebo individuální pracovníci "na kontrakt" (*e-lancers*) v těch odvětvích, kde byla práce "na volné noze" (freelance) populární již dříve.

V některých zemích (např. ČR) mohou popularitě *e-lancers* napomáhat i pracovněprávní a daňové předpisy, které zvýhodňují samostatně výdělečně činné osoby proti zaměstnancům - zaměstnavatel dá tedy přednost živnostníkovi na tzv. "švarcsystém" před zaměstnancem.

V zemích, kde jsou velmi "tvrdé" pracovněprávní předpisy (Velká Británie) a i klasická pracovní síla je tam daleko flexibilnější, viz např. `http://www.emergence.nu/news/growth.html`, je větší nárůst *zaměstnanců* (meziročně o 22%) před *e-lancers* (nárůst jen o 15%).

3.1.3. Změna modelů spotřeby

Význačným pozorovatelným trendem přechodu k informační společnosti (společnosti znalostí) je částečná *dematerializace výroby a spotřeby.* Stále větší podíl na produkovaných statcích mají statky nehmotné, především založené na informacích a znalostech a na jejich správě.

Tento vývoj má v zásadě dvě podoby - jednak je to *přímé nahrazení*, kdy je s příchodem nové IT technologie rovnocenně nahrazen dřívější materiální produkt novým - demateralizovaným. Příkladem je např. nahrazení klasického elektronického telefonního záznamníku jeho digitální podobou (zcela konkrétně např. služba *Memobox* Českého Telecomu), která fyzicky není ničím jiným než paměťovým prostorem na disku a příslušným obslužným softwarem. Tzv. *environmentální faktor* tohoto nahrazení se pohybuje od 20 (20x nižší hmotnost) do 240 (snížení emisí skleníkových plynů na 1/240). Dalším příkladem nahrazení je *e-mail*: zatímco materiálové vstupy pro vyprodukování a doručení 10g papírového dopisu činí 500 g, u e-mailu je ekvivalent asi 5 g.

Druhým paralelním trendem dematerializace je jakési "obrácení pozornosti" na nový nehmotný produkt - a tím zvýšení jeho podílu i v případě, že v absolutních číslem zůstává produkce původního materiálního výrobku nezměněná - tedy obvykle se *dematerializuje růst, původní hmotná produkce zůstává.* Zatím ve většině komodit jednoznačně převládá tento model, což bohužel ještě neznamená dostatečný obrat směrem k trvale udržitelného rozvoji.

Specifickým projevem dematerializace spotřeby je tendence *kupovat službu*, nikoli konkrétní výrobek - např. "zajištění dopravy podle potřeby" vs. nákup vlastního vozidla. Uvedené trendy mají svůj odraz i v sektoru IT - mnohem častěji firmy místo nákupu a správy prostředků IT vlastními silami volí outsourcing u *Application Services Providers (ASP)*, jež profesionálně zajišťují nejrůznější služby v oblasti IT.

Otázkou zůstává, kde až jsou ekonomické, právní, sociální, kulturní i psychologické meze posunu od tradičního *posesivního* vztahu ("koupím-vlastním-používám-udržuji-zahodím") ve prospěch ("průběžně platím za používání, o další se nestarám"), a ve kterých oblastech má tento posun největší rezervy. Jinak se těmto trendům postaví mladá generace vyrůstající v kulturně a sociálně vyspělé stabilní společnosti s vysokou vymahatelností práva a jinak v postkomunistickém "Divokém východě".

V současné - první - fázi informatizace společnosti je evidentním trendem nikoli pokles, ale *nárůst výroby* materiálních statků - zejména *technických prostředků IT* (počítačů, síťové infrastruktury). v budoucnu dá se očekávat - podobně jako tomu bylo s průmyslovou výrobou v éře klasické vědeckotechnické revoluci 20. století - postupná *intenzifikace* využívání prostředků IT. Intenzifikace může pozitivní environmentální efekt IT výrazně zvednout - environmentální zátěž IT je totiž daná (kromě spotřeby energie za provozu) především zátěží při výrobě. Studie (viz [Lovins96]) srovnávají běžné a intenzivní využití faxového přístroje a dokazují až *pětinásobný nárůst* environmentální efektivity.

Srovnejme např. běžné využití běžného kancelářského PC - s jedním počítačem se pracuje osm hodin denně, jen v pracovní dny a to jen v době, kdy je obsluha přítomna na pracovišti.

Příkladem intenzifikace je koncept *Net-Centric Computing*, kdy místo plnohodnotného PC stačí jednodušší (typicky bezdiskový) terminál připojený do sítě. Běžné realitě je ještě bližší systém *Multiuser PC*, kdy je jeden vcelku běžný kancelářský počítač virtuálně "rozmnožen" připojením dalších klávesnic, myší, videokaret a monitorů s příslušným softwarem až na čtyři téměř plnohodnotná kancelářská pracoviště.

Signifikantním ukazatelem posunu k intenzifikaci využití IT je momentální záporný výkyv poptávky po "klasických" PC při rostoucím zájmu o *mobilní zařízení.* Dá se očekávat, že silným impulsem pro intenzifikaci bude existence kvalitní a cenově dostupné mobilní datové sítě, jež může vzniknout např. na bázi UMTS (mobilních sítí třetí generace). Brzdicím faktorem vysokorychlostních mobilních sítí může být i vysoká cena za již prodané licence v řadě západoevropských zemí, jež byly u klasických GSM sítí tahounem světového vývoje.

3.1.4. Lepší informovanost

Informovanost v otázkách životního prostředí je podobně jako u jiných problémů základním předpokladem fungování demokratické společnosti s plnou odpovědností občanů za její vývoj.

V současnosti jsou standardní součástí legislativy vyspělých zemí a nadnárodních společenství (EU) zákony garantující svobodný přístup k environmentálním informacím. v evropském prostředí udávaly tón především země s dlouhou nepřerušenou demokratickou tradicí (UK, Švédsko), kde existují předpisy zajišťující obecný *svobodný přístup k informacím* shromažďovaným veřejnou správou již velmi dlouho. Pokud jde speciálně o environmentální informace, bývá legislativa ještě vstřícnější směrem k poskytování informací. Evropská Unie vydala v tomto smyslu poprvé v roce 1990 Direktivu č. 313/90.

Významným dalším krokem bylo přijetí tzv. Aarhuské úmluvy [Aarhus98] roku 1998, k níž se připojily i země mimo EU a úmluva se stala standardním měřítkem legislativy v oblasti práv na informace o ŽP.

Česká republika k úmluvě přistoupila a český zákon č. 123/1998 Sb., *o právu na informace o životním prostředí* a obecný zákon č. 106/1999 Sb., *o svobodném přístupu k informacím*[7] jsou potvrzením zákonných práv v této oblasti.

3.1.5. Lepší možnosti (participace na) rozhodování

S lepší obecnou informovaností souvisí také možnost přímo se aktivně podílet na rozhodovacích procesem ve veřejné správě. Co je často vytýkáno většině moderních demokracií je značná odtažitost rozhodovacích procesů od občanů.

Občan vyspělé země má obvykle explicitně zákonem dané právo *participace na "veřejném rozhodování"* ve věcech týkajících se životního prostředí. Např. v ČR má možnost (a v některých případech povinnost) vyjadřovat se k aktivitám, jež mohou mít vliv na životní prostředí, na základě zákona 100/2001 Sb. *o posuzování vlivů na životní prostředí.*

Pokud jde o "osobní" *rozhodování o tržním chování*, občanům napomáhají kromě obecné environmentální informovanosti a vzdělávání také systémy *Ecolabellingu* (značení ekologicky šetrných výrobků), jež jsou podpořeny např. direk-

[7]který ale nemění povinnosti dané zák. 123/98

tivou ECC 92/868. v ČR proces značení organizuje ČEÚ MŽP, `http://www.ceu.cz/esv`.

