Volume 11 Numbers 1—2

ACTA CYBERNETICA

Szeged, 1993

Information for authors: Acta Cybernetica publishes only original papers in English in the field of computer sciences. Review papers are accepted only exceptionally. Manuscripts should be sent in triplicate to one of the Editors. The manuscript must be typed double-spaced on one side of the paper only. For the form of references, see one of the articles previously published in the journal.

Acta Cybernetica, Vol. 11, No. 1-2, Szeged, 1993

A Tourist Guide through Treewidth

H. L. Bodlaender*†

Abstract

A short overview is given of many recent results in algorithmic graph theory that deal with the notions treewidth, and pathwidth. We discuss algorithms that find tree-decompositions, algorithms that use tree-decompositions to solve hard problems efficiently, graph minor theory, and some applications. The paper contains an extensive bibliography.

1 Introduction

In recent years, the notions 'treewidth', 'pathwidth', 'tree-decomposition', and 'path-decomposition' have received a growing interest. These notions underly several important and sometimes very deep results in graph theory and graph algorithms, and are very useful for the analysis of several practical problems.

In this paper, we give an overview of a number of these applications, and algorithmic results. In section 2 we give the main definitions. Applications of the notions discussed in this paper are given in section 3. In section 4 we explain the basic idea behind linear time algorithms on graphs with constant bounded treewidth. In section 5 we review some results that deal with graph minors. In section 6 we discuss algorithms that find 'suitable' tree- or path-decompositions.

It should be noted that the constant factors, hidden in the 'O'-notation can be quite large for several of the algorithms, discussed in this paper. In many cases, additional ideas will be required to turn the methods, described here, into really practical algorithms.

2 Definitions

In this section we give the most important definitions, with an example. The notions of treewidth and pathwidth were introduced by Robertson and Seymour [109,115].

*This works was partially supported by the ESPRIT Basic Research Actions of the EC under contract 7141 (project ALCOM II.).

†Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands, e-mail: hansb@cs.ruu.nl.

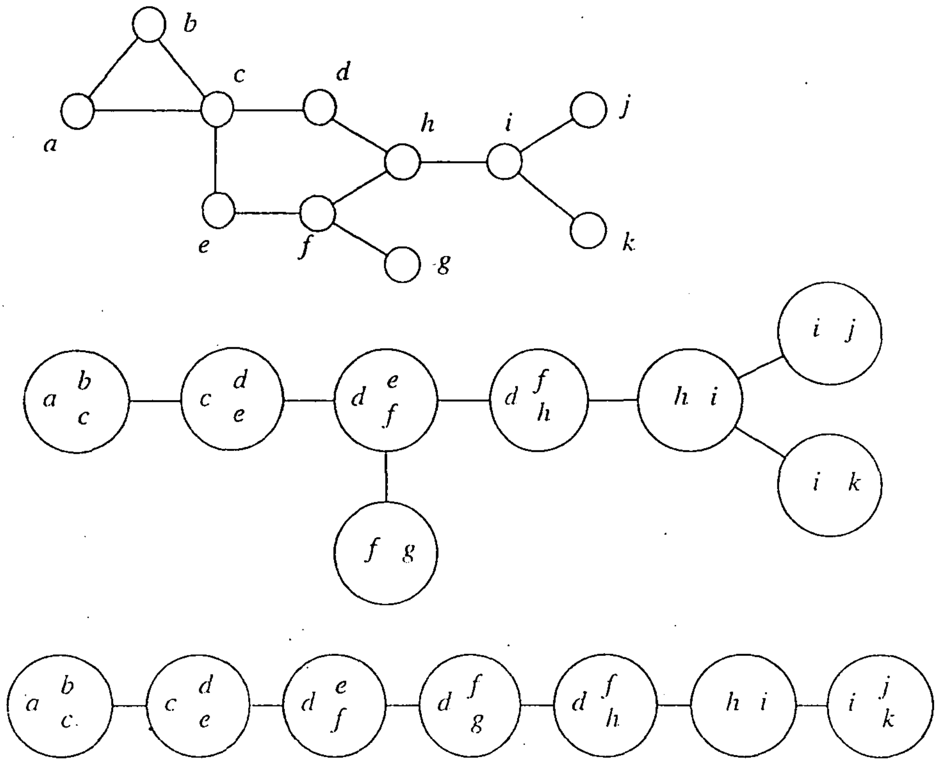


Figure 1.
Example of a graph with tree- and path-decomposition

Definition. A tree-decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a family of subsets of V, one for each node of T, and T a tree such that

- $\bigcup_{i \in I} X_i = V$.
- for all edges $(v, w) \in E$, there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.
- for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

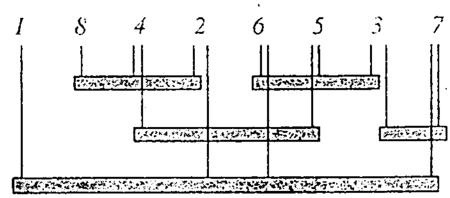
The treewidth of a tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph G is the minimum treewidth over all possible tree-decompositions of G.

The notion of pathwidth is defined similarly. Now T must be a path.

Definition. A path-decomposition of a graph $G = (V, E)$ is a sequence of subsets of vertices $(X_1, X_2, \ldots, X_r)$, such that

- $\bigcup_{1 \le i \le r} X_i = V$.
- for all edges $(v, w) \in E$, there exists an i, $1 \le i \le r$, with $v \in X_i$ and $w \in X_i$.
- for all $i, j, k \in I$: if $i \le j \le k$, then $X_i \cap X_k \subseteq X_j$.

	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
N_1	1	1	0	0	0	1	1	0
N_2	0	0	0	1	1	0	0	0
N_3	0	0	1	0	0	0	1	0
N_4	0	1	0	1	0	0	0	1
N_5	0	0	1	0	1	1	0	0



3 tracks

Figure 2.
Example of gate matrix layout

The pathwidth of a path-decomposition $(X_1, X_2, \ldots, X_r)$ is $\max_{1 \le i \le r} |X_i| - 1$. The pathwidth of a graph G is the minimum pathwidth over all possible path-decompositions of G.

In figure 1, an example of a graph with treewidth and pathwidth 2 is given, together with a tree- and path-decomposition of it.

Clearly, the pathwidth of a graph is at least its treewidth. There are several equivalent characterizations of the notions of treewidth and pathwidth, see e.g. [3,15,18,99,143]. The (probably) most well known equivalent characterization of treewidth is by the notion 'partial k-tree', see [132,139]. Also, tree decompositions are reflected by graph expressions, where graphs are built by operations on graphs with some special vertices (the *sources*) like: parallel composition, forget sources, renaming of sources. The treewidth can be characterized in terms of the number of sources used in the operations. See [50].

3 Applications

Several well-studied graph classes have bounded treewidth or pathwidth, hence many results discussed here also apply for these classes. Examples are trees (treewidth 1), series-parallel graphs (treewidth 2), outerplanar graphs (treewidth 2), and Halin graphs (treewidth 3). See e.g. [18,20,132,143]. We mention some other applications.

3.1 VLSI layouts

A well studied problem in VLSI layout theory is the GATE MATRIX LAYOUT problem. This problem is stated in terms of a matrix $M = (m_{ij})$, whose columns represent gates $G_1, \ldots, G_n$, and whose rows represent nets $N_1, \ldots, N_m$. If $m_{ij} = 1$, then net N_i must be connected with gate G_j. An example is given in figure 2. The problem of finding a permutation of the gates, such that all nets can be made within the minimum number of tracks is equivalent to the pathwidth problem (see [63]). See [99] for an extensive overview. See also [53].

3.2 Cholesky factorization

There is also a close connection between treewidth, and Choleski factorization on sparse symmetric matrices.

In the multifrontal method for Choleski factorization, one step is of the form

$$\begin{bmatrix} d & v^T \\ v & B \end{bmatrix} = \begin{bmatrix} \sqrt{d} & 0 \\ v/\sqrt{d} & I \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & B - v \cdot v^T/d \end{bmatrix} \cdot \begin{bmatrix} \sqrt{d} & v^T/\sqrt{d} \\ 0 & I \end{bmatrix}$$

where v is an $(n-1)$-vector, and B is an $n-1$ by $n-1$ maxtrix. I is the $n-1$ by $n-1$ identity matrix. The process is repeated with the matrix $B - v \cdot v^T$. Consider the graph with vertices $1, 2, \ldots, n$, and edges between vertices i and j, if the matrix entries on positions (i,j) and (j,i) are non-zero. One step as described above corresponds to removing a vertex and connecting all its neighbors. As the matrix is sparse, one wants to find an order of colums/rows to be eliminated for which all matrices $v \cdot v^T$ are small, i.e. have a large number of columns and rows that are entirely 0. One can show that to bound the maximum size of these matrices corresponds to bounding the treewidth of the graph, described above. For more details, see e.g. [29].

3.3 Expert systems

Graphs modelling certain type of expert systems have been observed to have small treewidth in practice. Tree-decompositions of small treewidth for these graphs can be used to perform efficiently certian otherwise time-consuming statistical computations needed for reasoning with uncertainity in these systems. See e.g. [92,138].

3.4 Evolution theory

Researchers in molecular biology are interested in the problem, given a set of species, a set of characteristics, and for each specie and each characteristic, the value that that characteristic has for that specie, to find a 'good' evolution tree for these species and their possibly extinct ancestors. One variant of this problem is called the PERFECT PHYLOGENY problem. This problem can be shown to be equivalent with the following graph problem: given a graph $G = (V, E)$ with a coloring of the vertices, can we add edges to G such that the resulting graph is chordal but has no edges between vertices of the same color? Equivalently, does there exist a tree-decomposition $(\{X_i \mid i \in I\}, T)$ of G such that for all $i \in I$: if $v, w \in X_i$, $v \neq w$, then v and w have different colors. So, a necessary condition is that the treewidth of G is smaller than the number of colors. See [2,28,33,79,80,98].

3.5 Natural language processing

Kornai and Tuza [88] have observed that dependency graphs of sentences encoding the major syntactic relations among the words have usually pathwidth at most 6. The pathwidth closely resembles the *narrowness* of these graphs. For the relationship of this notion to natural language processing, see [88].

4 Bounded treewidth and linear time algorithms

An important reason for the interest in tree-decompositions, is that if we have a tree-decomposition of a graph $G = (V, E)$ with its treewidth bounded by some fixed constant k, then we can solve many problems that are hard (intractable) for arbitrary graphs, in polynomial and often linear time. Problems which can be dealt with in this way include many well-known NP-complete problems, like INDEPENDENT SET, HAMILTONIAN CIRCUIT, STEINER TREE, etc., but also certain statistical computations (including some with applications to reasoning with uncertainity in expert systems [92,138]), and some PSPACE-complete problems [4,5,26]. Results of this type can be found — among others — in [3,4,5,8,10,14,19,26,22,31,37,44,47, 52,55,67,69,71,73,74,75,87,90,93,94,95,96,107,132,137,141,142,143,144,145].

As an example we consider the maximum independent set problem. In this problem, we a looking for the maximum size of a set $W \subseteq V$ in a given graph $G = (V, E)$, such that for all $v, w \in W : (v, w) \notin E$.

Given a tree-decomposition, it is easy to make one with the same treewidth, and with T a rooted binary tree. Suppose we have such a tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of input graph G, with root of T r, and with treewidth k. For each $i \in I$, define $Y_i = \{v \in X_j \mid j = i \text{ or } j \text{ is a descendant of } i\}$.

Note that if $v \in Y_i$, and $v \in X_j$ for some node $j \in I$ that is not a descendant of i, then by definition of tree-decomposition, $v \in X_i$. Similarly, if $v \in Y_i$, and v is adjacent to a vertex $w \in X_j$ with j a descendant of i, then $v \in X_i$ or $w \in X_i$. As a consequence, we have that, when we have an independent set W of the subgraph induced by Y_i, $G[Y_i]$, and want to extend this to an independent set of G, then important is *only* what vertices in X_i belong to W, not what vertices in $Y_i - X_i$ belong to W. Of the latter, only the *number* of the vertices in W is important.

For $i \in I$, $Z \subseteq X_i$, define $is_i(Z)$ to be the maximum size of an independent set W in $G[Y_i]$ with $W \cap X_i = Z$. Take $is_i(Z) = -\infty$, if no such set exists.

Our algorithm to solve the independent set problem on G basically consists of computing all tables is_i, for all nodes $i \in I$. This is done in a bottom-up manner in the tree: each table is_i is computed after the tables of the children of node i are computed. For a leaf node i, the following formula can be used to compute all $2^{|X_i|}$ values in the table is_i.

$$is_i(Z) = \begin{cases} |Z| & \text{if } \forall v, w \in Z : (v, w) \notin E \\ -\infty & \text{if } \exists v, w \in Z : (v, w) \in E \end{cases}$$

For an internal node i with two children j and k, we have the following formula.

$$is_i(Z) = \begin{cases} \max\{is_j(Z') + is_k(Z'') + |Z \cap (X_i - X_j - X_k)| \\ -|Z \cap X_j \cap X_k| \mid Z \cap X_j = Z' \cap X_i \\ \text{and } Z \cap X_k = Z'' \cap X_i\} & \text{if } \forall v, w \in Z : (v, w) \notin E \\ -\infty & \text{if } \exists v, w \in Z : (v, w) \in E \end{cases}$$

The idea behind the last formula is: take the maximum over all sets $Z' \subseteq X_j$ that agree with Z in which vertices in $X_i \cap X_j$ belong to the independent set, and similarly for $Z'' \subseteq X_k$. Vertices in $Z \cap X_i - X_j - X_k$ are not counted yet, so their number should be added, while vertices in $Z \cap X_j \cap X_k$ are counted twice, hence their number should be subtracted once.

We compute for each node $i \in I$ the table is_i in some bottom-up order, until we have computed the table is_r. Note that we then can easily find the maximum size of an independent set in G, as this is $\max_{Z \subseteq X_r} is_r(Z)$. Hence, we have an algorithm, that solves the independent set problem on G in $O(2^{3k}n)$ time. (Optimizations can bring the factor 2^{3k} down to 2^k.) It is also possible, by using standard dynamic programming techniques, to construct the maximum sized independent set W itself.

The idea behind this example is: each table entry gives information about an equivalence class of partial solutions. The number of such equivalence classes is bounded by some constant, when the treewidth is bounded by a constant. Tables can be computed using only the tables of the children of the node.

The technique works for many examples. However, there are also results that state that large *classes* of problems can be solved in linear time, when a tree-decomposition with constant bounded treewidth is available. One of the most powerfull results of this type is the result by Courcelle [47,51,46], which has been extended by Arnborg et al [8], by Borie et al [38], and by Courcelle and Mosbah [52], on (Extended) Monadic Second Order formulas. These result basically state that each graph problem that is expressible with a formula using the following language constructions: logical operations ($\wedge$, $\vee$, $\neg$, $\Rightarrow$), quantification over vertices, edges, sets of vertices, sets of edges (e.g. $\exists v \in V$, $\forall e \in E$, $\forall W \subseteq V$, $\exists F \subseteq E$), membership tests ($v \in W$, $e \in E$), adjacency tests $(v, w) \in E$, v is endpoint of e), and certain extensions, can be solved in linear time on graphs with given a tree-decomposition of constant bounded treewidth. The extensions allow not only to deal with decision problems, but also optimization problems (like maximum independent set).

For example, the problem whether a given graph G can be colored with three colors can be stated as

$$\exists W_1 \subseteq V : \exists W_2 \subseteq V : \exists W_3 \subseteq V : \forall v \in V : (v \in W_1 \vee v \in W_2 \vee v \in W_3) \wedge \forall v \in V : \forall w \in W : (v, w) \in E \Rightarrow (\neg(v \in W_1 \wedge w \in W_1) \wedge \neg(v \in W_2 \wedge w \in W_2) \wedge \neg(v \in W_3 \wedge w \in W_3))$$

In many cases, the information, computed per node $i \in I$ is an element of a finite set. Then, the algorithm can be seen as a finite state tree-automata, and optimalization techniques can be applied, similar to Myhill-Nerode theory [14,62]. (See also [48,45,49].)

In [64,65] parametric problems on graphs with bounded treewidth are solved, using modifications of the technique, presented above.

For some problems (e.g. the maximum independent set problem) polynomial time algorithms are still known to exist, if the input graph is given together with a tree-decomposition of treewidth $O(\log n)$. (See e.g. [19].) For other problems, it is unknown whether such algorithms exist.

The problem whether two given graphs are isomorphic is also solvable in polynomial time, when the graphs have bounded treewidth [11,22,42]. The techniques are here somewhat different.

There also exist problems that remain hard when restricted to graphs with constant bounded treewidth, for instance the bandwidth problem is NP-complete for a very restricted subclass of the trees [100]. For some problems the complexity when we restrict the instances to graphs with bounded treewidth is open, like the problem to determine the pathwidth of graphs with treewidth ≤ 2 [30].

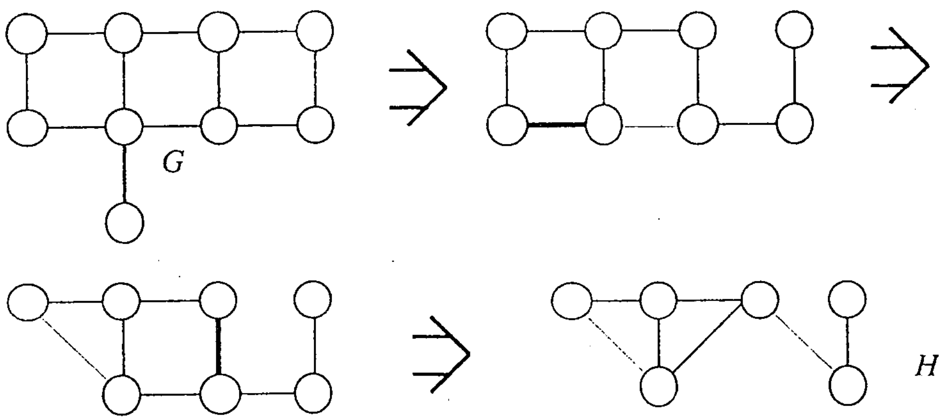


Figure 3.
G is a minor of H

5 Graph minors

In this section, we give a short overview of some recent results on graph minors. A graph $H = (W, F)$ is a minor of a graph $G = (V, E)$, if (a graph isomorphic to) H can be obtained from G by a series of zero or more vertex deletions, edge deletions, and/or edge contractions (in arbitrary order), where an edge contraction is the operation to replace two adjacent vertices v and w by a vertex that is adjacent to all vertices that were adjacent to v or w. For an example, see figure 3.

Robertson and Seymour obtained the following deep results on graph minors [17,109,115,111,122,122,116,117,121,124,123,125,114,118,119,120,126,127,128, 129,110,112,113].

Theorem 5.1
For every class of graphs $\mathcal{G}$, that is closed under taking of minors, there exists a finite set of graphs, $ob(\mathcal{G})$, called the obstruction set of $\mathcal{G}$, such that for each graph G: $G \in \mathcal{G}$, if and only if there is no $H \in ob(\mathcal{G})$ that is a minor of G.

For example, the obstruction set of the planar graphs is $\{K_5, K_{3,3}\}$ [140]. Theorem 5.1 was formerly known as Wagners conjecture.

Theorem 5.2
For every graph H, there exists an $O(n^3)$ algorithm, that, given a graph G, tests whether H is a minor of G.

Theorem 5.3
For every planar graph H, there exists a constant c_H, such that for every graph G: if H is not a minor of G, then the treewidth of G is at most c_H.

The constant factor of the algorithm in theorem 5.2 is very high, making this algorithm not suitable for practical use. In [129], it is shown that one can take in 5.3 $c_H = 20^{4|V_H|+8|E_H|^5}$. From theorem 5.1 and theorem 5.2 it follows that every class of graphs, closed under minor taking, is recognizable in $O(n^3)$ time (do a minor test for each graph in the obstruction set.) Using theorem 5.1, theorem 5.3, the result of the next section, that states that for graphs with constant bounded treewidth, a tree-decomposition of constant bounded treewidth can be found in $O(n)$ time, and the fact, that with such a tree-decomposition, minor tests can be done in linear time with a procedure of the type, discussed in section 4, the following result can be derived: every class of graphs that does not contain all planar graphs and that is closed under minor taking, can be recognized in $O(n)$ time. (See also [13].)

Many applications of this theory were found by Fellows and Langston [58,60,61]. Note however that the constants hidden in the 'O'-notation may be quite large, and that the proof of theorem 5.1 is inherently non-constructive (in a deep mathematical sense) [66]. I.e., it is not possible in all cases to extract the obstruction set of a class of graphs $\mathcal{G}$, given a formal proof that $\mathcal{G}$ is minor closed. Thus, we may arrive in a situation where we know that a polynomial algorithm exists for the problem without knowing the algorithm itself. Also, the algorithms are recognition algorithms: they do not constuct anything (like a vertex ordering, tree-decomposition, etc.)

A technique that allows us in some cases to overcome non-constructive aspects of this theory is self-reduction, advocated by Fellows and Langston, see e.g. [21,39, 59,63].