Zatímco na změny tržního chování mají IT vliv především jako zprostředkovatelé lepší informovanosti, pak pro zvýšení podílu občanů na rozhodování je nezbytná celková *informatizace veřejné správy*, zejména zavedení možnosti tzv. *one-stop-shop* elektronického přístupu ke službách veřejné správy.

Kromě toho je nezbytné budovat environmentální povědomí občanů cílenou environmentální osvětou a výchovou. Vyspělé státy přijímají programy environmentální osvěty a vzdělávání, např. vláda ČR přijala *Státní program environmentálního vzdělávání, výchovy a osvěty v České republice*, a na léta 2001-2003 má příslušný *Akční plán*, jenž se konkrétně projevil např. v grantovém financování podpory pre- a postgraduálního vzdělávání učitelů k zavádění environmentální výchovy na školách.

Velké možnosti má v této oblasti *e-learning*, učení spomocí IT, protože umožňuje daleký dosah výuky často poměrně specializovaných disciplín v oblasti ŽP, pro které není možné lokálně zajistit adekvátní kvalitu výuky (odborníků je málo, jsou vytížení).

3.2. Negativní dopady IT

Široká dostupnost prostředků IT bohužel *smazává pozitivní dopad* nižší materiálové a energetické náročnosti výroby a provozu moderních zařízení IT. Typickým příkladem je např. stagnující nebo rostoucí spotřeba papíru v zemích s rozvinutou IT infrastrukturou - dostupnost možnosti kvalitního a rychlého tiskového výstupu spotřebu papíru zvyšuje[8] .

Zvýšení materiální a energetické efektivity výroby přináší při konstantní ceně vstupů snížení reálných produkčních cen a tím potenciální *zvýšení spotřeby* vyráběného statku. Tak se může stát, že celková spotřeba zdrojů místo poklesu spíše mírně naroste, zejména v rozvíjejících se ekonomikách.

4. Shrnutí

Vliv informačních technologií na společnost je a zejména bude obrovský a dotýká se i oblasti životního prostředí:

[8]Opakuje se tak učebnicová situace po ropné krizi sedmdesátých let, kdy se výrobcům automobilů podařilo dosti výrazně snížit spotřebu každého jednoho vozu, ale díky jejich lepší dostupnosti (a nižší spotřebě) *narostl celkový počet aut* a tedy i jejich *celková spotřeba*.

- IT pravděpodobně posílí nerovnoměrnost dosavadního ekonomického a sociálního vývoje.
- IT na jednu stranu zefektivní komunikaci, výrobu, obchodování a administrativu, to však na druhou stranu způsobí lepší dostupnost levněji vyráběných produktů a tím jejich větší spotřebu.
- Nárůst životní úrovně v rozvojových zemích způsobí zvýšení světové spotřeby energie a surovin a nárůst znečištění.
- IT změní sociální a politické vazby ve společnosti, na jednu stranu umožní lepší veřejnou kontrolu, pružnější komunikaci sveřejnou správou a kvalifikovanější přímou participaci na rozhodování, na druhou stranu povede u části společnosti k izolovanosti do zájmových komunit ("nik“) a nezájmu o obecné problémy, až k psychickým problémům lidí.

Literatura

- [AG21] UNO: *Agenda 21*, MŽP SR, Bratislava, 1996 (slovenský překlad)
- [ISPT] Činčera, J.: *Internet a společnost na přelomu tisíciletí: (polo-) globální paradox*, VOŠ informačních služeb, Praha, 2000
- [ITZP] Činčera, J.: *Dopady informačních technologií na životní prostředí*, `http://www.ecn.cz/env/infos/01filos.htm`
- [GPSS] Činčera, J.: *Globální problémy současného světa*, `http://web.sks.cz/users/cn/zp/svet.html`
- [EMERG] Institute for Employment Studies: *The EMERGENCE Project*, `http://www.emergence.nu`
- [Huws2001] Huws, U. et al.: *Where the Butterfly Alights: the Global Location of eWork*, summary available at `http://www.employment-studies.co.uk/summary/378sum.html`
- [Lovins96] Lovins, A.B., Lovinsová, L. H.: *Faktor čtyři*, v českém překladu, MŽP ČR, Praha, 1996
- [OECDEO] OECD: *The OECD Environmental Outlook*, `http://www.oecd.org/env/outlook/outlook.htm`

- [DISS] Pitner, T.: *Analýza a návrh metadatového schématu pro environmentální data*, doktorská disertační práce, Masarykova univerzita vBrně, Fakulta informatiky, Brno, 1998
- [ITE] Swedish EPA: *IT and the environment*, `http://ww.internat.environ.se/documents/issues/technic/itenviro.htm`
- [EIICT] Telenor: *The environmental impact of ICT*, in Environmental Report of Telenor, Telenor, 2001
- [UKSDAR2001] UK Government: *Achieving a Better Quality of Life*, Government Annual Report, 2001, `http://www.sustainable-development.gov.uk/ann_rep/susdevel.pdf`
- [Aarhus98] UN/ECE: *Convention on Access to Information, Public Participation in Decision-making and Access to Justice in Environmental Matters* (Aarhus Convention), Aarhus, Denmark, 1998, `http://www.unece.org/env/pp/documents/cep43e.pdf`
- [GEO2000] UNEP: *GEO2000 - Global Environmental Overview*, `http://www.unep.org/Geo2000`

Orientované grafy jako nástroj systémové integrace

Jana Kohoutková

Masarykova univerzita v Brně, Ústav výpočetní techniky

Botanická 68a, 602 00 Brno, Česká republika

`kohoutkova@ics.muni.cz`

Abstrakt. V příspěvku jsou shrnuty hlavní rysy dvou aplikačních systémů, které k interní reprezentaci objektů a jejich vzájemných vztahů používají orientované grafy. V obou případech jsou grafy nástrojem k integraci modelů - v případě integrace datových modelů mají grafy pevně dány třídy uzlů a hran, v případě integrace obecných modelů jsou obecné i třídy uzlů a hran. Typy a funkce použitých grafů jsou formálně popsány abstraktními datovými typy.

Klíčová slova: modelování, integrace, grafy, abstraktní datové typy.

1. Motivace

U zrodu dvou projektů, o nichž budeme mluvit v tomto článku, stála shodná úvaha, že pro práci s objekty, které jsou vzájemně provázány komplikovanými vztahy, jsou vhodným nástrojem (*orientované*)*grafy*.

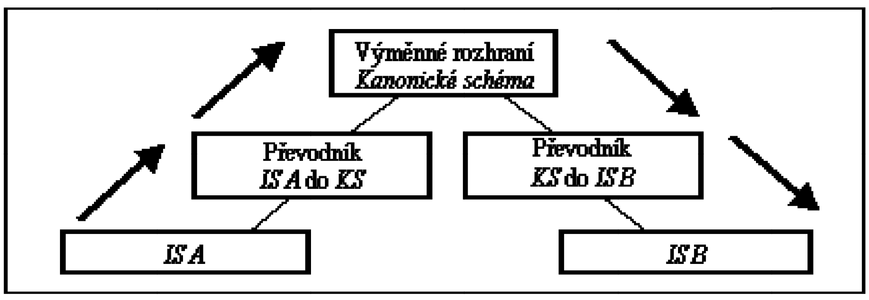
Cílem projektu HyperMeData (CP 94-0943, [1]) bylo vybudovat prostředí pro vzájemnou výměnu dat mezi nezávislými nemocničními informačními systémy a prezentaci těchto dat uživateli, cílem projektu StraDiWare (Copernicus 977132, [2]) je vybudovat prostředí pro integraci různých nástrojů podporujících analytika při tvorbě informačních strategií. V projektu HyperMeData se jednalo o *integraci datových modelů*, k jejichž interní reprezentaci byly použity orientované grafy s *pevně definovanými třídami uzlů a hran.* Projekt StraDiWare vyžaduje zobecnění tohoto přístupu směrem k univerzálnímu modelování (*integraci obecných*, nikoli pouze datových *modelů*), a proto jsou k interní reprezentaci objektů a vztahů použity orientované grafy s *obecnými třídami uzlů a hran.*

V následujícím textu přehledově popíšeme architektury aplikačních systémů HyperMeData a StraDiWare, shrneme hlavní rysy a způsoby využití grafů, jimiž jsou reprezentovány objekty a vztahy v těchto systémech, a na závěr typy a funkce použitých grafů formálně nadefinujeme pomocí abstraktních datových typů.

2. Integrace datových modelů cestou HyperMeData

2.1. Architektura systému

Základní architektura systému HyperMeData je zcela přímočará, jak ukazuje obrázek 1. Ve zvolené aplikační oblasti je navrženo standardní *kanonické výměnné schéma* a každý zúčastněný informační systém (IS) poskytne specifikaci svého exportního schématu. Speciálně navrženým formálním jazykem *DDL*, umožňujícím specifikovat jednak datová schémata, jednak mezischématové vztahy, jsou popsány jak kanonické schéma a exportní schémata všech IS, tak pravidla řídící procesy transformací datových instancí mezi schématy IS a kanonickým schématem. Schémata i specifikace transformačních pravidel jsou interně reprezentovány orientovanými grafy: grafy reprezentují struktury schémat i tok datových instancí výměnným systémem. Celý proces je prezentován uživateli přes hypermediální dokumenty.



Obr. 1. Transformace datových instancí z *IS A* do standardního výměnného rozhraní (*kanonického schématu - KS*) a dále do *IS B*

2.2. Jazyk DDL

Jazyk DDL (popsaný např. V [9]) používá deklarativní popisy pro datová schémata a funkcionální výrazy pro omezující podmínky uvnitř schémat a pro pravidla definující transformace datových instancí mezi dvojicemi schémat.

Definice dat v DDL

DDL kombinuje rysy tří úrovní datového modelování:*konceptuální* (popis entit, rolí, asociací, komplexních datových typů atd.), *logické* (každému konceptuálnímu objektu s atributy odpovídá množina instancí, tj. datových n-tic) a *intenzionální* (podmínkami je specifikováno, jak odpovídají konceptuální objekty logickým množinám n-tic).

Jazyk rozlišuje dva *objektové typy* - *entitu* a *asociaci*. Deklarace *entity* obsahuje podmínky definující vlastnosti instancí tohoto typu a volitelnou ISA klauzuli specifikující hierarchii entit a dědičnost atributů. *Asociace* je objektový typ definující vztah mezi entitami, jejími instancemi jsou n-tice instancí entit. Deklarace obsahuje *asociační predikát* (specifikující n-tice entit, jež jsou prvky asociace), *omezující podmínky* (definující vlastnosti vyhovujících instancí), *kardinalitní omezení* a v případě *komplexní asociace* i *seznam atributů*. Pro ilustraci uveďme (podrobněji viz [9]):

```
ENTITY superworkpl
  HAS pk_sup:integer; postcode:char[7]; city:char[25]; ... ...
  KEY pk_sup; UNDER pk_sup<>null; END
ENTITY workplace
  HAS pk_wpl:integer; postcode:char[7]; city:char[25]; ... ...
  KEY pk_wpl; UNDER pk_wpl<>null; END
ASSOC wpl_swpl CONN workplace[0,*], superworkpl[0,1]
  WHEN workplace.pk_sup==superworkpl.pk_sup; END
```

Funkcionální datové výrazy v DDL

Datové výrazy v DDL jsou vyhodnocovány nad instancí datového schématu a jsou založeny na funkcionálním jazyku (viz např. [5]), pro jehož použití jsou ideální předpoklady: nutná podmínka, aby všechny výrazy byly referenčně transparentní (tj. jejich hodnoty byly zcela nezávislé napořadí vyhodnocování), je automaticky splněna tím, že zdrojová schémata jsou read-only a konverzní systém je navržen tak, že již transformované instance cílového schématu se dále nemění ani nepoužívají.

Významným operátorem při zpracování seznamů ve funkcionálním jazyce je tzv. *ZF-výraz* podporující sofistikované iterace: v DDL se ZF-výrazy používají při konstruování speciálních datových výrazů založených nastruktuře datového schématu. Vedle běžných funkcí a operátorů pro numerickou aritmetiku a práci se seznamy bylo zavedeno několik speciálních operátorů pro přístup k datům v instancích schémat, např. operátory '$+>$' resp. '$->$', které k výskytu entity figurující v binární asociaci vracejí seznam výskytů z asociované entity resp. jeho první prvek.

Definice transformací v DDL

Transformace je proces, jímž se z instance *zdrojového schématu* vygeneruje instance *cílového schématu*, tj. ze seznamů instancí entit a asociací (a jejich atributů) ve *zdrojovém schématu* se vygenerují seznamy instancí *cílového schématu*. Proces transformace je řízen *transformačními pravidly*, která slouží k dekompozici transformace do bloků popisujících způsob, jímž se instance jedné *cílové entity* (nebo *cílové komplexní asociace*) konstruují z instancí jedné nebo více *zdrojových entit* (*asociací*); seznam instancí cílového objektu může být konstruován jako sjednocení výsledků více pravidel a každý zdrojový objekt může být použit ve více pravidlech.

Vedle specifikace *cílového objektu* a *zdrojových objektů* obsahuje transformační pravidlo *výběrové podmínky* (určující, které instance zdrojového seznamu mají být transformovány), *sekci přiřazení* (v níž transformační *výrazy* nad instancemi zdrojových objektů určují hodnoty každého z *atributů cílového objektu* - zapoužití funkcionálních datových výrazů zmíněných výše) a nepovinnou *sekci LET* (přiřazující výrazy nad zdrojovým schématem proměnným, jež mohou být referencovány ze sekce přiřazení). Pro ilustraci (podrobněji opět viz [9]):

```
BUILD superworkpl <- Workpl
  LET  a:=->Wpl_Adr; WHEN getSuperWpl(Workpl) = null;
  ASSIGN pk_sup:=gen\_id(); city:=a.City;
         postcode:=TransfPC(a.PostCode);   ... END
BUILD workplace <- Workpl
  LET  sw := getSuperWpl(Workpl); WHEN super<>null;
  ASSIGN pk_wpl:=gen\_id(); pk_sup:=gen\_FK(superworkpl,sw);
         city:=a.City; postcode:=TransfPC(a.PostCode);   ... END
FUNCTION getSuperWpl (wpl: TYPEOF Workpl): TYPEOF Workpl
  { (super JOIN SuperWorkpl | sub := wpl); }
```

2.3. Prezentační hyperdokument

Prezentační dokument v systému HyperMeData (popsaný např. V [3]) slouží k procházení a zobrazování instance datového schématu (modelu) - jednak struktury schématu, jednak instancí jednotlivých objektů, tj. entit a asociací. Prezentační systém je postaven na modelu dat, modelu dokumentu a transformačních vztazích mezi nimi - analogicky transformačním vztahům mezi dvojicemi datových modelů, popsaným v předcházející části 0. Jazyk popisu dokumentu je kombinací SGML (pro popis formátu dat) a DDL (pro popis obsahu dat a jejich vnitřní struktury). Součástí prezentačního dokumentu je i zobrazování jeho struktury v podobě grafu ([3]).

V případě HyperMeData objekty a vztahy dokumentového modelu přímo korespondují s objekty a vztahy datového modelu; transformační pravidla, popisující způsob, jímž se instance *cílového objektu* (*stránky dokumentu*) konstruují z instancí *zdrojových objektů* (*zdrojových entit* nebo *zdrojových asociací*), tedy obsahují pouze triviální přiřazení.

2.4. Grafová reprezentace

Vztahy mezi objekty ve schématech a dokumentech (strukturní definice) a transformační pravidla, obojí popsané v DDL, jsou v systému HyperMeData interně reprezentovány jako orientované grafy s rozlišenými typy uzlů a hran: grafy transformačních pravidel jsou "rozpjaty“ nad grafy schémat a dokumentů.