Self reduction is the technique to consult a decision algorithm a number of times with different inputs in order to construct a solution for the original problem. As an example, consider the problem of finding a simple path of length at least k (k constant) in an undirected graph. (There are direct and more efficient algorithms for this problem [27,63]; the solution here is presented only to explain the technique.) The class of graphs that do not contain such a path is closed under minor taking, and does not contain all planar graphs, so we have a linear time algorithm, deciding whether a given graph contains a simple path of length at least k. Given a graph G, we can solve the problem in $O(n \cdot e)$ time by first testing whether G contains a desired path, and then repeatedly trying to remove an edge from G, such that the resulting graph still contains a simple path of length k. When no edge can be deleted anymore, the resulting graph is precisely the desired path.

Even when we do not know the obstruction set, in several cases it is still possible to *construct* polynomial time algorithms based on minor tests (see [63]).

In some cases, obstruction sets, and hence the decision algorithms themselves are computable [12,16,40,57,62,78,81,91,103,131,136]. The size of the obstruction sets can grow very fast: for instance, the obstruction set of the graphs with pathwidth at most k contains at least $k!^2$ trees, each containing $\frac{5 \cdot 3^k - 1}{2}$ vertices [136]. This clearly limits the practicality of the approach described above.

Also, in some cases, linear time minor tests are possible [27,25,54,63]. For instance, suppose that H is a cycle of length k. The algorithm is as follows: first make a depth-first search spanning tree $T = (V, F)$ of the input graph $G = (V, E)$. If there is a backedge between a vertex v and a predecessor w of v which is at least $k - 1$ levels above v in T, then G contains H as a minor, stop. Otherwise, construct $(\{X_v \mid v \in V\}, T = (V, F))$, with $X_v = \{v\} \cup \{w \mid w$ is a predecessor of v and differs at most $k - 2$ levels from v in $T\}$. This is a tree-decomposition of G with treewidth at most $k - 2$. Use this tree-decomposition to solve the problem in linear time. (See [63].)

6 Finding tree-decompositions

In this section we consider the problem of finding tree-decompositions, and determining the treewidth of a graph. Unfortunately, determining whether the treewidth of a given graph $G = (V, E)$ is at most a given integer k is NP-complete [6]. The latter result holds also for pathwidth [6]. The complexity of these problem has been studied for several classes of graphs. Table 1 mentions several of the known results of this type.

Polynomial time approximation algorithms with $O(\log n)$ performance ratio for treewidth, and $O(\log^2 n)$ performance ratio for pathwidth, are presented in [29]. For several classes of perfect graphs, polynomial time approximation algorithms can be found in [84]. Seymour and Thomas gave a polynomial time algorithm for the branchwidth of planar graphs [134]; this directly implies a polynomial time approximation algorithm for the treewidth of planar graphs with a performance ratio $1\frac{1}{2}$ [114].

Class	Treewidth	Pathwidth
Bounded degree	N [35]	N [101] (3)
Trees/Forests	C	P [133]
Series-parallel graphs	C	P [32]
Outerplanar graphs	C	P [32]
Halin graphs	C [143]	P [32]
k-Outerplanar graphs	C [20]	P [32]
Planar graphs	O	N [101] (3)
Chordal graphs	P (1)	N [68]
Starlike chordal graphs	P (1)	N [68]
k-Starlike chordal graphs	P (1)	P [68]
Co-chordal graphs	P [85]	P [85]
Split graphs	P (1)	P [68,84]
Bipartite graphs	N	N
Permutation graphs	P [34]	P [34]
Circular permutation graphs	P [34]	O
Cocomparability graphs	N [6,72]	N [6,72]
Cographs	P [36]	P [36]
Chordal bipartite graphs	P [86]	N [35]
Interval graphs	P (2)	P (2)
Circular arc graphs	P [135]	O
Circle graphs	P [83]	N [35]

P = polynomial time solvable. C = constant, hence linear time solvable. N = NP-complete. O = Open problem. (1) The treewidth of a chordal graph equals its maximum clique size minus one. (2) The treewidth and pathwidth of an interval graphs equal its maximum clique size minus one. (3) NP-completeness is shown for vertex separation number, but this is equivalent to pathwidth.

Table 1:
Complexity of Pathwidth and Treewidth on different classes of graphs

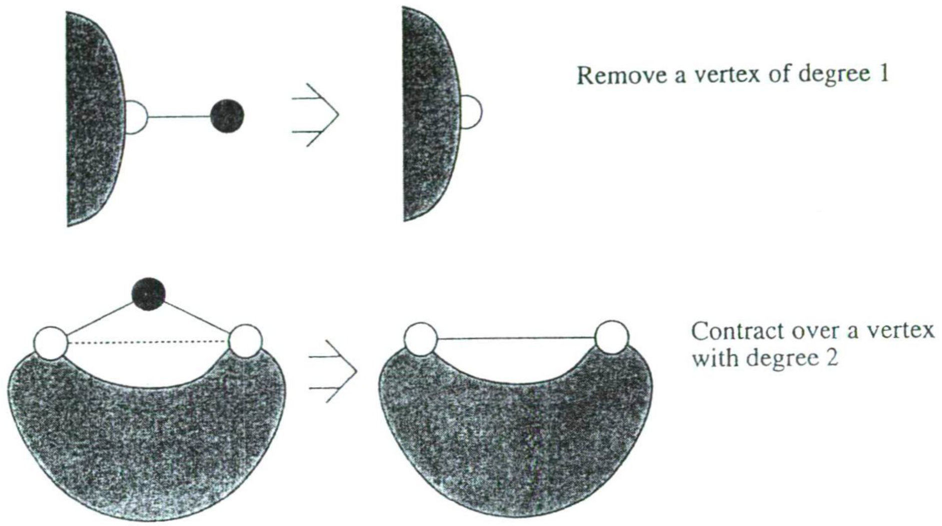


Figure 4.
Rewriting a graph with treewidth ≤ 2

For constant k, polynomial time algorithms exist for the problems. The graphs with treewidth 1 are exactly the forests. Algorithms that recognize graphs with treewidth 2 and 3 in linear time, and find the corresponding tree-decompositions were described by Matousek and Thomas [97], using results from [9]. A similar algorithm (with a quite involved case analysis) for treewidth 4 was found recently by Sanders [130]. For example, the connected graphs with treewidth 2 are exactly those graphs that can be rewritten to a single vertex, using the operations shown in figure 4. For larger k, also recognition algorithms based on rewriting exist [7]. (In [7], a much larger class of problems is also shown to be solvable with these rewrite techniques.) The latter algorithms can at present, not produce a corresponding tree-decomposition of the input graph.

For arbitrary fixed k, an $O(n \log n)$ algorithm can be found, using the following result, due to Reed [108].

Theorem 6.1
For every constant k, there exists an $O(n \log n)$ algorithm, that given a graph $G = (V, E)$, either outputs that the treewidth of G is larger than k, or outputs a tree-decomposition of G with treewidth at most $3k + 2$.

Actually, the result proven by Reed has a number, larger than $3k + 2$. Minor improvements give the result stated above. The running time of this algorithm is singly exponential in k. Similar, but slower algorithms have been found by

Robertson and Seymour [119] and by Lagergren [89], the latter result also has an efficient parallel variant.

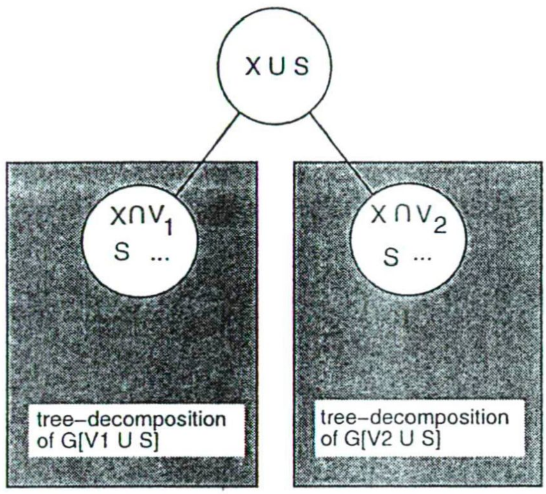


Figure 5.
Illustration to approximation algorithm

These algorithms and the approximation algorithm in [29] are based on repeatedly finding separators. An 1/3-2/3 separator of a set $W \subseteq V$ in a graph $G = (V, E)$ is a set $S \subseteq V$, such that $V - S$ can be partitioned into two non-adjacent sets of vertices V_1, V_2, such that both V_1 and V_2 contain at most $2|W|/3$ vertices in W.

Each of the algorithms can be described by a recursive procedure which is called with two arguments: a graph $G' = (V', E')$ (an induced subgraph of G), and a set of vertices $X \subseteq V'$. The algorithm produces a tree-decomposition with the root node set X_r of T containing all vertices in X ($X \subseteq X_r$). It works basically as follows: When V' is 'small enough', yield a one-node tree-decomposition, the node containing all vertices in V'. Otherwise, first find a 'small' 1/3-2/3 separator S of X in G', separating $V' - S$ into V_1 and V_2. Call the procedure recursively for graph $G[V_1 \cup S]$ and set $S \cup (X \cap V_1)$, and for graph $G[V_2 \cup S]$ and set $S \cup (X \cap V_2)$. The desired tree-decomposition is obtained by taking one new node containing $X \cap S$, and connecting this node to the root nodes of the two tree-decompositions yielded by the recursive calls of the procedure (see figure 5). If the treewidth of G is at most k, then a 1/3-2/3 separator, as needed for the algorithm, exists of size at most k, and can be found, in time, linear in V', using flow techniques [119]. Starting with an arbitrary set X of size at most $3k$, it follows with induction, that each call of the procedure uses sets X of size at most $3k$, assuming the treewidth of G is at most k. ($|X \cap V_i \cup S| \leq 2|X|/3 + |S| \leq 2k + k$.) Hence, the algorithm produces in this case a tree-decomposition of treewidth less than $4k$.

Reed [108] has shown that one can also find small sized separator sets S, that do not only separate X, but also partition V' into sets of size at most 3/4 of $|V'|$.

This gives a recursion depth of $O(\log n)$, and results in an $O(n \log n)$ algorithm. (The expose above is only a very rough sketch of some of the most important ideas of the algorithms. See further [29,89,108,119].)

Using the algorithm of theorem 5.1, and a constant number of minor tests, it follows that the 'treewidth $\leq k$' and 'pathwidth $\leq k$' problems (for constant k) are decidable in $O(n \log n)$ time. (Use that the treewidth and pathwidth can not increase by taking minors.) However, it is also possible to obtain direct, explicit and constructive algorithms for the problems.

Both Lagergren and Arnborg [91] and Bodlaender and Kloks [31,82] give such an algorithm, using an involved application of the technique, discussed in section 4. Independently, results of a similar nature were obtained by Abrahamson and Fellows [1]. From these results it follows that a technique of Fellows and Langston [62] can be used to compute the corresponding obstruction set. Bodlaender and Kloks [31] also discuss how in the same time bounds the path- or tree-decompositions with pathwidth or treewidth at most k can be found, if existing.

Recently, the author has found a linear time algorithm for the problems to decide whether a graph has pathwidth or treewidth at most some constant k, and if so, to find a path- or tree-decomposition with pathwidth or treewidth at most k [24]. This algorithm uses a recursion technique, and the result in [31] as essential ingredients.

A study to dynamic algorithms for graphs with small treewidth has been made by Cohen et al. [43] and recently by the author [23].

Acknowledgements

I thank Bruno Courcelle, Jens Gustedt, Ton Kloks, Mike Fellows, Detlef Seese, and Andrzej Proskurowski for useful comments on earlier versions of this tourist guide.

References

[1] K. R. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph Structure Theory, Contemporary Mathematics vol. 147*, pages 539–564. American Mathematical Society, 1993.

[2] R. Agarwala and D. Fernandez-Baca. A polynomical-time algorithm for the phylogeny problem when the number of character states is fixed. Manuscript, 1992.

[3] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985.

[4] S. Arnborg. Graph decompositions and tree automata in reasoning with uncertainty. Manuscript, to appear in Journal of Experimental and Theoretical AI, 1991.

[5] S. Arnborg. Some PSPACE-complete logic decision problems that become linear time solvable on formula graphs of bounded treewidth. Manuscript, 1991.

[6] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[7] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. In H. Ehrig, H. Kreowski, and G. Rozenberg, editors, *Proceedings of the Fourth Workshop on Graph Grammars and Their Applications to Computer Science*, pages 70–83. Springer Verlag, Lecture Notes in Computer Science, vol. 532, 1991. To appear in J. ACM.

[8] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.

[9] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.

[10] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.

[11] S. Arnborg and A. Proskurowski. Canonical representations of partial 2- and 3-trees. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory*, pages 310–319. Springer Verlag, Lecture Notes in Computer Science, vol. 477, 1990.

[12] S. Arnborg, A. Proskurowski, and D. G. Corneil. Forbidden minors characterization of partial 3-trees. *Disc. Math.*, 80:1–19, 1990.

[13] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. In E. Börger, H. Kleine Büning, M. M. Richter, and W. Schönfeld, editors, *Proceedings 4th Workshop on Computer Science Logic, CSL'90*, pages 1–16. Springer Verlag, Lecture Notes in Computer Science, vol. 533, 1991.

[14] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[15] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

[16] D. Bienstock and N. Dean. On obstructions to small face covers in planar graphs. *J. Comb. Theory Series B*, 55:163–189, 1992.

[17] D. Bienstock, N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a forest. *J. Comb. Theory Series B*, 52:274–283, 1991.

[18] H. L. Bodlaender. Classes of graphs with bounded treewidth. Technical Report RUU-CS-86-22, Dept. of Computer Science, Utrecht University, Utrecht, the Netherlands, 1986.

[19] H. L. Bodlaender. Dynamic programming algorithms on graphs with bounded tree-width. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, pages 105–119. Springer Verlag, Lecture Notes in Computer Science, vol. 317, 1988.

[20] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 36:116–126, 1988.

[21] H. L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. In *Proc. 15th Int. Workshop on Graph-theoretic Concepts in Computer Science WG'89*, pages 232–244. Springer Verlag, Lect. Notes in Computer Science, vol. 411, 1990. To appear in: Annals of Discrete Mathematics.

[22] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11:631–643, 1990.

[23] H. L. Bodlaender. Dynamic algorithms for graphs with treewidth 2. Manuscript, 1992.

[24] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. Technical Report RUU-CS-92-27, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1992. To appear in proceedings STOC'93.

[25] H. L. Bodlaender. On disjoint cycles. In *Proceedings 17th International Workshop on Graph-Theoretic Concepts in Computer Science WG'91*, pages 230–239. Springer Verlag, Lecture Notes in Computer Science, vol. 570, 1992.

[26] H. L. Bodlaender. Complexity of path-forming games. *Theor. Comp. Sc.*, 110:215–245, 1993.

[27] H. L. Bodlaender. On linear time minor tests with depth first search. *J. Algorithms*, 14:1–23, 1993.

[28] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow. Two strikes against perfect phylogeny. In *Proceedings 19th International Colloquium on Automata, Languages and Programming*, pages 273–283, Berlin, 1992. Springer Verlag, Lecture Notes in Computer Science 623.

[29] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. In G. Schmidt and R. Berghammer, editors, *Proceedings 17th International Workshop on Graph-Theoretic Concepts in Computer Science WG'91*, pages 1–12. Springer Verlag, Lecture Notes in Computer Science, vol. 570, 1992.

[30] H. L. Bodlaender and J. Gustedt. A conjecture on the pathwidth of k-trees. In: Proceedings AMS Summer Conference on Graph Minors, 1992. Contemp. Math. 147. In section "Open Problems", editor N. Dean, 1993.

[31] H. L. Bodlaender and T. Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 544–555. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

[32] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. Manuscript. A preliminary version appeared as [31], 1993.

[33] H. L. Bodlaender and T. Kloks. A simple linear time algorithm for triangulating three-colored graphs. *J. Algorithms*, 15:160–172, 1993.

[34] H. L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. In *Proceedings 20th International Colloquium on Automata, Languages and Programming*, pages 114–125, Berlin, 1993. Springer Verlag, Lecture Notes in Computer Science, vol. 700.

[35] H. L. Bodlaender, T. Kloks, D. Kratsch, and H. Müller, 1993. Unpublished results.

[36] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Meth.*, 6:181–188, 1993.

[37] R. B. Borie. *Recursively Constructed Graph Families*. PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1988.

[38] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–582, 1992.

[39] D. J. Brown, M. R. Fellows, and M. A. Langston. Nonconstructive polynomial-time decidability and self-reducibility. *Int. J. Computer Math.*, 31:1–9, 1989.

[40] R. L. Bryant, M. R. Fellows, N. G. Kinnersley, and M. A. Langston. On finding obstruction sets and polynomial-time algorithms for gate matrix layout. In *Proc. 25th Allerton Conf. on Communication, Control and Computing*, 1987.

[41] N. Chandrasekharan. *Fast Parallel Algorithms and Enumeration Techniques for Partial k-Trees*. PhD thesis, Clemson University, 1990.

[42] N. Chandrasekharan. Isomorphism testing of k-trees is in NC, for fixed k. *Inform. Proc. Letters*, 34:283–287, 1990.

[43] R. F. Cohen, S. Sairam, R. Tamassia, and J. S. Vitter. Dynamic algorithms for bounded tree-width graphs. Technical Report CS-92-19, Department of Computer Science, Brown University, 1992.

[44] D. G. Corneil and J. M. Keil. A dynamic programming approach to the dominating set problem on k-trees. *SIAM J. Alg. Disc. Meth.*, 8:535–543, 1987.

[45] B. Courcelle. The monadic second-order logic of graphs VI: On several representations of graphs by relational structures. Technical Report 89-99, Bordeaux-I University, 1989. To appear in: Discrete Applied Mathematics.

[46] B. Courcelle. Graph rewriting: an algebraic and logical approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, pages 192–242, Amsterdam, 1990. North Holland Publ. Comp.

[47] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[48] B. Courcelle. The monadic second-order logic of graphs V: On closing the gap between definability and recognizability. *Theor. Comp. Sc.*, 80:153–202, 1991.

[49] B. Courcelle. The monadic second-order logic of graphs VII: Graphs as relational structures. Manuscript, to appear in: Theoretical Computer Science, 1991.

[50] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. *Bulletin of the EATCS*, 46:193–226, 1992. To appear in: Proceedings AMS Summer Research Conference on Graph Minors.

[51] B. Courcelle. The monadic second-order logic of graphs III: Treewidth, forbidden minors and complexity issues. *Informatique Théorique*, 26:257–286, 1992.

[52] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comp. Sc.*, 109:49–82, 1993.

[53] N. Deo, M. S. Krishnamoorty, and M. A. Langston. Exact and approximate solutions for the gate matrix layout problem. *IEEE Trans. Computer Aided Design*, 6:79–84, 1987.

[54] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. Manuscript, 1991.

[55] E. S. El-Mallah and C. J. Colbourn. Partial k-tree algorithms. *Congressus Numerantium*, 64:105–119, 1988.

[56] M. R. Fellows. The Robertson-Seymour theorems: A survey of applications. *Contemporary Mathematics*, 89:1–18, 1989.

[57] M. R. Fellows, N. G. Kinnersley, and M. A. Langston. Finite-basis theorems, and a computational integrated approach to obstruction set isolation. In E. Kaltofen and S. M. Watt, editors, *Proceedings of the 3rd Conference on Computers and Mathematics*, pages 37–45, New York, 1989. Springer Verlag.

[58] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Inform. Proc. Letters*, 26:157–162, 1987.

[59] M. R. Fellows and M. A. Langston. Fast self-reduction algorithms for combinatorial problems of VLSI design. In *Proc. 3rd Aegean Workshop on Computing*, pages 278–287. Springer Verlag, Lecture Notes in Computer Science, vol. 319, 1988.

[60] M. R. Fellows and M. A. Langston. Layout permutation problems and well-partially-ordered sets. In J. Reif, editor, *5th MIT Conf. on Advanced Research in VLSI*, pages 315–327, Cambridge, MA, 1988. Springer Verlag Lecture Notes in Computer Science 319.

[61] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *J. ACM*, 35:727–739, 1988.

[62] M. R. Fellows and M. A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 520–525, 1989.

[63] M. R. Fellows and M. A. Langston. On search, decision and the efficiency of polynomial-time algorithms. In *Proceedings of the 21rd Annual Symposium on Theory of Computing*, pages 501–512, 1989.

[64] D. Fernández-Baca and G. Slutzki. Solving parametric problems on trees. *J. Algorithms*, 10:381–402, 1989.

[65] D. Fernández-Baca and G. Slutzki. Parametic problems on graphs of bounded treewidth. In O. Nurmi and E. Ukkonen, editors, *Proceedings 3rd Scandinavian Workshop on Algorithm Theory*, pages 304–316. Springer Verlag, Lecture Notes in Computer Science, vol. 621, 1992.

[66] H. Friedman, N. Robertson, and P. D. Seymour. The metamathematics of the graph minor theorem. *Contemporary Mathematics*, 65:229–261, 1987.

[67] D. Granot and D. Skorin-Kapov. On some optimization problems on k-trees and partial k-trees. Manuscript, to appear in Discrete Appl. Math., 1988.

[68] J. Gustedt. Path width for chordal graphs is NP-complete. Technical Report 221/1989, Technical University Berlin, 1989. To appear in Discr. Appl. Math.