Typy uzlů i hran jsou v grafové reprezentaci HyperMeData pevně dány: typem uzlu se rozlišují entity od asociací, stránek a transformací, typem hrany asociační vztahy od ISA vztahů, hyperlinků (tj. mezistránkových vztahů v dokumentech) a transformačních vztahů (např. typ hrany vedoucí od asociace k asociovaným entitám je odlišný od typu hrany vedoucí od entity k jejím superentitám v ISA vztahu apod.).

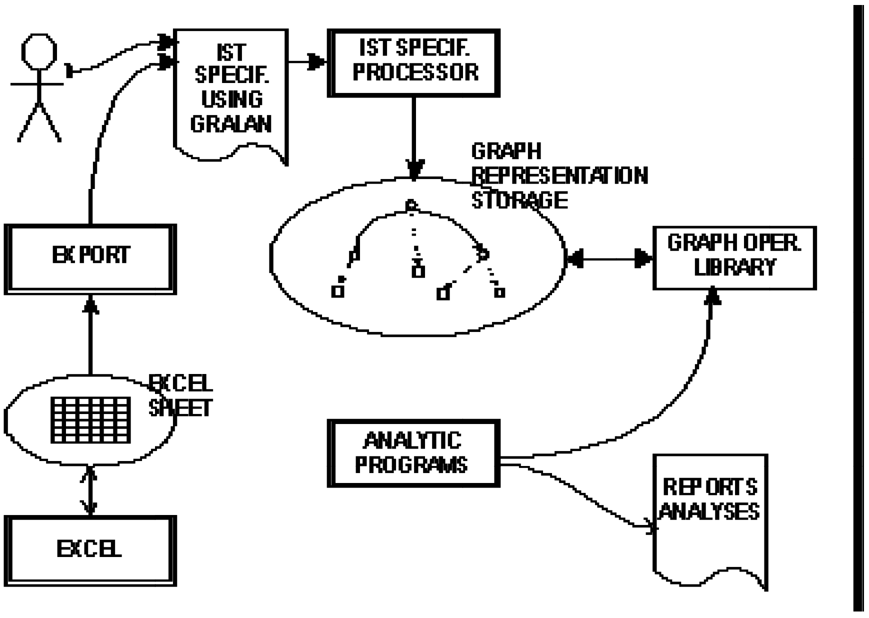
Grafová reprezentace se využívá především k hledání cest při vyhodnocování funkcionálních datových výrazů v transformačních pravidlech - konkrétně operátory $+>$ a $->$ používají grafy k nalezení asociovaných entit (jsou-li známy zdrojové entity a asociace) a k ověření, zda jsou konstrukce jednoznačné. Vedlejší využití mají interní grafy při zobrazování struktury prezentačního dokumentu: při konverzi popisu dokumentu do interní grafové reprezentace jsou objekty dokumentu (stránky) částečně uspořádány do *úrovní*, které jsou základem pro vykreslování snadno čitelného grafu.

3. Integrace obecných modelů cestou StraDiWare

3.1. Architektura systému

Jak ilustruje schéma naobrázku 2, jsou základem systému StraDiWare opět *orientované grafy* coby nástroj k zachycení objektů a jejich vzájemných (relativně složitých a proměnlivých) vztahů. Jsou uchovávány v *úložišti grafové reprezentace* a jsou složeny z uzlů a hran rozdělených dotypů odpovídajících typům objektů a vztahů identifikovaným v určité aplikační oblasti - např. oblasti informačních strategií (IST).

V termínech speciálně navrženého grafového jazyka *Gralan* se popíší (buď ručně, analytikem, nebo exportem & konverzí dat z existujících softwarových nástrojů) konkrétní výskyty objektů IST a vztahy mezi nimi. Tato *specifikace IST* zaznamenaná v grafové syntaxi se předloží *procesoru*, který ji převede do grafové reprezentace. S grafovou reprezentací pracují *analytické programy*, které využívají základní grafové operace uložené v *knihovně grafových operací*: vyhodnocují grafy, produkují přehledy, zprávy apod. Důležitou funkcí systému je, že analytické programy mohou *provádět kontroly korektnosti grafů* v grafové reprezentaci a tyto kontroly mohou vycházet (a zpravidla vycházejí) z potřeb konkrétního analytika a konkrétní IST.



Obr. 2. Architektura systému StraDiWare

3.2. Jazyk resp. knihovna Gralan

Od jazyka pro práci s grafy tedy StraDiWare vyžaduje:

1. podporu reprezentace orientovaných ohodnocených grafů, jimiž jsou modelovány objekty a meziobjektové vztahy (resp. typy objektů a objektově-typové nebo mezitypové vztahy) a které jsou vybaveny nástroji pro definici *správně utvořených grafů* (jednak ve smyslu typového chování jednotlivých objektů a vztahů, jednak ve smyslu grafového chování, řízeného speciálními pravidly vzájemných vztahů mezi grafovými objekty - např. pravidly *konektivity hran*, pravidly *strukturovanosti grafu* aj.);

2. podporu práce s těmito grafy (základní operace nad grafovými objekty, včetně operací ověřování korektnosti grafů);

3. vlastnosti běžného programovacího jazyka (typy, výrazy, řídící struktury atd.).

Ze dvou možných cest k tomuto cíli (tj. buď vývoje zcela nového programovacího jazyka s vestavěnou podporou grafů nebo obohacení některého vhodného, již existujícího jazyka o typy a operace podporující grafy), byla pro StraDiWare zvolena cesta druhá, která se soustřeďuje na samotnou reprezentaci a zpracování grafů a namísto vývoje ostatních částí jazyka využívá prostředků hostitelského jazyka.

Grafový jazyk *Gralan* (podrobně popsaný např. V [8]) je tedy realizován jako *grafová knihovna obohacující hostitelský programovací jazyk.* Ilustrativní příklad definic objektů z aplikační oblasti IST a vytvoření grafu, který reprezentuje data flow diagram a je zatížen pravidly ***konektivity hran*** (podrobně viz[8]), lze v syntaxi blízké syntaxi jazyka C++ (zdůvodu čitelnosti nikoli plně identické) zapsat takto:

```
class ISTObject {ISTObject(string name); ...}
class ExternalEntity : ISTObject {...}
...
class RelatesTo {...}
class DfdEdge : RelatesTo {...}
...
class ISTGraph : CorrectGraph<ISTObject, RelatesTo> {...}
ISTGraph ist=ISTGraph();
{ //data flow diagram:
  ExternalEntity person=ExternalEntity("Person");
  DataFlow request=DataFlow("Request");
  ...
  ist.putNode(person);
  ist.putNode(request);
  ...
  ist.putEdge(DfdEdge(),person,request);
  ... }
{ //pravidla konektivity hran:
  ist.addEdgeCheck( //podminky pro externi entitu
    ConnectRule("ExternalEntity",{"DataFlow"},"DfdEdge") );
  ist.addEdgeCheck( //podminky pro data flow
    ConnectRule("DataFlow",{"ExternalEntity",
        "BusinessFunct","BusinessEntity"},"DfdEdge") );
  ... }
```

Poznámka: predikát *ConnectRule* vyjadřuje implikaci "*je-li zdrojový uzel daného typu a hrana daného typu, pak cílový uzel musí být instancí typu z uve-*

dené množiny".

Z ukázky je zjevná potřeba vhodného formálního popisu grafových objektů a vztahů. V rámci StraDiWare byla proto navržena definice typu dokumentu (DTD) pro XML popis jak základních grafových objektů a vztahů, s nimiž pracuje knihovna Gralan, tak nadstavbových objektů univerzálního modelování (podrobněji viz [2]). Těmito prostředky (XML tagy a atributy) se popisuje jednak tzv. *doménizace*, tj. *instanciace zvolené aplikační oblasti*, konkrétně např. IST (popis definuje konkrétní typy objektů, jejich atributy, operace, jakým pravidlům podléhají atd.), a dále *konkrétní model ve zvolené aplikační oblasti*, tedy např. informační strategie určitého podniku nebo organizace (zápis instancí typů objektů, vztahů, pravidel, operací atd.).

3.3. Grafová reprezentace

Na rozdíl od HyperMeData nejsou v grafové reprezentaci StraDiWare typy uzlů a hran pevně dány: typem uzlu se opět rozlišují typy objektů a typem hrany typy vztahů, obojí však závisejí nazvolené aplikační oblasti a nelze je předem definovat.

Dalším důležitým požadavkem, který klade StraDiWare na grafovou reprezentaci nad rámec HyperMeData (kde díky pevně daným typům uzlů a hran obdobný požadavek nevzniká), je umožnit definovat a ověřovat již zmíněná *pravidla správně utvořených grafů*. Například v ukázce z předcházející části 0 reprezentujeme data flow model tak, že business funkce, externí entity, entity i data flow jsou uzly specifických typů, a požadujeme, aby hrany v grafu, jež není třeba typově rozlišovat, byly zatíženy *pravidly konektivity*, která vyjadřují, že z externí entity, entity nebo business funkce může vést data flow hrana pouze dodata flow a naopak z data flow může vést data flow hrana pouze do externí entity, entity nebo business funkce. Podmínky se mohou týkat i čistě grafových vztahů a mohou být značně složitější: můžeme například požadovat, aby graf neobsahoval kružnice a podobně.

Pro potřeby integrace modelů v systému StraDiWare byly proto navrženy obecnější grafy, než byly grafy použité pro HyperMeData - a ty nyní, ve zbývající části článku, formálně nadefinujeme.

4. Orientované grafy jako nástroj integrace

Zobecněním orientovaných grafů, používaných aplikačními systémy popsanými vpředchozích dvou kapitolách, jsou tzv. *podmíněné orientované pseudografy* (viz

[6], [7]): hrany v grafech jsou orientované, mezi dvěma uzly - i totožnými - může vést více hran a lze definovat *podmínky*, jež musí být v grafech splněny.

Podmíněný orientovaný pseudograf je čtveřice $G = (N, E, g, p)$, kde N je konečná neprázdná množina uzlů, E je konečná množina hran, g: $E - > N * N$ je incidenční funkce, která každé hraně přiřazuje uspořádanou dvojici uzlů (*zdrojový* a *cílový uzel hrany v grafu*), a p je grafový predikát, jenž musí v grafu trvale platit. *Vstupní* (*výstupní*) *hranové okolí* uzlu n v grafu g je množina všech hran majících v grafu g cílový (zdrojový) uzel n, *vstupní* (*výstupní*) *okolí* uzlu n v grafu g je množina zdrojových (cílových) uzlů všech hran majících v grafu g cílový (zdrojový) uzel n. *Podgrafem* grafu $G = (N, E, g, p)$ je graf $H = (N1, E1, g1, p1)$, kde $N1$ je podmnožinou N, $E1$ je podmnožinou E, $g1$ je restrikcí g na $E1 - > N1 * N1$ a $p1 => p$.

Typy a funkce podmíněných orientovaných pseudografů nyní přehledově popíšeme (sodkazem napodrobný popis uvedený v[8]) pomocí *abstraktních datových typů* (*ADT*, viz [4]). Použití čtyř sekcí popisu ADT nejprve ukážeme napříkladu základních ADT potřebných pro reprezentaci podmíněných orientovaných pseudografů - *seznamů*, *množin* a (*nepodmíněných*) *grafů*.

4.1 Seznamy, množiny a grafy

Sekce Typy

vyjmenovává množiny, které představují typy a jsou použity v deklaracích funkcí; typy mohou být *generické*, tj. parametrizovány jinými typy, a platí pro ně $X <> Y => A[X] <> A[Y]$ (kde $A[X]$ značí typ A parametrizovaný typem X).

Seznam resp. množina je generický typ parametrizovaný typem prvku (T). Graf je generický typ parametrizovaný typem uzlu (N) a typem hrany (E), v němž na typ uzlu ani na typ hrany nejsou kladeny žádné zvláštní požadavky, tj. mohou být jak primitivní (*integer*, ...), tak složitější objektové, definované v rámci aplikační oblasti:

List[*T*]

Set[*T*]

Graph[*N*, *E*]

Poznámka: Zápis $A : B$ bude v dalším textu znamenat, že ADT A je dědicem ADT B (tj. typ A je podmnožinou typu B, jsou na něj aplikovatelné všechny funkce zB a platí pro něj všechny axiomy a vstupní podmínky zB). Zápisem $A : B, C$ bude vyjádřena násobná dědičnost, tj. že A je dědicem jak B, tak C.

Sekce Funkce

deklaruje operace, jež lze s prvky typů provádět.

List[*T*] je seznam prvků typu *T*, v němž se hodnoty mohou opakovat; lze vytvářet jeho instance (konstruktorem *nil*), v nich přidávat prvky na začátek (*cons*), získávat nebo odebírat prvky (*head* a *tail*) a zjišťovat prázdnost (*null*) nebo délku (počet prvků *length*). *Set*[*T*] je obecná množina prvků typu *T*; lze vytvářet její instance (konstruktorem *empty*) a v nich přidávat (*put*) a mazat (*remove*) prvky, testovat přítomnost prvků (*contains*), zjišťovat prázdnost (*is-empty*) nebo kardinalitu (*size*). Dále lze získávat seznam prvků množiny (*members*), provádět základní množinové operace (*union*, *intersect*, *difference*) nebo ověřovat inkluzi dvou množin (*subset*).

Graph[*N*, *E*] je orientovaný pseudograf s uzly typu *N* a hranami typu *E* (uzly i hrany mohou být sdíleny více grafy, typicky v případě grafu a podgrafu). Lze vytvářet instance grafu (konstruktorem *emptyGraph*), v nich přidávat a mazat uzly a hrany (*putNode*, *putEdge*, *remNode*, *remEdge*) a získávat množiny všech uzlů resp. hran (*nodes*, *edges*), zdrojové resp. cílové uzly hran (*src*, *dst*) a vstupní resp. výstupní hranová okolí uzlů (*incomings*, *outgoings*). Býti podgrafem je relace mezi dvěma grafy, kterou lze ověřit (*subgraph*) - neexistuje tedy žádný speciální typ, jenž by odlišoval grafy a podgrafy. Příklady definic funkcí grafu:

putEdge: *Graph*[*N*, *E*] * *E* * *N* * *N* −/ > *Graph*[*N*, *E*]

incomings: *Graph*[*N*, *E*] * *N* −/ > *Set*[*E*]

subgraph: *Graph*[*N*, *E*] * *Graph*[*N*, *E*] − > *Boolean*

Poznámka: Zápis *name*: *A* −/ >*B* značí parciální funkci *name* (s parametrem typu *A* a návratovou hodnotou typu *B*), přiřazující hodnotu jen těm prvkům množiny *A*, které splňují podmínky ze sekce vstupních podmínek. Zápis *A* **B* značí kartézský součin množin *A* a *B*. Jména funkcí jsou v některých případech přetěžována.

Sekce Axiomy

definuje logické podmínky, které pro uvedené typy vždy platí. Pro seznamy a množiny vyplývají z matematických definic množin, u grafů musí být konzistentní uzly spojované hranami, okolí těchto uzlů a rovněž predikát určující podgrafy. Příklady:

subset(a, b) =for all x from members(a) : contains(b, x)

for all e from incomings(g, n) : dst(g, e) = n

subgraph(g, h) => subset(nodes(g), nodes(h)) and for all e from edges(g) : contains(edges(h), e)and src(g, e) = src(h, e) and dst(g, e) = dst(h, e)

Sekce Vstupní podmínky

definuje logické podmínky, jež musí splňovat parametry parciálních funkcí, aby funkce vracely hodnotu. V případě seznamu resp. grafu jsou jimi například:

head(a: List[T])require not null(a) - seznam nesmí být prázdný

src(g: Graph[N, E], e: E) require contains(edges(g), e) - hrana musí být v grafu

incomings(g: Graph[N, E], n: N) require contains(nodes(g), n) - podobně uzel

putEdge(g: Graph[N, E], e: E, s: N, d: N) require contains(nodes(g), s) and contains(nodes(g), d) and (contains(edges(g), e) => src(g, e) = s and dst(g, e) = d) - podobně uzly, příp. konzistentně s hranou

3.5. Podmíněné grafy

Typy

Generický typ *CheckedGraph* (podmíněný graf) je specializací grafu a nedovolí takové modifikace, jež by způsobily nesplnění určující podmínky. Podmínka *Predicate* je abstraktní typ (*T − > Boolean*), který definuje pouze funkci pro vyhodnocení, neurčuje však, o jakou konkrétní funkci se jedná. *Conjunction* představuje celou množinu predikátů, jež musí být splněny současně. *ForAll-Nodes* (*ForAllEdges*) je predikát pro graf, který je splněn tehdy, je-li určitý predikát pro graf a uzel (hranu) splněn pro všechny uzly (hrany) v grafu.