[69] A. Habel. Graph-theoretic properties compatible with graph derivations. In J. van Leeuwen, editor, *Proceedings 14th International Workshop on Graph-Theoretic Concepts in Computer Science WG'88*, pages 11–29. Springer Verlag, Lecture Notes in Computer Science, vol. 344, 1988.

[70] A. Habel and H. J. Kreowski. May we introduce to you: hyperedge replacement. In H. Ehrig, M. Nagl, and A. Rosenberg, editors, *Proc. Graph-Grammars and their Applications to Computer Science '86*, pages 15–26. Springer Verlag, Lect. Notes in Comp. Science vol. 291, 1987.

[71] A. Habel and H.-J. Kreowski. Filtering hyperedge-replacement languages through compatible properties. In *Proceedings 15th International Workshop on Graph-Theoretic Concepts in Computer Science WG'89*, 1990.

[72] M. Habib and R. H. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. Technical Report 336/1992, Fachbereich Mathematik, Technische Universität Berlin, 1992.

[73] E. Hare, S. Hedetniemi, R. Laskar, K. Peters, and T. Wimer. Linear-time comptability of combinatorial problems on generalized-series-parallel graphs. In D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, editors, *Proc. of the Japan-US Joint Seminar on Discrete Algorithms and Complexity*, Orlando, Florida, 1987. Academic Press, Inc.

[74] W. Hohberg and R. Reischuk. A framework to design algorithms for optimization problems on graphs. Preprint, April 1990.

[75] K. Jansen and P. Scheffler. Generalized coloring for tree-like graphs. In *Proceedings 18th International Workshop on Graph-Theoretic Concepts in Computer Science WG'92*, pages 50–59, Berlin, 1993. Springer Verlag, Lecture Notes in Computer Science, vol. 657.

[76] D. S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 6:434–451, 1985.

[77] D. S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 8:285–303, 1987.

[78] Y. Kajitani, A. Ishizuka, and S. Ueno. Characterization of partial 3 trees in terms of 3 structures. *Graphs and Combinatorics*, 2:233–246, 1986.

[79] S. Kannan and T. Warnow. Inferring evolutionary history from DNA sequences. In *Proceedings of the 31rd Annual Symposium on Foundations of Computer Science*, pages 362–371, 1990.

[80] S. Kannan and T. Warnow. Triangulating 3-colored graphs. *SIAM J. Disc. Meth.*, 5:249–258, 1992.

[81] N. G. Kinnersley. *Obstruction Set Isolation for Layout Permutation Problems.* PhD thesis, Washington State University, May 1989.

[82] T. Kloks. *Treewidth.* PhD thesis, Utrecht University, Utrecht, the Netherlands, 1993.

[83] T. Kloks. Treewidth of circle graphs. Technical Report RUU-CS-93-12, Department of Computer Science, Utrecht University, Utrecht, 1993.

[84] T. Kloks and H. Bodlaender. Approximating treewidth and pathwidth of some classes of perfect graphs. In *Proceedings Third International Symposium on Algorithms and Computation, ISAAC'92*, pages 116–125, Berlin, 1992. Springer Verlag, Lecture Notes in Computer Science, vol. 650.

[85] T. Kloks, H. Bodlaender, H. Müller, and D. Kratsch. Computing treewidth and minimum fill-in: All you need are the minimal separators. To appear in: proceedings 1st European Symposium on Algorithms, ESA'93, 1993.

[86] T. Kloks and D. Kratsch. Treewidth of chordal bipartite graphs. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *Proceedings Symp. Theoretical Aspects of Computer Science, STACS'93*, pages 80–89, Berlin, 1993. Springer Verlag, Lecture Notes in Computer Science, vol. 665.

[87] E. Korach and N. Solel. Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth. Manuscript, 1990.

[88] A. Kornai and Z. Tuza. Narrowness, pathwidth, and their application in natural language processing. Manuscript. Submitted Disc. Appl. Math., 1990.

[89] J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of the 31rd Annual Symposium on Foundations of Computer Science*, pages 173–182, 1990.

[90] J. Lagergren. *Algorithms and Minimal Forbidden Minors for Tree-decomposable Graphs.* PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1991.

[91] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 533–543. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

[92] S. J. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *The Journal of the Royal Statistical Society. Series B (Methodological)*, 50:157–224, 1988.

[93] C. Lautemann. Efficient algorithms on context-free graph languages. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, pages 362–378. Springer Verlag, Lect. Notes in Comp. Sc. 317, 1988.

[94] S. Mahajan and J. G. Peters. Regularity and locality in k-terminal graphs. Manuscript, 1990.

[95] E. Mata-Montero. Resilience of partial k-tree networks with edge and node failures. *Networks*, 21:321–344, 1991.

[96] J. Matousěk and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. Manuscript, to appear in: Topological, Algebraical, and Combinatorial Structures, J. Nesetril, ed., North-Holland, 1988.

[97] J. Matousěk and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

[98] F. R. McMorris, T. Warnow, and T. Wimer. Triangulating colored graphs. In proceedings SODA'92, to appear in SIAM J. Disc. Math., 1991.

[99] R. H. Möhring. Graph problems related to gate matrix layout and PLA folding. In E. Mayr, H. Noltemeier, and M. Sysło, editors, *Computational Graph Theory, Comuting Suppl. 7*, pages 17–51. Springer Verlag, 1990.

[100] B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM J. Alg. Disc. Meth.*, 7:505–512, 1986.

[101] B. Monien and I. H. Sudborough. Min cut is NP-complete for edge weighted trees. *Theor. Comp. Sc.*, 58:209–229, 1988.

[102] M. H. Mosbah. *Constructions d'Algorithmes Pour les Graphes Structurés par des Méthodes Algébriques et Logiques.* PhD thesis, Université Bordeaux-I, 1992.

[103] R. Motwani, A. Raghunathan, and H. Saran. Constructive results from graph minors: Linkless embeddings. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 398–407, 1988.

[104] A. Proskurowski. Separating subgraphs in k-trees: Cables and caterpillars. *Disc. Math.*, 49:275–285, 1984.

[105] A. Proskurowski. Maximal graphs of pathwidth k or searching a partial k-caterpillar. Technical Report CIS-TR-89-17, Dept. of Computer and Information Science, University of Oregon, 1989.

[106] A. Proskurowski and M. M. Sysło. Efficient computations in tree-like graphs. Technical Report 235, Mathematik, Techn. Univ. Berlin, 1989.

[107] V. Radhakrishnan, H. B. Hunt III, and R. E. Stearns. Efficient algorithms for solving systems of linear equations and path problems. Technical Report 91-21, Dept. of Computer Science, SUNY Albany, 1991.

[108] B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, 1992.

[109] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *J. Comb. Theory Series B*, 35:39–61, 1983.

[110] N. Robertson and P. D. Seymour. Generalizing Kuratowskis theorem. *Congressus Numerantium*, 45:129–138, 1984.

[111] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory Series B*, 36:49–64, 1984.

[112] N. Robertson and P. D. Seymour. Graph width and well-quasi ordering: a survey. In J. A. Bondy and U. S. R. Murty, editors, *Progress in Graph Theory*, pages 399–406, Toronto, 1984. Academic Press.

[113] N. Robertson and P. D. Seymour. Graph minors — a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge Univ. Press, 1985.

[114] N. Robertson and P. D. Seymour. Graph minors. XI. Distance on a surface. Manuscript, 1985.

[115] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[116] N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory Series B*, 41:92–114, 1986.

[117] N. Robertson and P. D. Seymour. Graph minors. VI. Disjoint paths across a disc. *J. Comb. Theory Series B*, 41:115–138, 1986.

[118] N. Robertson and P. D. Seymour. Graph minors. XII. Excluding a non-planar graph. Manuscript, 1986.

[119] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. Manuscript, 1986.

[120] N. Robertson and P. D. Seymour. Graph minors. XIV. Taming a vortex. Manuscript, 1987.

[121] N. Robertson and P. D. Seymour. Graph minors. VII. Disjoint paths on a surface. *J. Comb. Theory Series B*, 45:212–254, 1988.

[122] N. Robertson and P. D. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory Series B*, 48:227–254, 1990.

[123] N. Robertson and P. D. Seymour. Graph minors. IX. Disjoint crossed paths. *J. Comb. Theory Series B*, 49:40–77, 1990.

[124] N. Robertson and P. D. Seymour. Graph minors. VIII. A Kuratowski theorem for general surfaces. *J. Comb. Theory Series B*, 48:255–288, 1990.

[125] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B*, 52:153–190, 1991.

[126] N. Robertson and P. D. Seymour. Graph minors. XV. Etending an embedding. Manuscript, 1991.

[127] N. Robertson and P. D. Seymour. Graph minors. XVI. Giant steps. Manuscript, 1991.

[128] N. Robertson and P. D. Seymour. Graph minors. XVII. Excluding a non-planar graph. Manuscript, 1991.

[129] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. Technical Report TR89-16, DIMACS, 1989.

[130] D. P. Sanders. On linear recognition of tree-width at most four. Manuscript, 1992.

[131] A. Satyanarayana and L. Tung. A characterization of partial 3-trees. *Networks*, 20:299–322, 1990.

[132] P. Scheffler. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme.* PhD thesis, Akademie der Wissenschaften der DDR, Berlin, 1989.

[133] P. Scheffler. A linear algorithm for the pathwidth of trees. In R. Bodendiek and R. Henn, editors, *Topics in combinatorics and graph theory*, pages 613–620, Heidelberg, 1990. Physica-Verlag.

[134] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. Manuscript, 1990.

[135] R. Sundaram, K. Sher Singh, and C. Pandu Rangan. Treewidth of circular-arc graphs. Manuscript, to appear in SIAM J. Disc. Math., 1991.

[136] A. Takahashi, S. Ueno, and Y. Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. In *SIGAL 91-19-3, IPSJ*, 1991. To appear in: *Annals of discrete mathematics* (Proceedings of 2nd Japan conference on graph theory and combinatorics, 1990).

[137] J. Telle and A. Proskurowski. Efficient sets in partial k-trees. Technical report, Department of Computer and Information Science, University of Oregon, 1991.

[138] L. C. van der Gaag. *Probability-Based Models for Plausible Reasoning.* PhD thesis, University of Amsterdam, 1990.

[139] J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory*, pages 527–631, Amsterdam, 1990. North Holland Publ. Comp.

[140] K. Wagner. Über eine Eigenshaft der ebenen Complexe. *Math. Ann.*, 14:570–590, 1937.

[141] M. Wiegers. The k-section of treewidth restricted graphs. In B. Rovan, editor, *Proceedings Conference on Mathematical Foundations of Computer Science MFCS'90*, pages 530–537, Berlin, 1990. Springer Verlag, Lecture Notes in Computer Science, vol. 452.

[142] T. V. Wimer. Linear algorithms for the dominating cycle problems in series-parallel graphs, 2-trees and Halin graphs. *Congressus Numerantium*, 56, 1987.

[143] T. V. Wimer. *Linear Algorithms on k-Terminal Graphs.* PhD thesis, Dept. of Computer Science, Clemson University, 1987.

[144] T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear graph algorithms. *Congressus Numerantium*, 50:43–60, 1985.

[145] X. Zhou, S. Nakano, H. Suzuki, and T. Nishizeki. An efficient algorithm for edge-coloring series-parallel multigraphs. In I. Simon, editor, *Proceedings LATIN'92*, pages 516–529. Springer Verlag, Lecture Notes in Computer Science, vol. 583, 1992.

Received March 22, 1993

Acta Cybernetica, Vol. 11, No. 1-2, Szeged, 1993

A Lower Bound for On-Line Vector-Packing Algorithms*

G. Galambos[†] H. Kellerer[‡] G. Wöginger[§]

Abstract

In this paper we deal with the vector-packing problem which is a generalization of the well known one-dimensional bin-packing problem to higher dimensions. We give the first, non-trivial lower bounds on the asymptotic worst case ratio of any on-line d-dimensional vector packing algorithm.

Keywords. vector-packing, worst-case analysis, on-line algorithms, lower bounds, competitive algorithms.

1 Introduction

We consider the following problem, called *vector-packing*: Given a list $L_n = \langle a_1, \ldots a_n \rangle$ of n elements where each element is a d dimensional *vector* $(d \geq 1)$. The i-th vector in the liste is denoted by $v(a_i) = (v_1(a_i), \ldots, v_d(a_i))$, where $0 \leq v_j(a_i) \leq 1$ for $j = 1, 2, \ldots, d$. The goal is to pack all elements into the minimal number of bins in such a way that for any non-empty B bin of the packing and for any index $1 \leq j \leq d$

$$\sum_{a_i \in B} v_j(a_i) \leq 1.$$

For $d = 1$, this problem is the famous "classical" bin-packing problem, which is known to be NP-hard. Hence, we are mainly interested in 'good' approximation algorithms.

The quality of an approximation algorithm is usually measured by its *asymptotic worst-case ratio* that is defined as follows. For an arbitrary vector-packing algorithm A and an arbitrary list of d-dimensional vectors L, we denote by L^* the minimal number of bins needed to pack the list L and by $A(L)$ the number of bins which algorithm A uses to pack the elements of L. Let $R_A(k)$ denote the supremum of the ratios $A(L)/L^*$ over all lists L with $L^* = k$. The asymptotic worst case ratio R_A is defined by the equation

$$R_A = \limsup_{k \longrightarrow \infty} R_A(k).$$

*This research was supported by a grant from the Hungarian Academy of Sciences (OTKA Nr. 2037) and by the Christian Doppler Laboratorium für Diskrete Optimierung.

[†]Department of Computer Sciences, Teacher Trainer College, Szeged, Hungary.

[‡]Institute of Mathematik, University Graz, A-8010 Graz, Austria

[§]Institute of Mathematik, Technical University Graz, A-8010 Graz, Austria

The first approximation algorithms for vector-packing were designed by Kou and Markowsky [3] . They defined so-called *irreducible* algorithms as follows. During the packing of an irreducible algorithm, for any two non-empty bins B_p and B_q there exists an index j, $1 \leq j \leq d$ with

$$\sum_{a \in B_p} v_j(a) + \sum_{a \in B_q} v_j(a) > 1.$$

(This means that the algorithm only opens a new bin if a newly arrived item can not be packed into any old bin.) Kou and Markowsky proved the following proposition.

Proposition 1.1 (Kou and Markowsky, [3]) *The asymptotic worst case ratio of any irreducible algorithm fulfills*

$$R_A \leq d + 1.$$

Garey, Graham, Johnson and Yao [1] generalized the First-Fit (FF) and the First-Fit Decreasing (FFD) algorithms to the d-dimensional case. They proved that

$$R_{FF} = d + \frac{7}{10},$$

$$d \leq R_{FFD} \leq d + \frac{3}{10}$$

Note that both of these algorithms are irreducible and hence fulfill the statement of Proposition 1.1.

Now let us turn to lower bounds on the worst case ratios of heuristics. Yao in [6] studied the following class of the "decision-tree" algorithms. Let A be an algorithm for the vector-packing problem. For each $n > 0$, the action of A on a list L can be represented by a ternary tree $T_n(A)$. Each internal node of $T_n(A)$ contains a test. For any input L, the algorithm moves down the tree, testing and branching according to the result of the test, until it reaches some leaf. At the leaf, a packing valid for all lists that lead to this leaf is produced. The cost of A for input size n, $C_n(A)$, is defined to be the number of tests made in the worst-case. (In fact, this is the height of $T_n(A)$). Yao proved that if A is such an algorithm for which $C_n(A) = o(n \log n)$ then $R_A \geq d$.

In this paper we deal with the class of the *on-line* algorithms: If an algorithm A is in this class then it packs the elements one by one in the order given by the list L. After having packed an element into some bin, the element will be never moved again. E.g. algorithm FF mentioned above is an on-line algorithm. For $d \geq 2$ FF has the best worst case ratio among all known on-line heuristics for d-dimensional vector-packing.

As a consequence of the classical result of Liang [5] for one-dimensional on-line bin-packing algorithms, the inequality $R_A \geq 1.5364\ldots$ holds for all $d \geq 1$. Till today there is no better results were known. In this paper we will prove a d-dependent lower bound for on-line vector-packing algorithms. A formula for our lower bounds is given in Theorem 2.1. Table 1 depicts the numerical values for some small dimensions.

The rest of the paper is organized as follows. Section 2 contains some preliminaries and describes the construction of a bad item list for on-line heuristics. Section 3 gives a rigorous proof for the lower bound. Section 4 finishes with the conclusions.

d	Lower Bound	d	Lower Bound
2	1.67072	7	1.87504
3	1.75098	8	1.88891
4	1.80035	9	1.90002
5	1.83348	10	1.90910
6	1.85722	∞	2.00000

Table 1: Our lower bounds, rounded to five decimal places.

2 The construction

We start with defining the following sequence for any fixed $d \geq 1$. (Note that for every d, the reciprocal values $1/t_i(d)$ sum up to $1/2d$).

$$\begin{aligned} t_0(d) &= 2d+1 \\ t_i(d) &= t_{i-1}(d)(t_{i-1}(d)-1)+1, \qquad i \geq 1. \end{aligned}$$

A similar sequence introduced by Golomb [2] became one of the main tools in on-line bin-packing. Lee and Lee [4] used it to design a good one-dimensional bin-packing heuristic, and Liang [5] based his lower bound proof on the Golomb sequence.

With this definition, our main result may be stated as follows.

Theorem 2.1 *For any on-line d-dimensional vector-packing algorithm A, its asymptotic worst case ratio is at least*

$$R_A(d) \geq \frac{2d + \sum_{j=1}^{\infty} \frac{2d+j}{t_j(d)-1}}{\sum_{j=1}^{\infty} \frac{1}{t_j(d)-1} + d + \frac{1}{2}}.$$

Remark. If we set $d = 1$ in Theorem 2.1, we exactly arrive at the well-known lower bound of Liang [5].

The exact values for $2 \leq d \leq 10$ are depicted in Table 1. As d tends to infinity, the lower bound tends to 2. The remaining part of this paper is devoted to the proof of Theorem 2.1.

Intuitively speaking, the underlying idea of our paper is as follows. We construct an adverse strategy that forces *every* on-line algorithm A to behave poorly on a special item list L or on some prefix of L. In the first step, we give A a list of very small items to pack. In case A spreads these items on many bins, it does not receive any further item and looses the game. In case A produces a 'reasonable' packing for the small items, it receives another list of items. Again, A has the choice between either producing a bad packing and loosing the game immediately, or producing a (currently) good packing and receiving another list. Then in the final step, A gets a list of big items. Now it turns out that everything it did before was wrong. It had better packed the smaller items in such a way that remained enough space to pack the big items. A looses the game against the adversary.

Now we start with the definition of the item lists. Let $d \geq 1$ and $r \geq 1$ be arbitrarily fixed integers. We consider the following lists each consisting of n elements.

	L_6	L_5	L_4	L_3	L_2	L_1	L^1	L^2
$v_1(\cdot)$	$\frac{1}{2}+\delta$	$\frac{1}{3}+\delta$	0	0	0	0	0	0
$v_2(\cdot)$	$\frac{1}{4}+\delta$	$\frac{1}{4}+\delta$	$\frac{1}{5}+\delta$	0	0	0	0	0
$v_3(\cdot)$	$\frac{1}{6}+\delta$	$\frac{1}{6}+\delta$	$\frac{1}{6}+\delta$	$\frac{1}{6}+\delta$	$\frac{1}{6}+\delta$	$\frac{1}{7}+\delta$	$\frac{1}{43}+\varepsilon_1$	$\frac{1}{1807}+\varepsilon_2$

Table 2: The elements used in the lists for $d = 3$ and $r = 2$

1. For any $j \in \{1, \dots, r\}$ and $a \in L^j$,
$$v_i(a) = \begin{cases} 0 & \text{if } i < d \\ \frac{1}{t_j(d)} + \varepsilon_j(r) & \text{if } i = d. \end{cases}$$
2. For any $k \in \{1, \dots, d\}$ and $a \in L_{2k-1}$,
$$v_i(a) = \begin{cases} 0 & \text{if} \quad i \le d-k \\ \frac{1}{2i+1} + \delta & \text{if} \quad i = d-k+1 \\ \frac{1}{2i} + \delta & \text{if} \quad i = d-k+p, p = 2, \dots, k. \end{cases}$$
3. For any $k \in \{1, \dots, d\}$ and $a \in L_{2k}$,
$$v_i(a) = \begin{cases} 0 & \text{if} \quad i \le d-k \\ \frac{1}{2i} + \delta & \text{if} \quad i = d-k+p, p = 1, \dots, k. \end{cases}$$

where
$$\delta < \frac{1}{4d(t_{r+1}(d) - 1)},$$
$$\varepsilon_1(r) < \frac{1}{2r(t_{r+1}(d) - 1)},$$
and
$$\varepsilon_{j+1}(r) < \frac{1}{t_j(d) - 1}\varepsilon_j(r), \quad 1 \le j \le r-1.$$

The lists are presented to the on-line heuristic in the following order: First there come the lists L^j with j going down from r to 1, and afterwards there come the lists L_j with j going up from 1 to $2d$. The lists L^j with superscript contain the very small items (all components of the corresponding vectors are zero with the exception of the component with index d). The lists L_j with subscript, $1 \le j \le d$ contain the larger items; list L_{2d} is the list with the big items that arrive in the final step. An illustration for $d = 3$ and $r = 2$ is given in Table 2.