CheckedGraph[N, E] : Graph[N, E]

Predicate[T]

Conjunction[T] : Predicate[T], Set[Predicate[T]]

ForAllNodes[N, E] : Predicate[Graph[N, E]]

ForAllEdges[*N*, *E*] : *Predicate*[*Graph*[*N*, *E*]]

Typ *Conjunction* umožňuje nadefinovat, že má graf splňovat více podmínek, a typ *ForAllNodes* (*ForAllEdges*) umožňuje zkonstruovat podmínku pro celý graf pomocí podmínky pro jeho jednotlivý uzel (hranu). Speciálním příkladem je *pravidlo pro konektivitu hrany*: z predikátu, který rozhoduje, zda je spojení jedné hrany a jejího počátečního a koncového uzlu v grafu přípustné, se zkonstruuje predikát *ForAllEdges*, kontrolující, zda jsou všechny hrany v grafu v pořádku, a ten se zařadí do množiny *Conjunction*, jež je omezující podmínkou grafu.

Poznámka: Přípustnost spojení hrany a uzlů lze vyhodnocovat na základě různých aplikačně závislých kritérií, například na základě podtypu dané hrany a uzlů nebo na základě hodnot nějakých funkcí pro uzly a hrany (například funkce vracející "barvu“ uzlu či hrany).

Funkce

Podmíněný graf má konstruktor (*emptyCheckedGraph*, parametrem je predikát typu *Graph*[*N*, *E*] − > *Boolean*), funkci vracející jeho nepodmíněnou variantu (*unchecked*), funkci vracející jeho podmínku (*constraint*) a funkci převádějící obecný graf na graf podmíněný daným predikátem (*makeChecked*). Funkce vložení resp. vyjmutí uzlu nebo hrany (*putNode*, *putEdge*, *remNode*, *remEdge*) jsou u podmíněného grafu parciální. Typ *Predicate* definuje pouze funkci pro vyhodnocení predikátu nad daným objektem (*check*), typy *ForAllNodes* a *ForAllEdges* mají konstruktory. Příklady:

makeChecked: *Graph*[*N*, *E*] * *Predicate*[*Graph*[*N*, *E*]] −/ > *CheckedGraph* [*N*, *E*]

constraint: *CheckedGraph*[*N*, *E*] − > *Predicate*[*Graph*[*N*, *E*]]

check: *Predicate*[*T*] * *T* − > *Boolean*

newForAllNodes: *Predicate*[*Graph*[*N*, *E*] * *N*] − > *ForAllNodes*[*N*, *E*]

Axiomy (několik vybraných příkladů)

constraint(*emptyCheckedGraph*(*p*)) = *p* - podmínka grafu

constraint(*makeCheckedGraph*(*g*, *p*)) = *p* - podmínka grafu

unchecked(*makeCheckedGraph*(*g*, *p*)) = *g* - nepodmíněný graf *check*(*constraint*(*g*), *g*) - trvalá platnost podmínky grafu

p: *Conjunction*; *check*(*p*, *x*) <=> *for all q from p* : *check*(*q*, *x*) - splnění konjunkce

check(*newForAllNodes*(*p*), *g*) <=> *for all n from nodes*(*g*) : *check*(*p*, (*g*, *n*))

check(*newForAllEdges*(*p*), *g*) <=> *for all e from edges*(*g*) : *check*(*p*, (*g*, *e*)) - konzistentní definice

Vstupní podmínky

Funkce *unchecked* slouží k formulování podmínek, jež musí být splněny, aby mohl být podmíněný graf modifikován. Na nepodmíněné variantě grafu lze modifikující operaci "předem vyzkoušet":

emptyCheckedGraph(*p*: *Predicate*[*Graph*[*N*, *E*]]) *require* *check*(*p*, *newGraph*) - podmínku musí splňovat prázdný graf ...

makeCheckedGraph(*g*: *Graph*[*N*, *E*], *p*: *Predicate*[*Graph*[*N*, *E*]]) *require* *check*(*p*, *g*) - ... i graf, z něhož se vytváří podmíněná varianta

putNode(*g*: *CheckedGraph*[*N*, *E*], *n*: *N*)

require *check*(*constraint*(*g*), *putNode*(*unchecked*(*g*), *n*)) - podmínka musí být splněna i po vložení uzlu (podobně pro *putEdge*, *remNode* a *remEdge*)

5. Souvislosti a další práce

V článku jsme shrnuli základní charakteristiky dvou systémů, představujících dva přístupy k řešení problému systémové integrace, a zaměřili se na princip, který je jim společný: reprezentaci objektů a meziobjektových vztahů vhodnými třídami orientovaných grafů. Řada zajímavých otázek, jež jsou předmětem další práce, zůstala mimo rozsah článku - za všechny uveďme například otázku využití grafové reprezentace pro grafové porovnávání (analýzu) datových schémat s cílem automatizovaně generovat transformační pravidla.

Autorka článku děkuje programu EU-Copernicus za poskytnutí prostředků, které umožnily práci na obou zmíněných projektech a vytvářejí základ pro návaznou teoretickou práci i praktické aplikace.

Literatura

1. *CP 940943 HyperMeData* (interní dokumentace k projektu). HyperMeData Consortium, 1998.

2. *CP 977132 StraDiWare* (interní dokumentace k projektu). StraDiWare Consortium, 2001.

3. Kohoutková, J., Svoboda, A.: *Hypermédia: homogenní rozhraní k heterogenním systémům.* In: Proc. of MEDSOFT'99, Praha, 1999.

4. Meyer, B.: *Object-Oriented Software Construction* (2nd ed.). Prentice Hall, 1997.

5. Paton, N., Cooper, R., Williams, H., Trinder, P.: *Database Programming Languages.* Prentice-Hall, 1996. Chapter 4: *Functional Data Languages.*

6. Plesník, J.: *Grafové algoritmy.* Slovenská akadémia vied, Bratislava, 1983.

7. Rozenberg, G., Salomaa, A.: *Handbook of Formal Languages.* Springer, Berlin, 1997.

8. Skoupý, K., Benešovský, M., Kohoutková, J., Ocelka, J.: *Gralan: grafová knihovna propodporu univerzálního modelování.* In: Proc. of DATASEM '2001, Brno, 2001.

9. Skoupý, K., Kohoutková, J., Benešovský, M., Jeffery, K.G.: *HYPERMEDATA Approach: A Way to Systems Integration.* In: Proc. of ADBIS'99, Maribor, 1999.

Large Relationship between probability measure and metric in the sample space

Zdeněk Fabián

Institute of Computer Science, Academy of Sciences of the Czech republic, Pod Vodárenskou věží 2, 18200 Prague

zdenek@cs.cas.cz

1. The problem

Let us denote by S an open, finite or infinite interval of the real line R, $\mathcal{B}$ Borel sets of S and P probability measure absolutely continuous with respect to the Lebesgue measure λ on R. The probability space $\mathcal{P} = (S, \mathcal{B}, P)$ serves as a model of continuous random variables and it is both simple and very useful. S is the sample space of distribution P of a random variable X, say, which is completely described by the distribution function $F(x) = P(X < x)$ or by the density $f(x) = dF(x)/dx$. We will suppose that

$$\begin{aligned} f(x) &> 0 \qquad x \in S \\ f(x) &= 0 \qquad x \in R - S \end{aligned}$$

so that S is the support of distribution P. For simplicity reasons suppose f regular in the sense of existence of needed derivatives.