Convention. Next we shall work under a fixed dimension d and a fixed r. To simplify our notations, we shall use t_j and ε_j instead of $t_j(d)$ and $\varepsilon_j(r)$.

3 The Proof

In this section we prove that any on-line heuristic must perform poorly on the list $L = L^r \dots L^1 L_1 \dots L_{2d}$ (as defined in the preceding section) or on some prefix of L.

Observation 3.1 *For any integer* $1 \leq j \leq r$,

$$\sum_{i=j}^{r}\left(\frac{1}{t_i} + \varepsilon_i\right) < \frac{1}{t_j - 1} - 2d\delta.$$

Proof. It can be proved by induction from definitions of t_i, ε_i and δ. □

Lemma 3.2 *For any integer* $n > 0$, *if* $(t_{r+1} - 1)|n$ *then*

$$(L^r \dots L^j)^* \leq \frac{n}{t_j - 1} \quad 1 \leq j \leq r.$$

Proof. In this case $\frac{n}{t_j-1}$ $(j = 1, 2, \dots, r)$ are positive integers. On the other hand, by Observation 3.1, we can pack $t_j - 1$ items of each of the lists $L^r, \dots, L^j$ together into one bin. □

Now for any integer $1 \leq j \leq 2d$, let us define the set N_j in the following way:

$$N_1 = N_2 = \{k(t_{r+1} - 1) : k = 1, 2, \dots\},$$

$$N_j = \{n(2d + 1 - j) : n \in N_{j-1}\} \quad 3 \leq j \leq 2d.$$

It is clear that $N_1 \supseteq N_2 \supseteq \dots \supseteq N_{2d}$.

Lemma 3.3 *For any* $1 \leq j \leq 2d$ *and* $n \in N_j$,

$$(L^r \dots L^1 L_1 \dots L_j)^* \leq \frac{j}{2d} \cdot n.$$

Proof. The statement is proved by induction on j. First, the simple cases $j = 1$ and $j = 2$ are considered; the induction step is structured into two subcases. All we have to do that is to give a feasible packing. Note that Observation 3.1 yields

$$\sum_{i=1}^{r}\left(\frac{1}{t_i} + \varepsilon_i\right) < \frac{1}{t_1 - 1} - 2d\delta.$$

$(j = 1)$. Let $n \in N_1$ be arbitrary. By the definition of N_j, $2d|n$. So we always pack $2d$ elements from each list of $(L^r \dots L^1 L_1)$ together into one bin B. If $i < d$ then for any $a \in B$ $v_i(a) = 0$ holds, and if $i = d$ we have

$$\sum_{a \in B} v_d(a) < 2d\left(\frac{1}{2d+1} + \delta + \frac{1}{2d(2d+1)} - 2d\delta\right) < 1.$$

Hence we have a legal packing, using $\frac{n}{2d}$ bins.

($j = 2$). Let $n \in N_2$ be arbitrary. Then $d|n$. Let us pack together d elements from every list. For $i < d$ $v_i(a) = 0$ holds for each $a \in (L^r \dots L^1 L_1 L_2)$, and for $i = d$ we have

$$\sum_{a \in B} v_d(a) < d\left(\frac{1}{2d+1} + \delta + \frac{1}{2d} + \delta + \frac{1}{2d(2d+1)} - 2d\delta\right) \leq 1.$$

Therefore we obtain a feasible packing, using $\frac{n}{d}$ bins.

(Induction step) Now let $3 \leq j \leq 2d$ and assume that for any positive integer $j' < j$, the statement is valid. Let $n \in N_j$ be arbitrary. We shall distinguish two cases depending on whether j is odd or even.

A. $j = 2l - 1$ for some $2 \leq l \leq d$. In the sequel we say that a non-empty bin has type $\tau = (\tau^r, \dots \tau^1, \tau_1, \dots, \tau_{2d})$ if it contains exactly τ^i resp. τ_i elements from the list L^i resp. L_i. Let us pack the elements of the concatenated list $L^r \dots L^1 L_1 \dots L_j$ together into a bin B with type

$$\tau = (\underbrace{1, \dots, 1}_{2l+r-2}, 2d - 2l + 2, \underbrace{0, \dots, 0}_{2d-2l+1}).$$

First, we will prove that this gives a legal packing, i.e. the following claim holds for the bin B.

Claim 3.4

$$\sum_{a \in B} v_i(a) \leq 1 \qquad 1 \leq i \leq d.$$

Proof. The proof of this claim is divided into cases (i) thru (iv).

(i) If $i \leq d - l$, then $\sum_{a \in B} v_i(a) = 0$.

(ii) If $i = d - l + 1$ then only the elements of L_{2l-1} have non-zero coordinates and therefore

$$\sum_{a \in B} v_i(a) = (2d - 2l + 2)\left(\frac{1}{2d - 2l + 3} + \delta\right) < 1.$$

(iii) If $d - l + 1 < i < d$ then

$$\begin{aligned}
\sum_{a \in B} v_i(a) &= (2d - 2l + 2)\left(\frac{1}{2i} + \delta\right) + (2i - 2 - 2d + j)\left(\frac{1}{2i} + \delta\right) \\
&+ \left(\frac{1}{2i+1} + \delta\right) \\
&= (2i - 1)\left(\frac{1}{2i} + \delta\right) + \left(\frac{1}{2i+1} + \delta\right) < 1.
\end{aligned}$$

(iv) If $i = d$ then

$$\begin{aligned}
\sum_{a \in B} v_i(a) &\leq (2d - 2l + 2)\left(\frac{1}{2d} + \delta\right) + (j - 2)\left(\frac{1}{2d} + \delta\right) \\
&+ \left(\frac{1}{2d+1} + \delta\right) + \left(\frac{1}{2d(2d+1)} - 2d\delta\right) \\
&= \frac{2d-1}{2d} + \frac{1}{2d+1} + \frac{1}{2d(2d+1)} = 1.
\end{aligned}$$

This completes the proof of Claim 3.4 □

To get a feasible packing for $(L^r \dots L^1 L_1 \dots L_j)$, we first take $\frac{n}{2d-2l+2} = \frac{n}{2d-j+1}$ pieces of r type bins. By the definition of N_j, we know that $2d+1-j|n$, and so, we can pack all the elements of L_j into $\frac{\bar{n}}{2d+1-j}$ bins. From the other lists, there remain $\bar{n} = n - \frac{n}{2d+1-j} = \frac{n}{2d+1-j}(2d-j)$ items. By the definition of N_j, $\bar{n} \in N_{j-1}$. But then, by our induction hypothesis, these remaining items can be packed into $\bar{n}\frac{j-1}{2d}$ bins. Therefore, we can pack all elements of $(L^r \dots L^1 L_1 \dots L_j)$ into

$$\frac{n}{2d-j+1} + \bar{n}\frac{j-1}{2d} = n\frac{2d+(j-1)(2d-j)}{(2d+1-j)2d} = n\frac{j}{2d}$$

bins, and case **A** is settled.

B. $j = 2l$, $2 \le l \le d$. In this case we are going to pack $d-l+1$ items using the bin type below:

$$r = (\underbrace{1,\dots,1}_{2l+r-2}, d-l+1, d-l+1, \underbrace{0,\dots,0}_{2d-2l}).$$

Claim 3.5

$$\sum_{a \in B} v_i(a) \le 1 \qquad 1 \le i \le d.$$

Proof. The proof is done in a similar way as the proof of Claim 3.4:

(i) if $i \le d-l$ holds then the above sum is equal to 0,

(ii) if $i = d-l+1$ then only the lists L_{2l-1} and L_{2l} have positive coordinates on the position i

$$\sum_{a \in B} v_i(a) = (d-l+1)(\frac{1}{2i}+\delta) + (d-l+1)(\frac{1}{2i+1}+\delta) < 1,$$

(iii) if $d-l+1 < i < d$ then

$$\begin{aligned}\sum_{a \in B} v_i(a) &= (2d-2l+2)(\frac{1}{2i}+\delta) + (2i-3-2d+j)(\frac{1}{2i}+\delta) + (\frac{1}{2i+1}+\delta) \\ &= (2i-1)(\frac{1}{2i}+\delta) + (\frac{1}{2i+1}+\delta) < 1,\end{aligned}$$

(iv) if $i = d$ then

$$\begin{aligned}\sum_{a \in B} v_i(a) &\le (2d-2l+2)(\frac{1}{2d}+\delta) + (j-3)(\frac{1}{2d}+\delta) + (\frac{1}{2d+1}+\delta) \\ &+ (\frac{1}{2d(2d+1)} - 2d\delta) = 1.\end{aligned}$$

Thus, Claim 3.5 is true.

To obtain a feasible packing for $(L^r \dots L^1 L_1 \dots L_j)$, we first take $\frac{n}{d-l+1}$ pieces of τ type bins. By the definition of N_j, from $n \in N_j$ it follows that $n = (2d+1-j)(2d+2-j)n'$ with $n' \in N_{j-2}$, provided that $j \geq 4$. But then $n = 2(2d+1-j)(d-l+1)n'$. Therefore, $d - l + 1|n$, and so, we can pack all the elements of L_{j-1} and L_j into $\frac{n}{d-l+1}$ bins. After this packing each list from $(L^r, \dots, L^1, L_1, \dots, L_{j-2})$ contains $\bar{n}$ unpacked elements where $\bar{n} = n - \frac{n}{d-l+1} = \frac{n}{d-l+1}(d-l)$.

Now let us observe that $\bar{n} \in N_{j-2}$. Then, by our induction hypothesis, the unpacked items can be packed into $\bar{n}\frac{j-2}{2d}$ bins. Therefore, we can pack all elements of $(L^r \dots L^1 L_1 \dots L_j)$, into

$$\frac{n}{d-l+1} + \bar{n}\frac{j-2}{2d} = n\frac{2d + (j-2)(d-l)}{(d-l+1)2d} = n\frac{j}{2d}$$

bins, which completes the considered case and the proof of Lemma 3.3 too. □

Lemmas 3.2 and 3.3 give us upper bounds for the number of bins in the optimal packings. Next, we will investigate the potential behaviour of arbitrary on-line algorithms on the constructed list L. We introduce the following notations:

- $\beta = \{B_1, \dots, B_{A(L^r \dots L^1 L_1 \dots L_{2d})}\}$ denotes the final packing of the concatenated list $(L^r \dots L^1 L_1 \dots L_{2d})$ produced by the on-line heuristic A. For any type $\tau = (\tau^r \dots \tau^1 \tau_1 \dots \tau_{2d})$, the number $a(\tau)$ equals the number of bins of type τ in the packing β.
- The subset β^i resp. β_j, contain only those bins which were used for the first time by the on-line heuristic A during the packing of the list L^i resp. L_j (i.e. their first item comes from L^i resp. L_j). Moreover, define for every $1 \leq i \leq r$ and $1 \leq j \leq 2d$ the sets:

  $T^i = \{\tau : \text{there exists a bin of type } \tau \text{ in } \beta^i\}$,

  $T_j = \{\tau : \text{there exists a bin of type } \tau \text{ in } \beta_j\}$,

  and

  $T = \{\tau : \text{there exists a bin of type } \tau \text{ in } \beta\} = \bigcup_{1 \leq i \leq r} T^i \cup \bigcup_{1 \leq j \leq 2d} T_j$.

Now we investigate the number of bins used by an arbitrary on-line algorithm A while A is packing the elements of the concatenated list $(L^r \dots L^1 L_1 \dots L_j)$.

$$A(L^r \dots L^i) = \sum_{l=i}^{r} \sum_{\tau \in T^l} a(\tau), \qquad 1 \leq i \leq r, \tag{1}$$

$$A(L^r \dots L^1 L_1 \dots L_j) = \sum_{l=1}^{r} \sum_{\tau \in T^l} a(\tau) + \sum_{l=1}^{j} \sum_{\tau \in T_l} a(\tau) \qquad 1 \leq j \leq 2d \tag{2}$$

and the number of the packed elements for each i resp. j, $1 \leq i \leq r$, $1 \leq j \leq 2d$:

$$n = \sum_{\tau \in T} \tau^i a(\tau), \qquad 1 \le i \le r. \tag{3}$$

$$n = \sum_{\tau \in T} \tau_j a(\tau), \qquad 1 \le j \le 2d. \tag{4}$$

Let us multiply the equations of (3) by $\frac{2d+i}{t_i-1}$. Summarizing the equations of (1) - (2) and subtracting the multiplied equations of (3) and (4) we get:

$$\sum_{i=1}^{r} A(L^r \dots L^i) + \sum_{j=1}^{2d} A(L^r \dots L^1 L_1 \dots L_j) - 2dn - n\sum_{i=1}^{r} \frac{2d+i}{t_i - 1} =$$

$$= \sum_{i=1}^{r}(2d+i) \sum_{\tau \in T^i} a(\tau) + \sum_{j=1}^{2d}(2d-j+1) \sum_{\tau \in T_j} a(\tau) - \tag{5}$$

$$\sum_{\tau \in T} a(\tau)(\sum_{i=1}^{r} \frac{2d+i}{t_i - 1}\tau^i + \sum_{j=1}^{2d} \tau_j).$$

Lemma 3.6 *The right hand side of (5) is non-negative.*

Proof. The proof is constructed into three parts.

A. First we prove that for any $1 \le i \le r$ and $\tau \in T^i$

$$\sum_{s=1}^{r} \frac{2d+s}{t_s - 1}\tau^s + \sum_{v=1}^{2d} \tau_v \le 2d + i.$$

Since $\tau \in T^i$, $\tau^r = \dots = \tau^{i+1} = 0$ and $\tau^i > 0$. Now if we have some component $\tau_v > 0$ for some v (i.e. some item from L_v is contained in the corresponding bin), then we replace this item by $2d$ elements of L^1. After the replacement we obtain a feasible packing of the considered bin and a new bin type $\bar{\tau}$ which is not neccessarily contained in T^i, but its first nonzero component is $(\bar{\tau})^i$. On the other hand, it is easy to check that the weighted sums on the left hand side do not decrease. Therefore, it is enough to prove that for any bin type τ of the items from the lists $L^r, \dots, L^1, L_1, \dots, L_{2d}$, if $\tau^r = \dots = \tau^{i+1} = 0$, then

$$\sum_{s=1}^{r} \frac{2d+s}{t_s - 1}\tau^s \le 2d + i.$$

Now we replace each element of L^u by $t_u - 1$ elements of L^{u+1}. This replacement results a feasible packing, since

$$(t_u - 1)(\frac{1}{t_{u+1}} + \varepsilon_{u+1}) \le \frac{1}{t_u} + \varepsilon_u.$$

On the other hand, the weighted sum in the newly constructed packing increases:

$$(t_u - 1)\frac{2d+u+1}{t_{u+1}-1} = \frac{2d+u+1}{t_u} > \frac{2d+u}{t_u-1}.$$

Repeating this procedure for every $u < i$, we finally obtain a feasible packing with only items from L^i and with an increased weighted sum. Since for every feasible packing in a bin, $\tau^i \le t_i - 1$ holds, we obtain the desired result.

B. Secondly, we prove that for any $1 \le j \le 2d$ and $\tau \in T_j$

$$\sum_{v=1}^{2d} \tau_v (= \sum_{v=j}^{2d} \tau_v) \le 2d - j + 1.$$

B1. Let us consider the subcase $j = 2k$, $1 \le k \le d$. We examine the $(d-k+1)$-th coordinate of the list $L_j, \ldots, L_{2d}$. Because of the definitions, it follows for each list that for any $a \in (L_j \ldots L_{2d})$, $v_{d-k+1}(a) = \frac{1}{2d-j+2} + \delta$, and so the statement is true.

B2. If $j = 2k-1$ then we again consider the $(d-k+1)$-th coordinate. Now the smallest elements in this coordinate are those ones which belong to the list L_j: if $a \in L_j$ then $v_{d-k+1}(a) = \frac{1}{2d-j+2} + \delta$, and so the desired inequality holds.

C. Finally, we prove that the right hand side of (5) is nonnegative. Indeed, by case **A**, we obtain

$$\begin{aligned}
\sum_{i=1}^{r}(2d+i)\sum_{\tau\in T^i} a(\tau) &= \sum_{i=1}^{r}\sum_{\tau\in T^i} a(\tau)(2d+i) \\
&\ge \sum_{i=1}^{r}\sum_{\tau\in T^i} a(\tau)\Big(\sum_{s=1}^{r}\frac{2d+s}{t_s-1}\tau^s + \sum_{v=1}^{2d}\tau_v\Big) \\
&= \sum_{\tau\in\cup_{1\le i\le r}T^i} a(\tau)\Big(\sum_{s=1}^{r}\frac{2d+s}{t_s-1}\tau^s + \sum_{v=1}^{2d}\tau_v\Big).
\end{aligned}$$

On the other hand by the case **B**,

$$\begin{aligned}
\sum_{j=1}^{2d}(2d-j+1)\sum_{\tau\in T_j} &= \sum_{j=1}^{2d}\sum_{\tau\in T_j} a(\tau)(2d-j+1) \\
&\ge \sum_{j=1}^{2d}\sum_{\tau\in T_j} a(\tau)\Big(\sum_{v=1}^{2d}\tau_v\Big) \\
&= \sum_{\tau\in\cup_{1\le j\le 2d}T_j} a(\tau)\Big(\sum_{v=1}^{2d}\tau_v\Big).
\end{aligned}$$

Let us observe that for any $1 \le j \le 2d$ and $\tau \in T_j$, $\tau^r = \ldots = \tau^1 = 0$, and so

$$\sum_{\tau\in\cup_{1\le j\le 2d}T_j} a(\tau)\Big(\sum_{s=1}^{r}\frac{2d+s}{t_s-1}\tau^s\Big) = 0.$$

Therefore the last three inequalities give us that the considered right hand side is nonnegative which completes the proof of Lemma (3.6). □

Now we are ready to prove of Theorem 2.1. For this reason let $n \in N_{2d}$ be arbitrary. Lemma 3.6 together with equation (5) yields

$$\sum_{i=1}^{r} A(L^r \dots L^i) + \sum_{j=1}^{2d} A(L^r \dots L^1 L_1 \dots L_j) \geq 2dn + n \sum_{i=1}^{r} \frac{2d+i}{t_i - 1} \tag{6}$$

We define

$$r^i = \frac{A(L^r \dots L^i)}{(L^r \dots L^i)^*} \qquad 1 \leq i \leq r$$

$$r_j = \frac{A(L^r \dots L^1 L_1 \dots L_j)}{(L^r \dots L^1 L_1 \dots L_j)^*} \qquad 1 \leq j \leq 2d$$

and

$$R = \max\left\{\max_i r^i, \max_j r_j\right\}.$$

Now plugging R into (6) and using the results stated in Lemmas 3.2 and 3.3, we get

$$nR \sum_{i=1}^{r} \frac{1}{t_i - 1} + R \frac{n}{2d} \sum_{j=1}^{2d} j \geq n(2d + \sum_{i=1}^{r} \frac{2d+i}{t_i - 1})$$

Finally, dividing by n and making $r \to \infty$ yields the statement of Theorem 2.1 □

4 Conclusion

In this paper we derived the first non-trivial lower bound for d-dimensional on-line vector packing algorithms. The best on-line algorithm known today, the First-Fit algorithm has asymptotic worst case ratio $d + \frac{7}{10}$. In relation to this result, our lower bound is not too attractive, as it remain beneath 2 for any given d and there is a wide gap to the upper bound.

Of course, the main open (and probably very hard) problem consists in giving a better lower bound for on-line approximation algorithms that tends to infinity as d tends to infinity, e. g. $\Omega(\sqrt{d})$ or $\Omega(\log d)$. Moreover, we invite the researchers to design better on-line algorithms with smaller asymptotic worst-case ratios. A good candidate might be the vector-generalization of the *Harmonic Fit* algorithm analysed by Lee and Lee [4].

Acknowledgment. We thank Günter Rote and Balázs Imreh for constructive criticisms on the earlier version of this paper.

References

[1] M.R.Garey, R.L.Graham, D.S. Johnson and A.C.C. Yao, Resource constrained scheduling as generalized bin packing, *J. Comb. Th. Ser. A.* **21**, (1976), 257-298.

[2] S. Golomb, On certain non-linear sequences, *American Math. Monthly* **70**, (1963), 403-405.

[3] L.T. Kou and G. Markowsky, Multidimensional Bin Packing Algorithms, *IBM Journal of Research and Development*, (1977), 443-448.

[4] C.C. Lee and D.T. Lee, A Simple On-line Bin Packing Algorithm, *J. Assoc. Comp. Mach.* **32**, (1985), 562-572.

[5] F.M. Liang, A Lower Bound for On-line Bin Packing, *Inf. Proc. Letters* **10**, (1980), 76-79.

[6] A.C.C.Yao, New Algorithms for Bin Packing, *J. Assoc. Comp. Mach.* **27**, (1980), 207-227.

(Received May 20, 1991.)

(Revised September 10, 1993 and November 15, 1993.)