On the other hand, various metrics ρ can be introduced in S to obtain various metric spaces (S, ρ).

There is the following problem: *Which metric ρ in (S, ρ) is to be preferred with respect to a given P ?* Or, in other words, which metric can be considered as generated in S by P?

This is apparently a pure mathematical problem. It has not been, surprisingly, solved or even generally posed in the probability theory. On the other hand, this problem is daily solved in statistics. Many of the statistical estimation procedures are nothing else than attempts to find unknown distributions of continuous random variables having observed data (samples from unknown distributions) and some vague ideas about the geometry of the sample space. However, in many practical cases some information about the family of possible distributions of the studied random variables are available. A relationship between the distribution and the geometry of the sample space is therefore of uttermost importance.

In this paper we present a solution. It is based on the recently introduced concept of the core function of the distribution. Unlike the usual course of matters, when statistics uses the probability theory for judgements and inspiration, we use the statistical experience to solve the problem of mathematical nature.

2. The solution for $S = R$

Glivenko-Cantelli theorem states that the empirical distribution function F_n of random sample $\mathcal{X}_n = (x_1, ..., x_n)$ taken from distribution P converges to the true F when $n \to \infty$. Usually, n is not sufficiently large and the parametric approach is used: the true distribution P is assumed to be a member of a parametric family $\{P_\theta, \theta \in \Theta\}$, $\Theta \subset R^m$, known apart from the unknown parameter $\theta = (\theta_1, ..., \theta_m)$, which is to be estimated.

Let us consider distribution Q with support $S_Q = R$. The density g of Q necessarily goes to zero when $y \to -\infty$ and $y \to -\infty$, and $g(y)$ has maximum at inner point y^*, say. To estimate the location of y^* from data, the density is written in a parametric form $g(y-\mu) = \tilde{g}(y-\mu, \theta_2, ..., \theta_m)$ where μ is a location parameter.

Statistical experience says that from the set of various estimators, the M-estimators are the most useful. Let ψ be a real function with property $\psi(0) = 0$. The $M-$estimate $\hat{\mu}_n$ of μ based on observed values $(y_1, ..., y_n)$ is a solution of equation

$$\sum_{i=1}^{n} \psi(y_i - \hat{\mu}_n) = 0. \tag{1}$$

Equation (1) actually gives the distance of points y_i and $\hat{\mu}_n$ in the form $d(y_i, \hat{\mu}_n) = |\psi(y_i - \hat{\mu}_n)|$. The choice of function ψ thus determines the distance between points $y_1, y_2 \in S_Q$ as $d(y_1, y_2) = |\psi(y_2 - \hat{\mu}_n) - \psi(y_1 - \hat{\mu}_n)|$. If $\hat{\mu}_n$ is consistent (which means $\hat{\mu}_n \to \mu$ for $n \to \infty$), the M-estimator (1) introduces in S_Q the distance

$$d_\psi(y_1, y_2) = |\psi(y_2 - \mu) - \psi(y_1 - \mu)|.$$

From $M-$estimators, the *maximum likelihood* estimator (MLE) has an exclusive position due to its efficiency. Although the maximum likelihood estimates may be non-robust and does not necessarily represents the optimal option when dealing with real data, it is undoubtedly an estimator relevant to the non-contaminated random sample from P. Consider a general location and scale family $g(u) = \tilde{g}(u, \theta_3, ..., \theta_m)$ where

$$u = \frac{y - \mu}{\sigma},$$

$\mu \in R$, $\sigma, \theta_2, ..., \theta_m \in (0, \infty)$. The $\psi-$function of the maximum likelihood estimator for μ (with other 'nuisance' parameters known) is the *likelihood score for location*,

$$\psi^g_{ML(\mu)}(y) = \frac{\partial}{\partial \mu} \ln g(u). \tag{2}$$

It holds that

$$\frac{\partial}{\partial \mu} \ln g(u) = \frac{1}{g(u)} \frac{d}{du} g(u) \frac{\partial u}{\partial \mu} = \frac{1}{\sigma} T_g(u) \tag{3}$$

where

$$T_g(u) = -\frac{g'(u)}{g(u)} \tag{4}$$

is the well known score function of distribution Q. Since the name score function is sometimes understood as the whole likelihood score for location (2), we call its inner part (4) the *core function.*

The *core distance*

$$d_{T_g}(y_1, y_2) = |T_g(y_2) - T_g(y_1)| = \int_{y_1}^{y_2} \rho_Q(y)\, dy, \tag{5}$$

where $\rho_Q(y) = T'_g(y)$, is our solution to the problem. If T_f is continuous and strictly increasing (it holds for many model distributions), ρ_Q defines in S_Q the Riemannian metric corresponding to Q. If not (example: the Cauchy distribution), further considerations are needed (perhaps the total variation of T_f can be used).

3. The solution for $S \neq R$

The solution given in the foregoing paragraph is simple and incontestable, but it is not general. For continuous distributions P with support $S_P \neq R$ the problem is more complicated. Distributions defined on the real halfline or interval may not have a maximum and cannot be provided by the location parameter. Their density may be even unbounded and their score functions (4) are odd. Let us give a simple example. The exponential distribution with support $S = (0, \infty)$ and density $f(x) = e^{-x}$ has score function $-f'(x)/f(x) = 1$, which is clearly of no use for any distance considerations. The conclusion was that the approach of the foregoing paragraph cannot be used for distributions with support $S \neq R$.

However, it is not the mathematical form (4), which is of interest but the fact, that T_g expresses the influence of points $y \in S_Q = R$ with respect to the 'centre of gravity' of distribution Q. The integral $\int_R T_g(y)g(y)\, dy = 0$, so that the point $y = 0$ (or, in a parametric case, $y = \mu$) is the 'centre of gravity' in the geometry given by (5).

The problem is which point $x^* \in S_P$ can represent a 'centre of gravity' of a distribution with support $S_P \neq R$? Mean and mode may not exist and median has no geometrical sense.

The following procedure was suggested in [1]. Consider P as a distribution $\varphi-$*induced* by some *source distribution* Q, which means $P = Q\varphi$ where $\varphi : S_P \to S_Q$ be a fixed one-to-one mapping. If the density of Q is g, the density of the $\varphi-$induced distribution $P = Q\varphi$ is clearly

$$f(x) = g(\varphi(x))\dot{\varphi}(x) \tag{6}$$

where $\dot{\varphi}(x) = d\varphi(x)/dx$ is the Jacobian of transformation $\varphi : S_Q \to R$. The idea is fifty years old (c.f. Johnson (1949)), but it was used only for derivation of the *related* families from some source families. In [1] Johnson's approach is used consistently: all regular continuous distributions are divided into disjunct sets of related distributions, each set consisting of the source distribution Q with support $S_Q = R$ and the induced distributions $P = Q\varphi$ with supports $S_P = (a, b) \subset R$, where $-\infty \geq a < b \leq +\infty$, by the use of mapping

$$y = \varphi_{ab}(x) = \ln \frac{(b - x_0)(x - a)}{(x_0 - a)(b - x)}, \tag{7}$$

which reduces, in cases of supports $S_P = (0, \infty)$ and $S_P = (0, 1)$, to Johnson transformations

$$\varphi_{0\infty}(x) = \ln x \qquad \varphi_{01}(x) = \ln \frac{x}{1 - x}. \tag{8}$$

It has been shown in [2] that with few exceptions all currently used model distributions on various $S_P \neq R$ can be considered as $\varphi_{ab}-$induced by the use of the generalized Johnson's mapping (7). Denote the set of these distributions by $\mathcal{J}$ and write $\varphi = \varphi_{ab}$.

Now, any distribution $P \in \mathcal{J}$ with support $S \neq R$ and density f can be considered to be φ–induced by some source distribution Q with density g. By (4) we can derive T_g and construct its 'image' on S_P.

DEFINITION 1 *The function*

$$T_f(x) = T_g(\varphi(x)). \tag{9}$$

is called a core function of distribution P.

It is important that T_f can be expressed independently of the source distribution by means of density f and of the Jacobian of the transformation.

THEOREM 1 *The core function of distribution P with support S_P is*

$$T_f(x) = \frac{1}{f(x)}\frac{d}{dx}(-f(x)/\dot{\varphi}(x)). \tag{10}$$

PROOF. Put $y = \varphi(x)$. By (9) and (6)

$$T_f(x) = T_g(\varphi(x)) = -\frac{1}{g(y)}g'(y) = \frac{\dot{\varphi}(x)}{f(x)}\frac{d}{dx}(-f(x)/\dot{\varphi}(x))\frac{dx}{dy}$$

and $(dy/dx) = \dot{\varphi}(x)$. □

The equivalent of the 'centre of gravity' of the source distribution has been defined in [2] as its 'image' on S_f, i.e. $x^* = \varphi^{-1}(y^*)$ or, in a parametric case,

$$\tau = \varphi^{-1}(\mu) \tag{11}$$

which we call a *Johnson location.*

EXAMPLE. Let $S_P = (0, \infty)$. Then $\varphi(x) = \ln x$, $\dot{\varphi}(x) = 1/x$, $T_f(x) = (-xf(x))'/f(x) = -1 - xf'(x)/f(x)$ and $\tau = e^{\mu}$. The exponential distribution $f_\tau(x) = \tau^{-1}e^{-x/\tau}$ has source distribution $g_\mu(y) = e^{y-\mu}\exp(-e^{y-\mu})$ with $T_g(y) = e^{y-\tau} - 1$. By (9), $T_f(x) = x/\tau - 1$, which reduces when $\tau = 1$ to the core $T_f(x) = x - 1$ of $f(x) = e^{-x}$.

Parametric distributions had originated during the historical development. Some of the distributions on $S_P \neq R$ have a Johnson location parameter and some of them not. In the case that parametric distribution f_θ has parameter τ (i.e., $\theta = (\tau, \sigma, \theta_3, ..., \theta_m), \tau \in S_P$), the following theorem holds.

THEOREM 2 *Core function is the inner part of the likelihood score for τ.*

Proof. Let $g(u)$, $u = (y - \mu)/\sigma$ be the source density of $f_\theta(x)$. By (6), (8) and (11) $f_\tau(x) = g(w)\dot{\varphi}(x)$ where

$$w = \frac{\varphi(x) - \varphi(\tau)}{\sigma}.$$

By (6), likelihood score for τ is

$$\psi^f_{ML(\tau)}(x) = \frac{\partial}{\partial \tau} \ln f_t(x) = \frac{\partial}{\partial \tau} \ln(g(w)\dot{\varphi}(x)) = \frac{g'(w)}{g(w)} \frac{\partial w}{\partial \tau} = \frac{\dot{\varphi}(\tau)}{\sigma} T_g(w). \quad (12)$$

$\square$

We obtained the same result as in paragraph 2: The core function is the inner part of the likelihood score for the 'centre of gravity' $\tau = \varphi^{-1}(\mu)$ or $x^* = \varphi^{-1}(y^*)$ of distribution P. Thus the core functions are known functions (more precisely: the inner parts of known functions) for distributions which have parameter τ and were unknown for distributions without τ or without parameters at all.

Definition 2 *The core distance in the sample space S_f of distribution P is given by*

$$d_{T_f}(x_1, x_2) = |T_{f_\theta}(x_2) - T_{f_\theta}(x_1)| = \int_{x_1}^{x_2} \rho_{f_\theta}(x)\, dx, \quad (13)$$

where $\rho_{f_\theta} = T'_{f_\theta}$.

Using the same reasoning as in the foregoing paragraph, (13) is the general solution to our problem for all distributions $P \in \mathcal{J}$. If T_f is continuous and strictly increasing, (13) defines a metric.

4. Examples

To illustrate our result by examples, some currently used distributions were selected. Their densities, core functions and the corresponding core distances $d_P(x_1, x_2) = |T_f(x_2) - T_f(x_1)|$ in the sample space S_P are given in Table 1. The couples of source and the induced distributions in Table 1 are normal and lognormal, Gumbel and Weibull (in both cases $\beta = 1/\sigma$) and logistic and log-logistic. Note that in different S_P there exist different distributions with Euclidean core distances : the normal distribution on $S_P = R$, the gamma (and exponential) distribution on $S_P = (0, \infty)$ and the beta (and uniform) distribution on $S_P = (0, 1)$.

TABLE 1. Densities $f(x)$, core functions $T_f(x)$ and core distances $d_T(x_1, x_2)$ of points $x_1, x_2 \in S_P$ for some currently used probability distributions.

Name	S_P	$f(x)$	$T_f(x)$	$d_P(x_1, x_2)$
Normal	R	$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{(x-\mu)}{\sigma}\right)^2}$	$\frac{x-\mu}{\sigma}$	$(x_2 - x_1)/\sigma$
Lognormal	$(0, \infty)$	$\frac{1}{\sqrt{2\pi}\ \sigma x} e^{-\frac{1}{2}\ln^2(x/\tau)^\beta}$	$\ln\ (x/\tau)^\beta$	$\ln(\frac{x_2}{x_1})^\beta$
Gumbel	R	$e^{\frac{x-\mu}{\sigma}} e^{-e^{\frac{x-\mu}{\sigma}}}$	$e^{\frac{x-\mu}{\sigma}} - 1$	$e^{-\mu/\sigma}(e^{x_2/\sigma} - e^{x_1/\sigma})$
Weibull	$(0, \infty)$	$\frac{\beta}{x} \left(\frac{x}{\tau}\right)^\beta e^{-(x/\tau)^\beta}$	$\left(\frac{x}{\tau}\right)^\beta - 1$	$(x_2 - x_1)^\beta / \tau^\beta$
Extr. val. II	$(0, \infty)$	$x^{-2} e^{-1/x}$	$1 - 1/x$	$\frac{x_2 - x_1}{x_1 x_2}$
Logistic	R	$e^x/(1 + e^x)^2$	$\tanh(x/2)$	$\frac{\sinh((x_2 - x_1)/2)}{\cosh(x_1/2) \cosh(x_2/2)}$
Log-logistic	$(0, \infty)$	$1/(z + 1)^2$	$(z - 1)/(z + 1)$	$\frac{2(x_2 - x_1)}{(x_1+1)(x_2+1)}$
Lomax	$(0, \infty)$	$\alpha/(1 + x)^{\alpha+1}$	$(\alpha(x - 1)/(x + 1)$	$\frac{(\alpha+1)(x_2 - x_1)}{(x_1+1)(x_2+1)}$
Gamma	$(0, \infty)$	$\frac{\gamma^\alpha}{\Gamma(\alpha)} x)^{\alpha-1} e^{-\gamma x}$	$\gamma x - \alpha$	$\gamma(x_2 - x_1)$
Beta	$(0, 1)$	$\frac{1}{B(p,q)} x^{p-1}(1 - x)^{q-1}$	$(p + q)x - p$	$(p + q)(x_2 - x_1)$
Cauchy	R	$1/\pi(1 + x^2)$	$2x/(1 + x^2)$	?

Aknowledgements. The author thanks to I. Vajda for valuable comments. The work was supported by grant GA AV A1075101.

References

[1] Fabián, Z. (1997). Information and entropy of continuous random variables. *IEEE Trans. on Information Theory*, 43, 1080-1083.

[2] Fabián, Z. (2001). Induced cores and their use in robust parametric estimation. *Commun. in Statistics, Theory Methods*, 30, 3, 537-556.

[3] Johnson, N.L. (1949). Systems of Frequency Curves Generated by Methods of Translations. *Biometrika* **36**, 149-176.