Acta Cybernetica, Vol. 11, No. 1-2, Szeged, 1993

Some problems concerning Armstrong relations of dual schemes and relation schemes in the relational datamodel*

J. Demetrovics† V. D. Thi†

Abstract

Several papers [3,5,6,7,8,9,11,12] have appeared for investigating dual dependency. The practical meaning of dual dependency was shown in [5,6]. In this paper we give some new results concerning dual dependency. The concept of dual scheme is introduced. Some characterizations of dual scheme, such as closure, generator, generating Armstrong relation, inferring dual dependencies, irredundant cover, normal cover are studied from different aspects. We give a characterization of Armstrong relations for a given dual scheme. We prove that the membership problem for dual dependencies is solved by a polynomial time algorithm. We show that the time complexity of finding an Armstrong relation of a given dual scheme is exponential in the number of attributes. Conversely, we give an algorithm to construct a dual scheme from a given relation R such that R is Armstrong relation of it. This paper gives some polynomial time algorithms which find closure, irredundant cover, normal cover from a given dual scheme.

In the second part of this paper we present some results related to Armstrong relations for functional dependency (FD for short) in Boyce-Codd normal form. The concepts of unique relation and unique relation scheme are introduced. We prove that deciding whether a given relation R over a set of attributes U is unique is solved by a polynomial time algorithm. We show some cases in which FD-relation equivalence problem is solved in polynomial time.

Key Words and Phrases: relation, relational datamodel, dual dependency, dual scheme, generating Armstrong relation, inferring dual dependencies, membership problem, closure, closed set, irredundant cover, normal cover, minimal generator, Boyce-Codd normal form.

1 Introduction

Now we give some necessary definitions that are used in next sections. The next sections present our new results.

*Research supported by Hungarian Foundation for Scientific Research Grant-2575.

†Computer and Automation Institute Hungarian Academy of Sciences P.O.Box 63, Budapest, Hungary, H-1502

Definition 1.1 *Let $R = \{h_1, \ldots, h_m\}$ be a relation over U, and $A, B \subseteq U$. Then we say that B dually depends on A in R denoted $A \xrightarrow[R]{d} B$) iff*

$$(\forall h_i, h_j \in R)(\exists a \in A)(h_i(a) = h_j(a)) \Longrightarrow (\exists b \in B)(h_i(b) = h_j(b)))$$

Let $D_R = \{(A, B) : A, B \subseteq U,\ A \xrightarrow[R]{d} B\}$. D_R is called the full family of dual dependencies of R. Where we write (A, B) or $A \to B$ for $A \xrightarrow[R]{d} B$ when R, d are clear from the context.

Definition 1.2 *A dual dependency (DD) over U is a statement of the form $A \to B$,where $A, B \subseteq U$. The DD $A \to B$ holds in a relation R if $A \xrightarrow[R]{d} B$ We also say that R satisfies the DD $A \to B$.*

Definition 1.3 *Let U be a finite set, and denote $P(U)$ its power set. Let $Y \subseteq P(U) \times P(U)$. We say that Y is a d-family over U iff for all $A, B, C, D \subseteq U$*

(1) $(A, A) \in Y$,

(2) $(A, B) \in Y,\ (B, C) \in Y \Longrightarrow (A, C) \in Y$,

(3) $(A, B) \in Y,\ C \subseteq A,\ B \subseteq D \Longrightarrow (C, D) \in Y$,

(4) $(A, B) \in Y,\ (C, D) \in Y \Longrightarrow (AUC, BUD) \in Y$.

(5) $(A, \emptyset) \in Y \Longrightarrow A = \emptyset$.

Clearly, D_R is a d-family over U.

It is known [6,7] that if Y is an arbitrary d-family, then there is a relation R over U such that $D_R = Y$.

Definition 1.4 *A dual scheme P is a pair $< U, D >$, where U is a set of attributes, and D is a set of DDs over U. Let D^+ be a set of all DDs that can be derived from D by the rules in Definition 1.3. It is easy to see that D^+ is a d-family over U.*

Clearly, if $P =< U, D >$ is a dual scheme, then there is a relation R over U such that $D_R = D^+$ (see, [6,7]). Such a relation is called an Armstrong relation of P.

In this paper we consider the comparision of two attributes as an elementary step of algorithms. Thus, if we assume that subsets of U are represented as sorted lists of attributes, then a Boolean operation on two subsets requires at most $|U|$ elementary steps.

Definition 1.5 *Let $I \subseteq P(U)$, $U \in I$, and $A, B \in I \Longrightarrow A \cap B \in I$. Let $M \subseteq P(U)$. Denote $M^+ = \{\cap M' : M' \subseteq M\}$. We say that M is a generator of I iff $M^+ = I$. Note that $U \in M^+$ but not necessarily in M, since it is the intersection of the empty collection of sets.*

Denote $N = \{A \in I : A \neq \cap\{A' \in I : A \subset A'\}\}$.

It is proved [7] that N is the unique minimal generator of I. Thus, for any generator N' of I we obtain $N \subseteq N'$.

Definition 1.6 *Let D be a d-family over U, and $(A, B) \in D$. (A, B) is called a maximal left-side dependency of D if $\forall A' : A \subseteq A', (A', B) \in D \Longrightarrow A' = A$. Denote by $M(D)$ the set of all maximal left-side dependencies of D. Then A is called a maximal left-side of D if there existst a B such that $(A, B) \in M(D)$. Denote by $G(D)$ the set of all maximal left-sides of D.*

Definition 1.7 *Let $G \subseteq P(U)$. We say that G is a d-semilattice over U if $\emptyset, U \in G$, $A, B \in G \Longrightarrow A \cap B \in G$.*

Theorem 1.8 [6] *Let D be a d-family over U. Then $G(D)$ is a d-semilattice over U. Conversely, if G is a d-semilattice over U, then there exists exactly one d-family D such that $G(D) = G$, where $D = \{(A, B) : \forall C \in G : A \not\subseteq C \Longrightarrow B \not\subseteq C\}$.*

Theorem 1.9 *Let K be a Sperner system over U. We define the set of antikeys of K, denoted by K^{-1}, as follows:*

$$K^{-1} = \{A \subset U : (B \in K) \Longrightarrow (B \not\subseteq A) \, and \, (A \subset C) \Longrightarrow (\exists B \in K)(B \subseteq C)\}$$

It is easy to see that K^{-1} is also a Sperner system over U.

2 Dual schemes

Definition 2.1 *Let R be a relation over U. Set $N_{ij} = \{a \in U : h_i(a) \neq h_j(a)\}$, and $N_R = \{N_{ij} : 1 \leq i < j \leq |R|\}$. Then N_R is called the non-equality system of R.*

According to definition of relation $\emptyset \notin N_R$.

Let $P =< U, D >$ a dual scheme over U. Then D^+ is a d-family over U, $G(D^+)$ is the set of all maximal left-sides of D^+. Clearly, $G(D^+)$ is a d-semilattice over U. Denote by $N(D^+)$ the minimal generator of $G(D^+)$.

Now we present a characterization of Armstrong relations for a given dual scheme.

Theorem 2.2 *Let $P =< U, D >$ be a dual scheme, R be a relation over U. Then R is an Armstrong relation of P if and only if $N(D^+) \subseteq N_R \cup \{\emptyset\} \subseteq G(D^+)$.*

Proof: ($\Longrightarrow$): We assume that R is an Armstrong relation of P, i.e. $D_R = D^+$. According to Theorem 1.8 we obtain $G(D_R) = G(D^+)$. Now we prove that for an arbitrary relation R $G(D_R) = (N_R - U)^+ \cup \{\emptyset\}$ holds. Because $G(D_R)$ is a d-family over U, we have $\emptyset, U \in G(D_R)$. Clearly, $U \in (N_R - U)^+$. It is obvious that $\forall N_{ij} \neq \emptyset$. We suppose that $N_{ij} \neq U$.Because for any $a \in U - N_{ij}$ we obtain $h_i(a) = h_j(a)$, but $\forall b \in N_{ij}$: $h_i(b) \neq h_j(b)$,i.e. $\{a\} \cup N_{ij} \xrightarrow[R]{d} N_{ij}$. Hence, $N_{ij} \in G(D_R)$, holds. Consequently, $N_R \subseteq G(D_R)$. Thus, we obtain $(N_R - U)^+ \cup \{\emptyset\} \subseteq G(D_R)$. Conversely, if $A \in G(D_R) - \{\emptyset, U\}$, then if we suppose that for all $h_i, h_j \in R$ then there is $a \in A$ such that $h_i(a) = h_j(a)$. So $U \xrightarrow[R]{d} A$ which contradicts the definition of A. Consequently, there is an index pair (i, j) such that $A \subseteq N_{ij}$. We set $T = \{N_{ij} : A \subseteq N_{ij}\}$. If there exists an $N_{ij} : A = N_{ij}$ then $A \in N_R$. In the converse case we set $B = \bigcap_{N_{ij} \in T} N_{ij}$. If $A \subset B$ then for all $N_{ij} \in T$ we have $A \subset N_{ij}$.So

$B \xrightarrow[R]{d} A$ which contradicts $A \in G(D_R) - \{\emptyset, U\}$.Consequently, we obtain $A = B$. Hence, $A \in (N_R - U)^+ \cup \{\emptyset\}$ holds. Thus, $G(D_R) = (N_R - U)^+ \cup \{\emptyset\}$ holds. Consequently, we have $G(D^+) = (N_R - U)^+ \cup \{\emptyset\}$. According to definition of minimal generator we obtain $N(D^+) \subseteq N_R \cup \{\emptyset\} \subseteq G(D^+)$.

($\Longleftarrow$):From $N(D^+) \subseteq N_R \cup \{\emptyset\} \subseteq G(D^+)$ we have $G(D^+) = (N_R - U)^+ \cup \{\emptyset\}$. According to above part of proof we obtain $G(D_R) = G(D^+)$. By Theorem 1.8 R is an Armstrong relation of P. The theorem is proved.

Let $P =< U, D >$ be a dual scheme. We set $H_P(A) = \{a \in U : \{a\} \to A \in D^+\}$. Let $Z(P) = \{A \in P(U) : H_P(A) = A\}$. It is easy to see that $Z(P) = G(D^+)$. Clearly, for all $A \in P(U) : A \subseteq H_P(A) = H_P(H_P(A))$ and $A \subseteq B \Longrightarrow H_P(A) \subseteq H_P(B)$.

Algorithm 2.3 *(Compute $H_P(A)$)*

Input: $P =< U, D = \{A_i \to B_i : i = 1, \ldots, m\} >$ a dual scheme over U, $A \in P(U)$.

Output: $H_P(A)$

Step 1: We set $A(0) = A$.

Step $i+1$: If there is an $A_j \to B_j \in D$ such that $B_j \subseteq A(i)$ and $A_j \not\subseteq A(i)$, then we set $A(i+1) = A(i) \cup (\bigcup_{B_j \subseteq A(i)} A_j)$. In the converse case we set $H_P(A) = A(i)$.

It can be seen that there is a t such that $A = A(0) \subseteq A(1) \subseteq \ldots \subseteq A(t) = A(t+1) = \ldots$

By rules (3) and (4) in Definition 1.3 it can be seen that the DD $\{a_{i1}, \ldots, a_{it}\} \to B$ is equivalent to a set of DDs $\{\{a_{i1}\} \to B, \ldots, \{a_{it}\} \to B\}$. Consequently, we can assume that D only contains the DDs form $\{a\} \to B$. Clearly, if $A \neq \emptyset$ then $A \to \emptyset \notin D$.

In [2] the notion of a F-based derivation tree for functional dependency is introduced, in the analogous way we present a derivation tree for dual dependency as follows.

Definition 2.4 *Let $P =< U, D >$ be a dual scheme and D only contains the DDs form $\{a\} \to B$. The set of derivation trees (DT for short) over P is constructed as follows:*

1. *A node labeled with a is a DT,where $a \in U$.*
2. *If a is label of a leaf of DT Q and $\{a\} \to \{b_1, \ldots, b_t\} \in D$. Then we replace this leaf in Q by the subtree whose root labeled with a and $b_1, \ldots, b_t$ as chidren of root.An obtained tree is a DT.*
3. *Nothing else is a DT.*

Remark 2.5 *Let $P =< U, D >$ be a dual scheme and D only contains the DDs form $\{a\} \to B$. We call a sequence DDs $(d_1, \ldots, d_m)$ is a derivation of a DD $E \to F$ over P if $d_m = E \to F$ and for each $i (1 \leq i \leq m)$ one of the following holds:*

(1) $d_i \in D$ or $d_i = A \to A$

(2) d_i is the result of applying rule (2) to two of DDs $d_1, \ldots, d_{i-1}$

(3) d_i is the result of applying rule (3) to one of DDs $d_1, \ldots, d_{i-1}$

(4) d_i is the result of applying rule (4) to two of DDs $d_1, \ldots, d_{i-1}$.

Where rules (2),(3),(4) in Definition 1.3.

Proposition 2.6 *By Algorithm 2.3 we obtain $H_P(A) = A(t)$ and the time complexity of Algorithm 2.3 is polynomial in the size of P.*

Proof: It is easy to see that the time complexity of Algorithm 2.3 is polynomial in the size of P. Now we have to prove that $a \in A(t)$ iff $a \in H_P(A)$.

($\Longrightarrow$) : We prove by the induction. It is obvious that $a \in A(0) = A \subseteq H_P(A)$. We assume that $A(i) \subseteq H_P(A)$, and $a \in A(i+1) - A(i)$.

According to construction of Algorithm 2.3 there exists $A_j \rightarrow B_j \in D$ such that $B_j \subseteq A(i), a \in A_j - A(i)$. By (2) and (3) of Definition 1.3 we have $\{a\} \rightarrow B_j$. By $B_j \subseteq A(i)$ and (3) of Definition 1.3 $B_j \rightarrow A(i)$ holds. According to the inductive hypothesis $A(i) \rightarrow A$ holds. Consequently, by (2) of Definition 1.3 we obtain $\{a\} \rightarrow A$. Thus, $a \in H_P(A)$ holds.

($\Longleftarrow$): We can assume that D only contains the DDs form $\{a\} \rightarrow B$. By induction on the length of the derivation of $\{a\} \rightarrow F$ we can show that if $\{a\} \rightarrow F \in D^+$ then there is a DT with root labeled a and a set of leaves of this DT is a subset of F. This proof is in the analogous way as for functional dependency ,see [2], it will be omitted. From this consider and based on the notion of DT by induction on the depth of derivation trees we can show that if $a \in H_P(A)$ then $a \in A(t)$. This proof is easy, it will be omitted. Our proof is complete.

It can be seen that $A \rightarrow B \in D^+$ iff $A \subseteq H_P(B)$. From this and by Algorithm 2.3 the following proposition is clear.

Proposition 2.7 *(The membership problem)*

Let $P =< U, D >$ be a dual scheme. $X \rightarrow Y$ is a dual dependency. Then there exists a polynomial time algorithm deciding whether $X \rightarrow Y \in D^+$.

Let D be a d-family over U, $G(D)$ is the set of all maximal left-sides of D. Denote by $N(D)$ the minimal generator of $G(D)$. Denote $s(D) = min\{m: |R| = m, D_R = D\}$.

Theorem 2.8 [11] $(2|N(D)|)^{1/2} \leq s(D) \leq 2|N(D)|$.

Theorem 2.9 *(Generating Armstrong relation for a given dual scheme) The time complexity of finding Armstrong relation of a given dual scheme P is exponential in the size of P.*

Proof: Let $P =< U, D >$ be a dual scheme. We set $H_P(A) = \{a \in U : \{a\} \rightarrow A \in D^+\}$. Let $Z(P) = \{A \in P(U) : H_P(A) = A\}$. It is easy to see that $Z(P) = G(D^+)$. Thus, $N(D^+)$ is the minimal generator of $Z(P)$. First we contruct an exponential time algorithm that finds a relation R such that $D_R = D^+$. From P we compute $Z(P)$ by Algorithm 2.3. After that we construct the minimal generator of $Z(P)$. We assume that $N(D^+) = \{A_1, \ldots, A_s\}$. Construct a relation $R = \{h_1, h_2, \ldots, h_{2s-1}, h_{2s}\}$ as follows:

$\forall i = 1, \ldots, s \quad \forall a \in U: h_{2i-1}(a) = 2i - 1$

$$h_{2i}(a) = \begin{cases} 2i & \text{if } a \in A_i \\ 2i-1 & \text{otherwise.} \end{cases}$$

According to Theorem 2.2 we obtain $D_R = D^+$.

Let us take a partition $U = X_1 \cup, \ldots, \cup X_m \cup W$, where $m = [n/3]$, and $|X_i| = 3$ $(1 \leq i \leq m)$.

We set

$H = \{B: |B| = 2, B \subseteq X_i \text{ for some } i\}$ if $|W| = 0$,
$H = \{B: |B| = 2, B \subseteq X_i \text{ for some } i : 1 \leq i \leq m-1 \text{ or } B \subseteq X_m \cup W\}$ if $|W| = 1$,
$H = \{B: |B| = 2, B \subseteq X_i \text{ for some } i : 1 \leq i \leq m \text{ or } B = W\}$ if $|W| = 2$.
It is easy to see that
$H^{-1} = \{A: |A \cap X_i| = 1, \forall i\}$ if $|W| = 0$,
$H^{-1} = \{A: |A \cap X_i| = 1, (1 \leq i \leq m-1) \text{ and } |A \cap (X_m \cup W)| = 1\}$ if $|W| = 1$,
$H^{-1} = \{A: |A \cap X_i| = 1, (1 \leq i \leq m) \text{ and } |A \cap W| = 1\}$ if $|W| = 2$.

It is clear that $n - 1 \leq |H| \leq n + 2, 3^{[n/4]} < |H^{-1}|$. We construct a dual scheme $P = < U, D = \{U \rightarrow B: B \in H\} >$. Based on Definition 1.9 and by Algorithm 2.3 we obtain $H^{-1} \subseteq N(D^+)$. By Theorem 2.8 we have $(2|N(D^+)|)^{1/2} \leq s(D^+)$. Consequently, we obtain $3^{[n/8]} < s(D^+)$. Based on the definition of $s(D^+)$ it can be seen that we always can construct a dual scheme P such that the number of rows of any Armstrong relation of P is exponential in the size of P. Our proof is complete.

Algorithm 2.10 *(Inferring dual dependencies)*

Input: a relation $R = \{h_1, \ldots, h_m\}$ over U.
Output: a dual scheme $P = < U, D >$ such that $D_R = D^+$.
Step 1: Find the non-equality system $NE_R = \{N_{ij} : 1 \leq i < j \leq m\}$, where $N_{ij} = \{a \in U : h_i(a) \neq h_j(a)\}$,
Step 2: Find the minimal generator N, where $N = \{A \in NE_R : A \neq \cap\{B \in NE_R : A \subset B\}\}$.
Denote elements of N by $A_1, \ldots, A_s$.
Step 3: For every $B \subseteq U$ if there is A_i such that $B \subseteq A_i$, we compute $C = \bigcap_{B \subseteq A_i} A_i$ and set $C \rightarrow B$. In the converse case we set $U \rightarrow B$.
Denote T the set of all such dual dependencies
Step 4: Set $D = T - Q$, where $Q = \{X \rightarrow Y \in T : X = Y \text{ or there is } X \rightarrow Y' \in T: Y' \subseteq Y\}$.

Clearly, according to Theorem 2.2, Algorithm 2.10 finds a relation scheme P such that a given relation R is an Armstrong relation of P.

Definition 2.11 *Let $P = < U, D >, P' = < U, D' >$ be two dual schemes. We say that P' is a cover of P if $D'^{+} = D^+$. It is obvious that P also is a cover of P'.*

It can be seen that if P, P' are dual schemes over U then based on Proposition 2.7 and Algorithm 2.3 there is a polynomial time algorithm deciding whether $D^+ = D'^{+}$.

Definition 2.12 *Let $P = < U, D >$, $D = \{A_i \rightarrow B_i : i = 1, \ldots, m\}$ be a dual scheme. We say that P is an irredundant cover if for all $T \subset D : D^+ \neq T^+$.*

Now we give an algorithm to find an irredundant cover of a given dual scheme.

Algorithm 2.13 *(Finding an irredundant cover)*

Input : Let $P = < U, D = \{A_i \rightarrow B_i : i = 1, \ldots, m\} >$ be a dual scheme.
Output : $P' = < U, D' >$ is an irredundant cover of P.
Step 1: Set $L(1) = D$

Step (i+1) : Set $Q = L(i) - \{A_i \to B_i\}$, and

$$L(i+1) = \begin{cases} Q & \text{if } A_i \to B_i \in Q^+ \\ L(i) & \text{otherwise.} \end{cases}$$

Then we set $D' = L(m+1)$.

Proposition 2.14 $< U, L(m+1) >$ *is an irredundant cover of P.*

Proof: First we show that $< U, L(i+1) >$ is a cover of $< U, L(i) >$. If $L(i+1) = Q$ then by $A_i \to B_i \in Q^+$ we have $L(i)^+ = L(i+1)^+$. If $L(i+1) = L(i)$ it is obvious that $L(i+1)^+ = L(i)^+$. So we have $D^+ = L(1)^+ = \ldots = L(m+1)^+ = D'^+$. Now we show that $< U, D' >$ is irredundant. Suppose that there is an irredundant cover $< U, L >$ of P such that $L \subset L(m+1)$. Thus, there is a DD $A_j \to B_j \in L(m+1)$ but $A_j \to B_j \notin L$, where $1 \leq j \leq m$. From the definition of $L(j+1)$ we obtain $A_j \to B_j \notin Q^+$, where $Q = L(j) - \{A_j \to B_j\}$. Since $L(m+1) \subseteq L(j)$ it follows that $A_j \to B_j \notin Q'^+$, where $Q' = L(m+1) - \{A_j \to B_j\}$. Clearly, $Q' \subseteq Q$, $L \subseteq L(m+1) - \{A_j \to B_j\}$ hold. Consequently, $A_j \to B_j \notin L^+$. This conflicts with the fact that $L^+ = D^+$. Our proof is complete.

Let $P =< U, D >$ be a dual scheme. We can assume that the set D only contains the DDs form $\{a\} \to B$. Based on this we give the next definition

Definition 2.15 *Let $P =< U, D >$ be a dual scheme. P is called a normal dual scheme if P is irredundant and the following properties hold :*

(1) D only contains the DDs form $\{a\} \to B$, where $a \in U, B \in P(U)$,

(2) for all $\{a\} \to B \in D$ and $B' \subset B$: $< U, D - \{\{a\} \to B\} \cup \{\{a\} \to B'\} >$ is not a cover of P.

Proposition 2.16 *Let $P =< U, D >$ be a dual scheme. Then there is an algorithm finding a normal cover of P. The time complexity of it is polynomial in the size of P.*

Proof: (1) is clear. Consequently, we assume that D only contains the DDs form $\{a\} \to B$. Based on Algorithm 2.13 from P we construct an irredundant dual scheme P' which is a cover of P. Assume that $P' =< U, D' = \{\{a_i\} \to B_i : i = 1, \ldots, t\} >$, and $B_i = \{b_{i1}, \ldots, b_{ih}\}$. For each $i (1 \leq i \leq t)$ we set $E(1) = B_i$, for $j = 1, \ldots, h$

$$E(j+1) = \begin{cases} E(j) - b_{ij} & \text{if } \{a\} \to \{E(j) - b_{ij}\} \in D'^+ \\ E(j) & \text{otherwise.} \end{cases}$$

Denote $T_i = E(h+1)$. According to Algorithm 2.3 and Proposition 2.7 we compute T_i in polynomial time in the size of P'. By induction we can show that $\{a_i\} \to T_i \in D'^+$ and $\forall T \subset T_i$ we obtain $\{a_i\} \to T \notin D'^+$. This is clear and so its proof will be omitted. Now we set $P'' =< U, D'' = \{\{a_i\} \to T_i : i = 1, \ldots, t\} >$. It is easy to see that P'' is a normal cover of P. By Algorithm 2.13 and Algorithm 2.3 we can compute P'' in polynomial time in the size of P. Our proof is complete.

3 Relation schemes in BCNF

In this section we give some new results concerning relation schemes in BCNF. We show some cases in which FD-relation equivalence problem is solved by polynomial time algorithms. Now we give some necessary definitions.

Definition 3.1 *Let $R = \{h_1, \ldots, h_m\}$ be a relation over U, and $A, B \subseteq U$.*

Then we say that B functionally depends on A in R denoted $(A \xrightarrow[R]{f} B)$ iff

$$(\forall h_i, h_j \in R)(\forall a \in A)(h_i(a) = h_j(a)) \Longrightarrow (\forall b \in B)(h_i(b) = h_j(b)))$$

Let $F_R = \{(A, B) : A, B \subseteq U,\ A \xrightarrow[R]{f} B\}$. F_R is called the full family of functional dependencies of R. Where we write (A, B) or $A \to B$ for $A \xrightarrow[R]{f} B$ when R, f are clear from the context.

A functional dependency over U is a statement of the form $A \to B$,where $A, B \subseteq U$. The FD $A \to B$ holds in a relation R if $A \underset{R}{\overset{f}{\cup}} B$. We also say that R satisfies the FD $A \to B$.

It is easy to see that F_R satisfies the following properties:

$\forall B \subseteq A$: $A \to B \in F_R$ (pseudoreflexivity), if $A \to B \in F_R$ and $C \subseteq D$, then $\{A \cup D\} \to \{B \cup C\}$ (augmentation), if $A \to B \in F_R$ and $\{B \cup C\} \to D$, then $\{A \cup C\} \to D$(pseudotransitivity).

Definition 3.2 *A relation scheme S,or RS for short, is a pair $< U, F >$. Where U is a set of attributes, and F is a set of FDs over U. Let F^+ be a set of all FDs that can be derived from F by the above rules. Denote $A^+ = \{a: A \to \{a\} \in F^+\}$. A^+ is called the closure of A over S.Denote $Z(F^+) = \{A \subseteq U: A^+ = A\}$.*

Clearly, in [1] if $S =< U, F >$ is a RS, then there is a relation R over U such that $F_R = F^+$. Such a relation is called an Armstrong relation of S.

Let R be a relation, $S =< U, F >$ be a RS, and $A \subseteq U$. Then A is a key of R (a key of S, *respectively*) if $A \xrightarrow[R]{f} U$ ($A \to U \in F^+$, *respectively*). A is a minimal key of R(S, *respectively*) if A is a key of R(S, *respectively*), and any proper subset of A is not a key of R(S, *respectively*). Denote K_R(K_S, *respectively*) the set of all minimal keys of R(S, *respectively*).

Clearly, K_R, K_S are Sperner systems over U.

Let R be a relation,$S =< U, F >$ be a RS.R, S are in Boyce-Codd normal form (BCNF) if for each $A \to \{a\} \in F^+$($\in F_R$, *respectively*) and $a \notin A$ then $A \to U \in F^+$($\in F_R$, *respectively*).

Definition 3.3 *Let $S =< U, F >$ be a RS. We say that S is a k-RS over U if $F = \{K_1 \to U, \ldots, K_m \to U\}$, where $\{K_1, \ldots, K_m\}$ is a Sperner system over U. It is easy to see that $K_S = \{K_1, \ldots, K_m\}$.*

It can be seen that a relation scheme $S =< U, F >$ is in BCNF iff $\forall A \subseteq U$ either $A^+ = A$ or $A^+ = U$. Clearly, if $S =< U, F >$ is in BCNF then using the algorithm for finding a minimal cover we can construct in polynomial time a k-RS $S' =< U, F' >$ such that $F^+ = {F'}^+$, see [10]. Conversely, it can be seen that an arbitrary k-RS is in BCNF. Consequently, we can consider a RS in BCNF as a k-RS.

Theorem 3.4 [4] *Let $S_1 =< U, F_1 >, S_2 =< U, F_2 >$ be two RS over U. Then ${F_1}^+ = {F_2}^+$ iff $Z({F_1}^+) = Z({F_2}^+)$, and ${F_1}^+ \subseteq {F_2}^+$ iff $Z({F_2}^+) \subseteq Z({F_1}^+)$.*

Theorem 3.5 [4] *Let K be a Sperner system and $S =< U, F >$ be a RS over U. Then $K_S = K$ iff*

$$\{U\} \cup K^{-1} \subseteq Z(F^+) \subseteq \{U\} \cup G(K^{-1}),$$

where $G(K^{-1}) = \{A \subseteq U : \exists B \in K^{-1}: A \subseteq B\}$.

Based on Theorem 3.5 we have

Theorem 3.6 *Let $K = \{K_1, \dots, K_t\}$ be a Sperner system over U. Consider the relation scheme $S = (U, F)$ with $F = \{K_1 \to U, \dots, K_t \to U\}$.*

Then $K_S = K$, and $Z(F^+) = G(K_S^{-1}) \cup \{U\}$.

Let R be a relation over U. Denote $A_R^+ = \{a \in U: A \to \{a\} \in F_R\}$, and $Z(F_R) = \{A \subseteq U: A_R^+ = A\}$.

According to Theorem 3.5 we can give examples for which there are two RSs $S_1 =< U, F_1 >, S_2 =< U, F_2 >$ such that $K_{S_1} = K_{S_2}$, but ${F_1}^+ \neq {F_2}^+$. Clearly, for relations this consider is the same.

We give the following notion.

Definition 3.7 *Let $S =< U, F >$ be a RS, R be a relation over U. We call S (R,respectively) is an unique RS (relation,respectively) if for all RS $S' =< U, F' >$ (relation R',respectively) : $K_S = K_{S'}$ ($K_R = K_{R'}$,respectively) then $F^+ = F'^+$ ($F_R = F_{R'}$,respectively).*

Proposition 3.8 *The time complexity of deciding whether a given relation R over U is unique is polynomial in the sizes of R and U.*

Proof: Let R a relation over U. By [13] from R we can compute ${K_R}^{-1}$ in polynomial time in the sizes of R and U, where K_R is a set of all minimal keys of R.

Denote elements of ${K_R}^{-1}$ by $A_1, \dots, A_t$. Set $M_R = \{A_i - a: a \in U, i = 1, \dots, t\}$.

Denote elements of M_R by $B_1, \dots, B_s$. We construct a relation $R' = \{h_0, h_1, \dots, h_s\}$ as follows:

For all $a \in U$, $h_0(a) = 0$, for each $i = 1, \dots, s$ $h_i(a) = 0$ if $a \in B_i$, in the converse case we set $h_i(a) = i$.

By [10] R' is in BCNF and $K_R = K_{R'}$.

We construct a relation $R'' = \{l_0, l_1, \dots, l_t\}$ as follows:

$l_0(a) = 0$ for all $a \in U$. For all $j = 1, \dots, t$ then $l_j(a) = j$ if $a \notin A_j$,

in the converse case set $l_j(a) = 0$.

It can be seen that $K_R = K_{R''}$ and $Z(F_{R''}) = (K_R^{-1})^+$. (see Definition 1.5).

It is easy to see that M_R, R'' and R' are constructed in polynomial time in the sizes of U and R.

Based on Theorem 3.5 we see that R is unique iff $F_{R'} = F_{R''}$. Clearly, $F_{R'} = F_{R''}$ can be tested in polynomial time in the sizes of R' and R''. The proposition is proved.

Definition 3.9 [4] *Let K be a Sperner system over U. We say that K is saturated if for any $A \notin K$, $\{A\} \cup K$ is not a Sperner system.*

Theorem 3.10 [4] *Let $S = < U, F >$ be a RS. If K_S is a saturated Sperner system, then S is an unique RS.*

Examples show that there is a Sperner system K (K^{-1},respectively) such that K (K^{-1}, respectively) is saturated, but K^{-1} (K,respectively) is not saturated.
Now we define the next notion.

Definition 3.11 *Let K be a Sperner system over U. We say that K is inclusive, if for every $A \in K$ there is a $B \in K^{-1}$ such that $B \subset A$. We call K is embedded if for each $A \in K$ there exists a $B \in H: A \subset B$, where $H^{-1} = K$.*

Theorem 3.12 [13] *Let K be a Sperner system over U. Denote H a Sperner system for which $H^{-1} = K$. The following facts are equivalent:*

(1) K is saturated,

(2) K^{-1} is embedded,

(3) H is inclusive.

Let $S = < U, F >$ be a RS in BCNF,R be a relation in BCNF. Then we say that S is an inclusive RS if K_S is inclusive and R an embedded relation if K_R^{-1} is embedded.
It can be seen that the BCNF property of S is polynomially recognizable. By [13] we can compute K_R^{-1} in polynomial time in the size of R, and based on polynomial time algorithm finding minimal cover we also construct K_S from a given BCNF relation scheme. On the other hand, by definitions of embedded,inclusive Sperner systems we obtain the following proposition.

Proposition 3.13 *Let $S = < U, F >$ be a RS ,R be a relation over U. Then*

1. *Deciding whether S is an inclusive RS is solved in polynomial time in the size of S.*

2. *There exists an algorithm deciding whether R is an embedded relation and the time complexity of it is polynomial in the sizes of U and R.*

It is easy to see that if $S = < U, F >, S' = < U, F' >$ are two RSs then deciding whether $F^+ = F'^+$ can be tested in polynomial time in the sizes of S and S'.

Now we introduct the next problem.
Let $S = < U, F >, S' = < U, F' >$ be two RSs. Decide whether $K_S = K_{S'}$.

The following proposition is clear.

Proposition 3.14 *Let S, S' be two RSs.If S is unique then deciding whether $K_S = K_{S'}$ is polynomially recognizable.*

In [10] the FD-relation equivalence problem is introduced as follows:
Let $S = < U, F >$ be a RS, R be a relation over U. Decide whether $F^+ = F_R$, i.e. R is an Armstrong relation of S.

Definition 3.15 *Let K_1, K_2 be two Sperner system over U. We set $K = K_1 \cup K_2$ and $T_K = \{A \in K: \nexists B \in K: A \subset B\}$. We say that the union $K = K_1 \cup K_2$ is equality if $\forall A_1, A_2 \in T_K: |A_1| = |A_2|$.*

Based on Definition 3.15 we give the next theorem related to the FD-relation equivalence problem .

Proposition 3.16 *Let $S = < U, F >$ be a relation scheme in BCNF and R a relation over U in BCNF. $K_S = \{A_1, \ldots, A_p\}$ ($K_R^{-1} = \{B_1, \ldots, B_q\}$) is the set of minimal keys of S (the set of antikeys of R). Then if $K_S \cup K_R^{-1}$ is equality then the FD-relation equivalence problem is solved in polynomial time in the sizes of S and R.*

Proof: Clearly, by [13] from R we compute K_R^{-1} in polynomial time in the size of R, and from S we find a k-relation scheme that is a minimum cover of S. The minimum cover is constructed in polynomial time in the size of S. We set $K = K_S \cup K_R^{-1}$. Because K is equality,we assume that $|A| = m$, and $|U| = n$. We compute the number C_n^m. Clearly, K and K^{-1} are uniquely determined by each other. By definitions of K_S and K_R^{-1} we can see that if $|T_K| \neq C_n^m$ then $K_S \neq K_R$. Thus, in BCNF class we obtain $F^+ \neq F_R$.

Now we assume that $|T_K| = C_n^m$. If there is $A_i (1 \leq i \leq p)$ such that $A_i \subseteq B_j (1 \leq j \leq q)$ then $K_S \neq K_R$. Consequently,we can assume that $A_i \not\subseteq B_j$ for all i, j. For each $j = 1, \ldots, q$ we compute B_j^+. It can be seen that for all $D \subseteq U$ D^+ is computed in polynomial time in the size of S. We set $M = \{B_j \cup \{a\}: a \in U - B_j\} = \{M_1, \ldots, M_t\}$. It is obvious that M is computed in polynomial time. If $B_j^+ \neq U$ and for all $l = 1, \ldots, t$ $M_l^+ = U$ hold then $B_j \in K_S^{-1}$ holds, otherwise we obtain $B_j \notin K_S^{-1}$. If there is a B_j: $B_j \notin K_S^{-1}$ then by the definition of antikeys $K_R \neq K_S$.We assume that for all j=1,...,q $B_j \in K_S^{-1}$. For each $i = 1, \ldots, p$ we set $N = \{A_i - \{a\}: a \in A_i\} = \{N_1, \ldots, N_s\}$. It can be seen that N is computed in polynomial time. If there is a $N_n (1 \leq n \leq s)$ such that $N_n \not\subseteq B_j$ for all $j = 1, \ldots, q$ then $A_i \notin K_R$ holds. In the converse case we obtain $A_i \in K_R$. Clearly, if there is an $A_i \notin K_R$ then $K_S \neq K_R$. We assume that for each $i = 1, \ldots, p$ we have $A_i \in K_R$. We set

$$Z = \{A_i - \{a\}: a \in A_i, i = 1, \ldots, p\},$$

$$W = \{A \in Q: A = A^+, (A \cup \{a\})^+ = U, \forall a \in U - A\},$$

$$J = \{B_j \cup \{a\}: a \in U - B_j, j = 1, \ldots, q\},$$

$$I = \{B \in J: B_R^+ = U, \{B - a\}_R^+ \neq U \forall a \in B\}.$$

Based on definition of K_S and definition of K_R^{-1} we can see that if either there is an $A \in W$ such that $A \notin K_R^{-1}$ or there exists a $B \in I$ but $B \notin K_S$ then $K_S \neq K_R$. It can be seen that W, I are constructed in polynomial time in the sizes of S, R, K_S, K_R^{-1}. Finally, we see that if for all $i = 1, \ldots, p, j = 1, \ldots, q$ $A_i \in K_R, B_j \in K_S^{-1}, W \subseteq K_R^{-1}, I \subseteq K_S$ hold then by $|T_K| = C_n^m$ and according to definition of set of minimal keys and definition of set of antikeys we obtain $K_R = K_S$. Since S, R are in BCNF we have $F_R = F^+$. The proof is complete.

Let K be a Sperner system over U. We say that K is pseudo-monotonous if for each Sperner system K' : $K \cap K' = \emptyset$ and $K \cup K'$ is a Sperner system over U then $K^{-1} \subseteq \{K \cup K'\}^{-1}$.

We say that K is a changed Sperner system if for each H' : $H' \subset H$ then there are $A \in K, B \in H'^{-1}$ such that $A \subset B$, where $H^{-1} = K$.

Proposition 3.17 *Let S be a RS in BCNF, R be a relation in BCNF. Then if either K_S is pseudo-monotonous or K_R^{-1} is changed, then FD-relation equivalence problem is solved in polynomial time in the sizez of S and R.*

Proof: First we assume that K_R^{-1} is a changed Sperner system. Based on a polynomial time algorithm finding a minimal cover,we construct a set of all minimal keys K_S. It is known [13] that from R we compute K_R^{-1} in polynomial time in the size of R.

If there are $A \in K_S$ and $B \in K_R^{-1}$ such that $A \subseteq B$, then $K_S \neq K_R$. Thus,for all $A \in K_S, B \in K_R^{-1}$ we can assume that $A \not\subseteq B$. We set $X = \{A - \{a\}: A \in K_S, a \in A\}$. If for all $C \in X, B \in K_R^{-1}$ we obtain $C \subseteq B$ then $K_S \subseteq K_R$. In the converse case we have $K_S \neq K_R$. It is easy to see that X is computed in polynomial time. We assume that $K_S \subseteq K_R$.

For each $B \in K_R^{-1}$ we compute B^+. If there is a B such that $B^+ = U$ then $K_S \neq K_R$. We assume that $B^+ \neq U$ for all $B \in K_R^{-1}$. We set $Y = \{B \cup \{a\}: B \in K_R^{-1}, a \in U - B\}$. It is obvious that Y is computed in polynomial time. If for all $D \in Y$ we have $D^+ = U$ then $K_R^{-1} \subseteq K_S^{-1}$. In the converse case we obtain $K_R^{-1} \neq K_S^{-1}$. Because K and K^{-1} are uniquely determined by each other, we have $K_R \neq K_S$. Now assume that $K_R^{-1} \subseteq K_S^{-1}$ and $K_S \subseteq K_R$. By hypothesis K_R^{-1} is a changed Sperner system. Consequently,if $K_S \subset K_R$ then there are $B \in K_R^{-1}$ and $E \in K_S^{-1}$ such that $B \subset E$. Hence, $K_R^{-1} \not\subseteq K_S^{-1}$ holds.Thus, $K_S = K_R$. Because S, R are in BCNF, we obtain $F_R = F^+$.

If S is pseudo-monotonous then the proof is the same. The proof is complete.

4 Conclusion

Our further research will be devoted to the following problems:

1. What is the time complexity of finding a dual scheme P from a given relation R such that $D^+ = D_R$
2. Given a relation scheme S and a relation R. What is the time complexity of deciding whether $K_S = K_R$.
3. Let S_1, S_2 be two relation schemes over U. What is the time complexity of deciding whether $K_{S_1} = K_{S_2}$.
4. Let S be a RS. What is the time complexity of deciding whether S is an unique RS.

Acknowledgments: The authors are grateful to Dr. Uhrin Bela for useful comments to the first version of the manuscript.

References

[1] Armstrong W.W. Dependency Structures of Database Relationships. Information Processing 74, Holland publ. Co. (1974) pp. 580-583.

[2] Beeri C., Bernstein P.A. Computational problems related to the design of normal form relational schemas.ACM Trans. on Database syst.4,1,(1979)pp.30-59.

[3] Berman J., Blok W. Positive Boolean Dependencies. Infor.Proces. Letters 27 (1988) pp. 147-150.

[4] Burosch G., Demetrovics J., Katona G.O.H. The poset of closures as a model of changing databases, Order 4 (1987) pp. 127-142.

[5] Czédli G. d-dependency structures in the relational model of data. Acta Cybernetica Hungary V/I (1980) pp.49-57.

[6] Czédli G. On Dependencies in the relational model of data.EIK 17 (1981) 2/3 pp.103-112.

[7] Demetrovics J. Relacios adatmodell logikai es strukturalis vizsgalata MTA-SZTAKI Tanulmanyok,Budapest, 114 (1980) pp.1-97.

[8] Demetrovics J.,Gyepesi G. On the functional dependency and some generalizations of it. Acta Cybernetica Hungary, Tom. 5, Fasc. 3 (1981), pp. 295-305.

[9] Demetrovics J.,Gyepesi G. Some generalized type functional dependencies formalized as equality set in matrices.Discrete Appl.Math. 6 (1983) pp.35-47.

[10] Gottlob G. , Libkin L. Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. Acta Cybernetica Hungary Tom.9 Fasc.4 (1990) pp. 385-402.

[11] V.D.Thi. Logical dependencies and irredundant relations. Computers and Artificial Intelligence 7 (1988), pp.165-184.

[12] Thi V.D. Funkcionalis fuggoseggel kapcsolatos nehany kombinatorikai jellegu vizsgalat a relacios adatmodellben. MTA-SZTAKI Tanulmanyok, Budapest, 191 (1986) pp. 1-157. Ph.D. Dissertation.

[13] Thi V.D. Minimal keys and Antikeys.Acta Cybernetica Hungary Tom.7 Fasc.4 (1986) pp.361-371.

Received October 21, 1992

Acta Cybernetica, Vol. 11, No. 1-2, Szeged, 1993

Fundamental Concepts of Object Oriented Databases

K.-D. Schewe* B. Thalheim*

Abstract

It is claimed that object oriented databases (OODBs) overcome many of the limitations of the relational model. However, the formal foundation of OODB concepts is still an open problem. Even worse, for relational databases a commonly accepted datamodel existed very early on whereas for OODBs the unification of concepts is missing. The work reported in this paper contains the results of our first investigations on a formally founded object oriented datamodel (OODM) and is intended to contribute to the development of a uniform mathematical theory of OODBs.

A clear distinction between *objects* and *values* turns out to be essential in the OODM. *Types* and *Classes* are used to structure values and objects repectively. Then the problem of unique object identification occurs. We show that this problem can be be solved for classes with extents that are completely representable by values. Such classes are called *value-representable*.

Another advantage of the relational approach is the existence of structurally determined *generic update operations*. We show that this property can be carried over to object-oriented datamodels if classes are value-representable. Moreover, in this case *database consistency* with respect to implicitly specified referential and inclusion constraints will be automatically preserved.

This result can be generalized with respect to distinguished classes of explicitly stated static constraints. Given some arbitrary method and some integrity constraint there exists a *greatest consistent specialization* (GCS) that behaves nice in that it is compatible with the conjunction of constraints. We present an algorithm for the GCS construction of user-defined methods and describe the GCSs of generic update operations that are required herein.

1 Introduction

The shortcomings of the relational database approach encouraged much research aimed at achieving more appropriate data models. It has been claimed that the *object-oriented approach* will be the key technology for future database systems and languages [8]. Several systems [4,6,7,9,15,16,17,19,26,36,37,38] arose from these

*Cottbus Technical University, Computer Science Institute, P.O.Box 101344, D-03013 Cottbus

efforts. However, in contrast to research in the relational area there is no common formal agreement on what constitutes an object-oriented database [10,11,13].

The basic question "What is an object?" seems to be trivial, but already here the variety of answers is large. In object oriented programming the notion of an *object* was intended as a generalization of the abstract data type concept with the additional feature of *inheritance*. In this sense object orientation involves the isolation of data in semi-independent modules in order to promote high software development productivity. The development of object oriented databases regarded an object also as a basic unit of persistent data, a view that is heavily influenced by existing semantic datamodels (SDMs) [2,29,31,39,40,60]. Thus, object oriented databases are composed of independent objects but must also provide for the maintenance of inter-object consistency, a demand that is to some degree in dissonance with the basic style of object orientation.

A view that is common in OODB research is that objects are abstractions of real world entities and should have an identity [8]. This leads to a distinction between *values* and *objects* [10,11]. A value is identified by itself whereas an object has an identity independent of its value. This object identity is usually encoded by object identifiers [1,3,34]. Abstracting from the pure physical level the identifier of an object can be regarded as being immutable during the object's lifetime. Identifiers ease the sharing and update of data. However, such abstract identifiers do not relieve us from the task to provide unique identification mechanisms for objects. In object oriented programming object names are sufficient, but retrieving mass data by name is senseless.

In most approaches to OODBs an object is coupled with a value of some fixed structure. To our point of view this contradicts already the goal of objects being abstractions of reality. In real situations an object has several and also changing aspects that should be captured by the object model. Therefore, in our object model each *object* o consists of a unique identifier id, a set of (type-, value-)pairs (T_i, v_i), a set of (reference-, object-)pairs (ref_j, o_j) and a set of methods $meth_k$.

Types are used to structure values. Classes serve as structuring primitive for objects having the same structure and behaviour. It is obvious that the multiple aspects view of an object allows them to be simultaneously members of more than one class and to change class memberships. This setting also makes every discussion on "object migration" unnessecary, as migration is only a specific form of value change.

In our model a class structure uniformly combines aspects of object values and references. The extent of classes varies over time, whereas types are immutable. Relationships between classes are represented by references together with referential constraints on the object identifiers involved. Moreover, each class is accompanied by a collection of methods. A schema is given by a collection of class definitions together with explicit integrity constraints.

The Identification Problem. One important concept of object-oriented databases is *object identity*. Following [1,12] the immutable identity of an object can be encoded by the concept of abstract object-identifiers. The advantages of this approach are that sharing, mutability of values and cyclic structures can be represented easily [42]. On the other hand, object identifiers do not have a meaning for the user and should therefore be hidden.

We study whether equality of identifiers can be derived from the equality of values. In the literature the notion of "deep" equality has been introduced for objects with equal values and references to objects that are also "deeply" equal. This recursive definition becomes interesting in the case of cyclic references.

Therefore, we introduce ***uniqueness constraints***, which express equality on identifiers as a consequence of the equality of some values or references. On this basis we can address the problem how to characterize those classes that are completely representable (and hence also identifiable) by values.

Generic Update Operations. The success of the relational data model is due certainly to the existence of simple query and update-languages. Preserving the advantages of the relational in OODBs is a serious goal.

The generic querying of objects has been approached in [1,12]. While querying is per se a set-oriented operation, i.e. it is not necessary to select just one single object, and hence does not raise any specific problems with object identifiers, things change completely in case of updates. If an object with a given value is to be updated (or deleted), this is only defined unambigously, if there does not exist another object with the same value. If more than one object exists with the same value or more generally with the same value and the same references to other objects, then the user has to decide, whether an update- or delete-operation is applied to *all* these objects, to only *one* of these objects selected non-deterministically or to *none* of them, i.e. to reject the operation. However, it is not possible to specify a priori such an operation that works in the same way for all objects in all situations. The same applies to insert-operations. Hence the problem, in which cases operations for the insertion, deletion and update of objects can be defined generically.

Some authors [43] have chosen the solution to abandon generic operations. Others [6,7,9] use identifying values to represent object identity, thus embody a strict concept of surrogate keys to avoid the problem. Our approach is different from both solutions in that we use the concept of hidden abstract identifiers, but at the same time formally characterize those classes for which unique generic operations for the insertion, deletion and update of single objects can be derived automatically. It turns out that these are exactly the value-representable ones.

The Consistency Problem. One of the primary benefits that database systems offer is automatic enforcement of database integrity. One type of integrity is maintained through automatic concurrency control and recovery mechanisms; another one is the automatic enforcement of user-specified integrity constraints. Most commercial database systems, especially relational database management systems enforce only a bare minimum of constraints, largely because of the performance overhead associated with updates.

The *maintenance problem* is the problem how to ensure that the database satisfies its constraints after certain actions. There are at present two approaches to this maintenance problem. The first one, more classical is the modification of methods in accordance to the specified integrity constaints. The second approach uses generation mechanisms for the specified events. Upon occurrence of certain database events like update operations the management component is activated for integrity maintenance. The first research direction did not succeed because of some limitations within the approach. The second one is at present one of the most active database research areas. One of our objectives is to show that the first approach can be extended to object-oriented databases using stronger mathematical fundamentals.

Accuracy is an obviously important and desirable feature of any database. To this end, *integrity constraints*, conditions that data must satisfy before a database is updated, are commonly employed as a means of helping to maintain consistency. In relational databases the specification and enforcement of integrity constraints has a long tradition [61], whereas in OODBs the integrity problem has only recently drawn attention [48].

In object oriented databases, integrity maintenance can be based on two different approaches. The first one uses blind update operations. In this case, any update is allowed and the system organizes the maintenance. The second approach is based on methods rewriting. This approach is more effective. Assuming a consistent database state the modified method can not lead to an inconsistent state.

In relational databases distinguished classes of static integrity constraints have been discussed such as *inclusion*, *exclusion*, *functional*, *key* and *multi-valued dependencies*. All these constraints can be generalized to the object oriented case. Then the result on the existence of integrity preserving methods can be generalized to capture also these constraints. We shall also describe the resulting methods.

The Organization of the Paper. We start with a motivating example in Section 2 then introduce in Section 3 a core OODM to formalize the concepts used intuitively in the example. In Section 4 the notions of (weak) value-representability are introduced in order to handle the identification problem. The genericity problem will be approached in Section 5. We show the relationship between value-representability and the unique existence of generic update operations. The consistency problem is dealt with in Section 6. We outline an operational approach based on the computation of greatest consistent specializations (GCSs). Since the used algorithm allows the problem to be reduced to basic update operations, we describe the GCSs hereof. We summarize our results and describe some open problems in Section 7.

2 A Motivating Example

In this section we start giving a completely informal introduction to the OODM on the basis of a simple university example. We first introduce *types* and *classes*, then show an example of a *database instance*, i.e. the content of the database at a given timepoint. The representation of an instance requires *object identifiers*. Then we extend the example by introducing user-defined *constraints*. We shall see that this enables alternative representations without using identifiers, hence leads to the notion of *value-representability*. Finally, we indicate the definition of methods as a means to model database dynamics. For the sake of simplicity we only describe a *generic update method* that can be generated by the system.

As already said in the introduction, we distinguish between values and objects with the main difference defined by values identifying themselves whereas objects require an additional external identification mechanism. Types are used to structure values. Thus, let us first give some examples of types.

Example Basically, every type can be built from a few predefined *basic types* such as $BOOL$, NAT, $STRING$, etc. and also predefined *type constructors* for records, finite sets, lists, unions, etc.

The type definition for $PERSONNAME$ uses both a set constructor $\{\cdot\}$ and a (tagged) record constructor $(\cdot)$:

Type $PERSONNAME$
  = (FirstName : $STRING$,
    SecondName : $STRING$,
    Titles : $STRING$)
End $PERSONNAME$

The definition of a type *PERSON* uses the type *PERSONNAME*.

```
Type PERSON
   = ( PersonIdentityNo : NAT ,
     Name : PERSONNAME )
End PERSON
```

The following defines *STUDENT* as a subtype of *PERSON*, i.e. we can naturally project each value of type *STUDENT* onto a value of type *PERSON*.

```
Type STUDENT
   = ( PersonIdentityNo : NAT ,
     StudNo : NAT ,
     Name : PERSONNAME )

End STUDENT
```

Besides these definitions of types as sets of values we may also define new type constructors as follows, where α is a parameter for this new constructor:

```
Type MPERSON(α)
   = ( PersonIdentityNo : NAT ,
     Spouse : α )
End MPERSON
```

□

Next we use these types to build the structural part of an OODM schema. We define a schema as a collection of classes and a class as a variable collection of objects.

Example Each object in a class has a structure, which combines aspects of values associated with the object and references to other objects. This structure can be based on a type definition as above or involve itself a (nameless) type definition. Moreover, class definitions involve IsA relations in order to model objects in more than one class. We use $\circ$ to indicate concatenation for record types.

```
Schema University
   Class PERSONC
      Structure PERSON
   End PERSONC
   Class MARRIEDPERSONC
      IsA PERSONC
      Structure ( PersonIdentityNo : NAT ,
         Spouse : MARRIEDPERSONC )
   End MARRIEDPERSONC
   Class STUDENTC
      IsA PERSONC
      Structure STUDENT ∘
         ( Supervisor : PROFESSORC ,
         Major : DEPARTMENTC ,
         Minor : DEPARTMENTC )     End STUDENTC
   Class PROFESSORC
      IsA PERSONC
```

```
    Structure ( PersonIdentityNo : NAT ,
        Age : NAT ,
        Salary : NAT ,
        Faculty : DEPARTMENTC )    End PROFESSORC
Class DEPARTMENTC
    IsA PERSONC
    Structure ( DeptName : STRING )
End DEPARTMENTC
```
□

In principle, we are now able to describe the content of the database at a given timepoint. For such database instances we need a type *ID* of object identifiers that is used for two purposes, first as a unique and efficient internal identification mechanism for objects and second for modelling objects in different classes and references to other objects. In this case each class will be associated with a *representation type* that can be used directly for storing objects.

Example We use $\mathcal{D}$ as a name for the instance.

$\mathcal{D}$(PERSONC) =
{ (i_1 , (123 , ("John" , "Denver" , { "Professor" , "Dr" }))) ,
(i_2 , (124 , ("Mary" , "Stuart" , { "Dr" }))) ,
(i_3 , (456 , ("John" , "Stuart" , { }))) ,
(i_4 , (567 , ("Laura" , "James" , { }))) ,
(i_5 , (987 , ("Dave" , "Ford" , { }))) }

$\mathcal{D}$(MARRIEDPERSONC) =
{ (i_1 , (123 , i_2)) ,
(i_2 , (124 , i_1)) }

$\mathcal{D}$(PROFESSORC) =
{ (i_1 , (123, 48,8000, i_6)) }

$\mathcal{D}$(STUDENTC) =
{ (i_3 , (456 , 1023 , ("John" , "Stuart" , { }) , i_1 , i_6 , i_7)) ,
(i_4 , (567 , 2134 , ("Laura" , "James" , { }) , i_1 , i_6 , i_7)) }

$\mathcal{D}$(DEPARTMENTC) =
{ (i_6 , ("Computer Science")) ,
(i_7 , ("Philosophy")) ,
(i_8 , ("Music")) }

□

Note that the following three conditions are satisfied by the instance:

- The object identifiers are unique within a class,
- the IsA relations in the schema give rise to set inclusion relationships for the underlying sets of identifiers (inclusion integrity), and
- the identifiers occurring within an object's value at a place corresponding to a reference, always occur as an object identifier in the referenced class (referential integrity).

We shall always refer to these conditions as model inherent constraints that must be satisfied by each instance. Other integrity constraints can be defined by the user and added to the schema in order to capture more application semantics as shown in the next example.

Example First let us express that there are no two persons with the same PersonIdentityNo, no two students with the same StudentNo and no two departments with the same name. In order to formulate this, use x_P, x_S and x_D to refer to the content of the classes PERSONC, STUDENTC and DEPARTMENTC, and let $c_P : PERSON \rightarrow (\text{PersonIdentityNo} : NAT)$ and $c_S : STUDENT \times ID^3 \rightarrow (\text{StudNo} : NAT)$ be functions that arise from the natural projection to the components PersonIdentityNo and StudNo in $PERSON$ and $STUDENT$ respectively. This gives the following ***uniqueness constraints.***

$$\begin{array}{l} \forall i, j :: ID. \forall v, w :: \\ PERSON.\ (i, v) \in x_P \wedge (j, w) \in x_P \wedge c_P(v) = c_P(w) \ \Rightarrow \ i = j. \\ \forall i, j :: ID. \forall v, w :: \\ STUDENT \times ID^3.\ (i, v) \in x_S \wedge (j, w) \in x_S \wedge c_S(v) = c_S(w) \ \Rightarrow \ i = j \\ \forall i, j :: ID. \forall v, w :: \\ (\text{DeptName} : STRING).\ (i, v) \in x_D \wedge (j, w) \in x_D \wedge v = w \ \Rightarrow \ i = j \ . \quad (1) \end{array}$$

Let us further assume that the salary of a professor is determined by his/her age. For this purpose, let Age, Salary : $T_{Prof} \rightarrow NAT$ be the natural projections to the Age- and Salary-values respectively. Then we have the following *functional constraint* on the class PROFESSORC:

$$\forall i, j :: ID. \forall v, w :: T_{Prof}.\ (i, v) \in x_{Prof} \wedge (j, w) \in x_{Prof} \wedge \text{Age}(v) = \text{Age}(w) \ \Rightarrow \\ \text{Salary}(v) = \text{Salary}(w) \ . \quad (2)$$

Next assume that we want to guarantee that the spouse of a person's spouse is the person itself, which gives (with the abbreviations understood) the formula

$$\begin{array}{l} \forall i, j :: ID. \forall v, w :: \\ T_{MP}.\ (i, v) \in x_{MP} \wedge (j, w) \in x_{MP} \wedge \text{Spouse}(v) = j \ \Rightarrow \ \text{Spouse}(w) = i \ . (3) \end{array}$$

Note that all these constraints are also satisfied by the instance above. □

Now we have added uniqueness constraints, the object identifiers used in instances correspond one-to-one to values of some types associated with the classes. These are the so-called value identification types V_C. Hence we could remove identifiers and represent the same information in a purely value-based fashion. In our example the value representation type for the class PERSONC is simply $PERSON$, but for the class MARRIEDPERSONC we need the recursive type

$$V_{MP} \ = \ PERSON \circ (\text{Spouse} : V_{MP})$$

with values that are rational trees [45,47].

So far only structural aspects (types, classes, constraints) have been considered. Let us now add methods to classes in order to model the dynamics of the database. In the OODM methods will be modelled in a simple procedural style.

Example Let us describe an insert-method for the class PERSONC.

```
insert_PersonC (in: P :: PERSON, out: I :: ID) =
   IF ∃ O ∈ PERSONC . value(O) = P
   THEN I := ident(O)
   ELSE I := NewId ;
      PERSONC := PERSONC ∪ { ( I,P)}
   ENDIF
```

For an insertion into the class MARRIEDPERSONC we need a more complex input type V recursively defined as

$$V = PERSON \circ (V \cup ID)$$

For each $P :: V$ let $f(P) :: PERSON$ be the projection onto $PERSON$ corresponding to the subtype relation between V and $PERSON$. Then we have

```
insert_MarriedPersonC (in: P :: V, out: I :: ID) =
   I := insert_PersonC(f(P)) ;
   IF ∀ O ∈ MARRIEDPERSONC . ident(O) ≠ I
   THEN P' := substitute(I,P,Spouse(P)) ;
         IF P' :: ID
         THEN J := P'
         ELSE J := insert_MarriedPersonC(P')
         ENDIF ;
      MARRIEDPERSONC := MARRIEDPERSONC ∪ { (I,f(P) ∘ (J))}
   ENDIF
```

We used the global method NewId to denote the selection of a new identifier. The expression substitute(I,P,T) denotes the result of replacing the value I for P in the expression T. Later we shall use a more abstract syntax oriented toward guarded commands [20,41,46]. □

Later we shall see that methods as described in this example are canonical and can be automatically derived from the schema. Corresponding generic update methods look quite similar with the only difference that there is no output. Such generic update methods only exist for value representable classes in which case, however, they enforce integrity with respect to the model inherent constraints. However, generic update methods need not be consistent with respect to the user-defined constraints. To achieve this, we have to apply the GCS algorithm to user-defined methods.

In the following sections we formally define the concepts above and proof the main results on value representation, generic updates and integrity enforcement.

3 A Core Object Oriented Datamodel

In this section we present a slightly modified version of the object oriented datamodel (OODM) of [45,47,49]. We observe that an object in the real world always has an identity. Therefore, abstract (i.e. system-provided) object identifiers are introduced to capture identity. However, neither the real world object that was the basis of the abstraction nor the abstract identifier can be used for the identification of an object.

In contrast to existing object oriented datamodels [1,3,4,6,7,8,9,16,17,26,36,37, 42,43,54] an object is not coupled with a unique type. In contrast, we observe that real world objects can have different aspects that may change over time. Therefore, a primary decision was taken to let an object be associated with more than one type and to let these types even change during the object's lifetime. The same applies to references to other objects.

In the following let N_P, N_T, N_C, N_R, N_F, N_M and V denote arbitrary pairwise disjoint, denumerable sets representing parameter-, type-, class-, reference-, function-, method- and variable-names respectively.

3.1 A Simple Type System

Relational approaches to data modelling are called value-oriented since in these models real world entities are completely represented by their values. In the object-oriented approach we distinguish between objects and values. Values can be gouped into types. In general, a type may be regarded as an immutable set of values of a uniform structure together with operations defined on such values. Subtyping is used to relate values in different types.

In [12,47,49] algebraic type specifications as in [21,23] have been used to allow open type systems. For the sake of simplicity we deviate here from this approach and follow the more classical view of [14,15,45] using a type system that consists of some *basic types* such as $BOOL$, $NATURAL$, $INTEGER$, $STRING$, etc., and *type constructors* for records, finite sets, bags, lists, etc. and a *subtyping* relation. Moreover, assume the existence of *recursive types*, i.e. types defined by (a system of) domain equations. In principle we could use one of the type systems defined in [4,5,14,15,19,24,38]. In addition we suppose the existence of an abstract identifier type ID in $\mathcal{T}$ without any non-trivial supertype. Arbitrary *types* can then be defined by nesting. A type T without occurrence of ID will be called a *value-type*. We shall proceed giving a more formal definition of types.

Definition 1 *1. A* base type *is either* $BOOL$, NAT, INT, $FLOAT$, $STRING$, ID *or* $\perp$.

2. Let $a_i \in N_F$ *and* $\alpha, \beta, \alpha_i \in N_P$ $(i = 1, \dots, n)$. *A* type constructor *is either* $(a_1 : \alpha_1, \dots, a_n : \alpha_n)$ *(record),* $\{\alpha\}$ *(finite set),* $[\alpha]$ *(list),* $\langle\alpha\rangle$ *(bag) or* $\alpha \cup \beta$ *(union).*

3. A type t *is either a base type, a type constructor, a generalized constructor that results from replacing some parameters in a type constructor by types or a recursive type defined by an equation* $t = \{\alpha/t\}.t'$*, where* t' *is a generalized constructor and one of its parameters* α *is replaced by* $t \in N_T$.

In the latter two cases the remaining parameters of the type constructor together with the parameters of the replacing types yield the parameters $\alpha_1, \dots, \alpha_n$ *of* t.

4. A type t is called proper *iff the number of its parameters is 0. t is called a* value type *iff there is no occurrence of* ID *in* t.

5. A type form *consists of a type name* $t \in N_T$ *and a type* t' *with possibly some of its parameters replaced by type names.*

6. A type specification $\mathcal{T}$ *is a finite collection of type forms* $t_1, \dots, t_n$ *such that the only type names occurring herein are the names of* $t_1, \dots, t_n$.

The semantics of such types as sets of values is defined as usual. Moreover, we assume the standard operators on base types and on records, sets, bags, ... We omit the details here.

If t' is a proper type occurring in a type t, then there exists a corresponding *occurrence relation*

$$o : t \times t' \rightarrow BOOL \ .$$

Finally, we introduce subtypes. For a more detailed introduction to types see either [14] or [49].

Definition 2 *1. A* subtype relation $\leq$ *on types is given by the following rules:*

(a) Every type t is its own subtype and a subtype of $\perp$.

(b) $NAT \leq INT \leq FLOAT$.

(c) $(\dots, a_{i-1} : \alpha_{i-1}, a_i : \alpha_i, a_{i+1} : \alpha_{i+1}, \dots) \leq (\dots, a_{i-1} : \alpha'_{i-1}, a_{i+1} : \alpha'_{i+1}, \dots)$ *whenever* $\alpha_j \leq \alpha'_j$.

(d) $\left\{ \begin{array}{ccc} \{\alpha\} & \leq & \{\beta\} \\ [\alpha] & \leq & [\beta] \\ \langle\alpha\rangle & \leq & \langle\beta\rangle \end{array} \right\}$ *iff* $\alpha \leq \beta$.

(e) $\{\alpha\} \leq \langle\alpha\rangle$ *and* $[\alpha] \leq \langle\alpha\rangle$.

(f) $\alpha, \beta \leq \alpha \cup \beta$.

2. A subtype function *is a function $t' \rightarrow t$ from a subtype to its supertype ($t' \leq t$) defined by (a)-(f) above.*

3.2 The Class Concept as a Structural Primitive

The class concept provides the grouping of objects having the same structure which uniformly combines aspects of object values and references. Moreover, generic operations on objects such as object creation, deletion and update of its values and references are associated with classes provided these operations can be defined unambigously. Objects can belong to different classes, which guarantees each object of our abstract object model to be captured by the collection of possible classes. As for values that are only defined via types, objects can only be defined via classes.

Each object in a class consists of an identifier, a collection of values and references to objects in other classes. Identifiers can be represented using the unique identifier type ID. Values and references can be combined into a representation type, where each occurence of ID denotes references to some other classes. Therefore, we may define the structure of a class using parameterized types.

Definition 3 *1. Let t be a value type with parameters $\alpha_1, \dots, \alpha_n$. For distinct reference names $r_1, \dots, r_n \in N_R$ and class names $C_1, \dots, C_n \in N_C$ the expression derived from t by replacing each α_i in t by $r_i : C_i$ for $i = 1, \dots, n$ is called a* structure expression.

2. A structural class *consists of a class name $C \in N_C$, a structure expression S and a set of class names $D_1, \dots, D_m \in N_C$ (in the following called the set of* superclasses*). We call r_i the* reference *named r_i from class C to class C_i. The type derived from S by replacing each reference $r_i : C_i$ by the type ID is called the* representation type *T_C of the class C, the type $U_C = (ident : ID, value :: T_C)$ is called the* class type *of C.*

3. A (structural) schema *S is a finite collection of structural classes $C_1, \dots, C_n$ closed under references and superclasses.*

4. *An* instance $\mathcal{D}$ *of a structural schema* S *assigns to each class* C *a value* $\mathcal{D}(C)$ *of type* U_C *such that the following conditions are satisfied:*

   uniqueness of identifiers: *For every class* C *we have*

   $$\forall i :: ID.\forall v, w :: T_C.(i, v) \in \mathcal{D}(C) \wedge (i, w) \in \mathcal{D}(C) \Rightarrow v = w . \quad (4)$$

   inclusion integrity: *For a subclass* C *of* C' *we have*

   $$\forall i :: ID.i \in dom(\mathcal{D}(C)) \Rightarrow i \in dom(\mathcal{D}(C')) . \quad (5)$$

   Moreover, if T_C *is a subtype of* T'_C *with subtype function* $f : T_C \to T'_C$, *then we have*

   $$\forall i :: ID.\forall v :: T_C.(i, v) \in \mathcal{D}(C) \Rightarrow (i, f(v)) \in \mathcal{D}(C') . \quad (6)$$

   referential integrity: *For each reference from* C *to* C' *with corresponding occurrence relation* o_r *we have*

   $$\forall i, j :: ID.\forall v :: T_C.(i, v) \in \mathcal{D}(C) \wedge o_r(v, j) \Rightarrow j \in dom(\mathcal{D}(C')) . \quad (7)$$

3.3 User Defined Integrity Constraints

Let us now extend the notion of schema by the introduction of explicit user-defined integrity constraints. First we define the notion of constraint schema in general, then we restrict ourselves to distinguished classes of constraints that arise as generalizations of constraints known from the relational model, e.g. functional and key constraints, inclusion and exclusion constraints [48,52].

Definition 4 *Let* $S = \{C_1, \ldots, C_n\}$ *be a structural schema.*

1. *An* integrity constraint *on* S *is a formula* I *over the underlying type system with free variables* $fr(I) \subseteq \{x_{C_1}, \ldots, x_{C_n}\}$, *where each* x_{C_i} *is a variable of type* $\{U_{C_i}\}$. *We call* x_{C_i} *the* class variable *of* C_i.

2. *A* constrained schema *consists of a structural schema* S *and a finite set of integrity constraints on* S.

3. *An* instance *of a constrained schema is an instance of the underlying structural schema. An instance* $\mathcal{D}$ *is said to be* consistent *with respect to the integrity constraint* I *iff substituting* $\mathcal{D}(C)$ *for each class variable* x_C *in* I *evaluates to true, when interpreted in the usual way.*

Note that the conditions for an instance in Definition 4 correspond to model inherent integrity constraints. We refer to these constraints as implicit *identifier*, *IsA* and *referential* constraints on the schema S. Let us now define some distinguished classes of user-defined constraints.

Definition 5 *Let* C, C_1, C_2 *be classes in a schema* S *and let* $c^i : T_C \to T_i$ $(i = 1, 2, 3)$ *and* $c_i : T_{C_i} \to T$ $(i = 1, 2)$ *be subtype functions.*

1. *A* functional constraint *on* C *is a constraint of the form*

   $$\forall i, i' :: ID.\forall v, v' :: \\ T_C.\, c^1(v) = c^1(v') \wedge (i, v) \in x_C \wedge (i', v') \in x_C \Rightarrow c^2(v) = c^2(v'). \quad (8)$$

2. *A* uniqueness constraint *on C is a constraint of the form*

$$\forall i, i' :: ID. \forall v, v' :: \\ T_C.\ c^1(v) = c^1(v') \wedge (i, v) \in x_C \wedge (i', v') \in x_C \Rightarrow i = i'. \tag{9}$$

A uniqueness constraint on C is called trivial *iff $T_C = T_I$ and $c^1 = id$ hold.*

3. *An* inclusion constraint *on C_1 and C_2 is a constraint of the form*

$$\forall t :: T. \exists i_1 :: ID, v_1 :: T_{C_1}.(i_1, v_1) \in x_{C_1} \wedge c_1(v_1) = t \Rightarrow \\ \exists i_2 :: ID, v_2 :: T_{C_2}.(i_2, v_2) \in x_{C_2} \wedge c_2(v_2) = t\ . \tag{10}$$

4. *An* exclusion constraint *on C_1, C_2 is a constraint of the form*

$$\forall i_1, i_2 :: ID. \forall v_1 :: T_{C_1}. \forall v_2 :: \\ T_{C_2}.(i_1, v_1) \in x_{C_1} \wedge (i_2, v_2) \in x_{C_2} \Rightarrow c_1(v_1) \neq c_2(v_2)\ . \tag{11}$$

3.4 Methods as a Basis for Behaviour Modelling

So far, only static aspects have been considered. A structural schema is simply a collection of data structures called classes. Let us now turn to adding dynamics to this picture. As required in the object oriented approach operations will be associated with classes. This gives us the notion of a method.

We shall distinguish between visible and hidden methods to emphasize those methods that can be invoked by the user and others. This is not intended to define an interface of a class, since for the moment all methods of a class including the hidden ones can be accessed by other methods. The justification for such a weak hiding concept is due to two reasons.

- Visible methods serve as a means to specify (nested) transactions. In order to build sequences of database instances we only regard these transactions assuming a linear invocation order on them.
- Hidden methods can be used to handle identifiers. Since these identifiers do not have any meaning for the user, they must not occur within the input or output of a transaction.

Definition 6 *Let S be a structural schema.*
Let $T_1, \dots, T_n, T'_1, \dots, T'_m$ be types, $M \in N_M$ and $\iota_1, \dots, \iota_n, o_1, \dots, o_m \in V$.

1. *A* method signature *consists of a method name M, a set of input-parameter / input-type pairs $\iota_i :: T_i$ and a set of output-parameter / output-type pairs $o_j :: T'_j$. We write*

$$o_1 :: T'_1, \dots, o_m :: T'_m \leftarrow M(\iota_1 :: T_1, \dots, \iota_n :: T_n)\ .$$

2. *Let C be some structural class in S. A* method *M on C consists of a method signature with name M and a body that is recursively built from the following constructs:*

   (a) *assignment $x := E$, where x is either the class variable x_C or a local variable within S, and E is a term of the same type as x,*

(b) *skip, fail, loop,*

(c) *sequential composition* $S_1; S_2$, *choice* $S_1 \square S_2$, *projection* $x :: T \mid S$, *guard* $\mathcal{P} \rightarrow S$, *restricted choice* $S_1 \boxtimes S_2$, *where* $\mathcal{P}$ *is a well-formed formula and* x *is a variable of type* T, *and*

(d) *instantiation* $x'_1, \ldots, x'_i \leftarrow C' : S'(E'_1, \ldots, E'_j)$, *where* S' *is a method on class* C' *with input-parameters* $\iota'_1, \ldots, \iota'_j$ *and output-parameters* $o'_1, \ldots, o'_i$, *such that the variables* o'_f, x'_f *have the same type and the term* E'_g *has the same type as the variable* ι'_g.

3. *A method* M *on a class* C *with signature* $o_1 :: T'_1, \ldots, o_m :: T'_m \leftarrow M(\iota_1 :: T_1, \ldots, \iota_n :: T_n)$ *is called* value-defined *iff all* T_i $(i = 1 \ldots n)$ *and* T'_j $(j = 1, \ldots, m)$ *are proper value types.*

As already mentioned the OODM distinguishes between transactions, i.e. methods visible to the user, and hidden methods. We require each transaction to be value-defined.

Subclasses inherit the methods of their superclasses, but overriding is allowed as long as the new method is a specialization of all its corresponding methods in its superclasses. Overriding becomes mandatory in the case of multiple inheritance with name conflicts. A method that overrides a hidden method on some superclass must also be hidden.

Definition 7 *Let* $\mathcal{S}$ *be a structural schema and* $C \in \mathcal{S}$ *be a structural class as in Definition 3 with superclasses* $D_1, \ldots, D_k$. *A* method specification *on* C *consists of two sets of methods* $\mathcal{S} = \{M_1, \ldots, M_n\}$ *(called* transactions*) and* $\mathcal{H} = \{M'_1, \ldots, M'_m\}$ *(called* hidden methods*) such that the following properties hold:*

1. *Each* M_i $(i = 1, \ldots, n)$ *is value-defined.*

2. *For each transaction* M^l *on some superclass* D_l *there exists some* $i \in \{1, \ldots, n\}$ *such that* M_i *specializes* M^l.

3. *For each hidden method* M^l *on some superclass* D_l *there exists some* $j \in \{1, \ldots, m\}$ *such that* M'_j *specializes* M^l.

Let us briefly discuss what specialization means for the input- and output-types. Sometimes it is required that the input-type for an overriding method should be a subtype of the original one (covariance rule), sometimes the opposite (contravariance rule) is required. The first rule applies e.g. if we want to override an insert method. In this case the inherited method has no effect on the subclass, but simply calls the "old" method. The second rule applies if input-types required on the superclass can be omitted on the subclass. Both rules are captured by the formal notion of specialization. We omit the details [44]. Now we are prepared to generalize the definition of classes and schemata.

Definition 8 1. *A* class *consists of a class name* $C \in N_C$, *a structure expression* S, *a set of class names* $D_1, \ldots, D_m \in N_C$ *(called the set of* superclasses*) and a method specification* $(\mathcal{S} = \{M_1, \ldots, M_n\}$, $\mathcal{H} = \{M'_1, \ldots, M'_{n'}\})$ *on* C.

2. *A* (behavioural) schema S *is a finite collection of classes* $\{C_1, \ldots, C_n$ *closed under references, superclasses and method call together with a collection of integrity constraints* $I_1, \ldots, I_n$ *on* S.

3. *An* instance $\mathcal{D}$ *of a behavioural schema* S *is an instance of the underlying structural schema. A* database **history** *on* S *is a sequence* $\mathcal{D}_0, \mathcal{D}_1, \ldots$ *of instances such that each transition from* $\mathcal{D}_{i-1}$ *to* $\mathcal{D}_i$ *is due to some transaction on some class* $C \in S$.

Note the relation between database histories used here and the work on the semantics of object bases in [22,28].

3.5 Queries and Views

Roughly speaking the querying of a database is an operation on the database without changing its state. The emphasis of a query is on the output. While such a general view of queries can be subsumed by transactions, hence by methods in the OODM, query languages are in particular intended to be declarative in order to support an ad-hoc querying of a database without the need to write new transactions [8].

Querying a relational database can be expressed by terms in relational algebra. This view can be easily generalized to the OODM using its type system. Therefore, terms over such types occur naturally. Moreover, type specifications are based on other type specifications via constructors, selectors and functions. Hence, $\mathcal{T}$ allows arbitrary terms involving more than one class variable x_C to be built. Then a *query* turns out be be represented by term t over some type T such that the free variables of t are all class variables. This approach is in accordance with the algebraic approach in [12] and with so called *universal traversal combinators* [25].

In relational algebra a *view* may be regarded simply as a stored query (or derived relation). We shall try to generalize also this view to the OODM.

However, things change dramatically, when object identifiers come into play [13], since now we have to distinguish between queries that result in values and those that result in (collections of) objects. Therefore we distinguish in the OODM between value queries and general access expressions.

A *value query* on a schema S can then be represented by a term t of some value type T with $fr(t) \subseteq \{x_C \mid C \in S\}$. Ad-hoc querying of a database should then be restricted to value queries. This is no loss of generality, because for any type T in $\mathcal{T}$ involving identifiers there exists a corresponding type T' allowing multiple occurrences. Take e.g. a class C. If we want to get all the objects in that class no matter whether they have the same values or not, the corresponding term would be x_C. This is not a value query, but if T_C is a value type, we may take $T' = \langle T_C \rangle$ and the natural projection given by the subtype functions

$$\{(ident : ID, value : \alpha)\} \rightarrow \langle (ident : ID, value : \alpha) \rangle \rightarrow \langle \alpha \rangle \ .$$

In the case of arbitrary access expressions another problem occurs [13]. So far, we can only build terms t that involve identifiers already existing in the database. Thus, such queries are called *object preserving*. If we want the result of a query to represent "new" objects, i.e. if we want to have *object generating queries*, we have to apply a mechanism to create new object identifiers. This can be achieved by *object creating functions* on the type ID with arity $ID \times \ldots \times ID \rightarrow ID$ [32,35].

The idea that a view is a stored query then carries over easily. However, the structure of a view should be compatible with the structure of the schema, i.e. each view may be regarded as a derived class. Summarizing, we get the following formal definition.

Definition 9 *Let $S = \{C_1, \ldots, C_n\}$ be some schema.*

1. *A* value query *on S is a term t over some proper value type T with $fr(t) \subseteq \{x_{C_1}, \ldots, x_{C_n}\}$.*

2. *An* access expression *on S is a term t over some proper type T with $fr(t) \subseteq \{x_{C_1}, \ldots, x_{C_n}\}$.*

3. *A* view *on S consists of a view name $v \in N_C$ such that there is no class $C \in S$ with this name, a structure expression $S(v)$ containing references to classes in S or to views on S and a defining access expression $t(v)$ of type $\{U_v\}$, where T_v is the representation type corresponding to $S(v)$.*

4. *A* (complete) schema *is a behavioural schema together with a finite set of views. An* instance *of a complete schema is an instance of the underlying structural schema such that for every view v replacing each class variable x_C in the access expressions of v yields a value of type $\{U_v\}$ satisfying the uniqueness property for identifiers.*

4 The Object Identification Problem

From an object oriented point of view a database may be considered as a huge collection of objects of arbitrary complex structure. Hence the problem to uniquely identify and retrieve objects in such collections.

Each object in a database is an abstraction of a real world object that has a unique *identity*. The representation of such objects in the OODM uses an abstract identifier I of type ID to encode this identity. Such an identifier may be considered as being immutable. However, from a systems oriented view permutations or collapses of identifiers without changing anything else should not affect the behaviour of the database.

For the user the abstract identifier of an object has no meaning. Therefore, a different access to the identification problem is required. We show that the unique identification of an object in a class leads to the notion of *(weak) value-identifiability*, where weak value-representability can be used to capture also objects that do not exists for there own, but depend on other objects. This is related to *weak entities* in entity-relationship models [62]. The stronger notion of *value-representability* is required for the unique definition of generic update operations.

4.1 The Notion of Value-Representability

According to our definitions two objects in a class C are identical iff they have the same identifier. By the use of constraints, especially uniqueness constraints, we could restrict this notion of equality.

Let us address the characterization of those classes, the objects in which are completely representable by values, i.e. we could drop the object identifiers and replace references by values of the referred object. We shall see in Section 5 that in case of value-representable classes we are able to preserve an important advantage of relational databases, i.e. the existence of structurally determined update operations.

Definition 10 *Let C be a class in a schema S with representation type T_C.*

1. *C is called* value-identifiable *iff there exists a proper value type I_C such that for all instances $\mathcal{D}$ of S there is a function $c : T_C \rightarrow I_C$ such that the uniqueness constraint on C defined by c holds for $\mathcal{D}$.*

2. *C is called* value-representable *iff there exists a proper value type V_C such that for all instances $\mathcal{D}$ of S there is a function $c : T_C \rightarrow V_C$ such that for $\mathcal{D}$*

   (a) *the uniqueness constraint on C defined by c holds and*

   (b) *for each uniqueness constraint on C defined by some function $c' : T_C \rightarrow V'_C$ with proper value type V'_C there exists a function $c'' : V_C \rightarrow V'_C$ that is unique on $c(codom(\mathcal{D}(C)))$ with $c' = c'' \circ c$.*

It is easy to see that each value-representable class C is also value-identifiable. Moreover, the *value-representation type* V_C in Definition 10 is unique up to isomorphism.

4.2 Value-Representability in the Case of Acyclic Reference Graphs

Since value-representability is defined by the existence of a certain proper value type, it is hard to decide, whether an arbitrary class is value-representable or not. In case of simple classes the problem is easier, since we only have to deal with uniqueness and value constraints. In this case it is helpful to analyse the reference structure of the class. Hence the following graph-theoretic definitions.

Definition 11 *The* reference graph *of a class C in a schema S is the smallest labelled graph $G_{rep} = (V, E, l)$ satisfying:*

1. *There exists a vertex $v_C \in V$ with $l(v_C) = \{t, C\}$, where t is the top-level type in the structure expression S of C.*

2. *For each proper occurrence of a type $t \neq ID$ in T_C there exists a unique vertex $v_t \in V$ with $l(v_t) = \{t\}$.*

3. *For each reference $r_i : C_i$ in the structure expression S of C the reference graph G^i_{ref} is a subgraph of G_{ref}.*

4. *For each vertex v_t or v_C corresponding to $t(x_1, \ldots, x_n)$ in S there exist unique edges $e_t^{(i)}$ from v_t or v_C respectively to v_{t_i} in case x_i is the type t_i or to v_{C_i} in case x_i is the reference $r_i : C_i$. In the first case $l(e_t^{(i)}) = \{S_i\}$, where S_i is the corresponding selector name; in the latter case the label is $\{S_i, r_i\}$.*

Definition 12 1. *Let $S = \{C_1, \ldots, C_n\}$ be a schema. Let $S' = \{C'_1, \ldots, C'_n\}$ be another schema such that for all i there exists a uniqueness constraint on C_i defined by some $c_i : T_{C_i} \rightarrow T_{C'_i}$. Then an* identification graph *G_{id} of the class C_i is obtained from the reference graph of C'_i by changing each label C'_j to C_j.*

2. *The identification graph G_{id} resulting from the use of trivial uniqueness constraints is called the* standard identification graph.

Clearly, there need not exist any identification graph nor does the existence of one identification graph imply the existence of the standard one. However, if the standard identification graph exist, then it is equal to the reference graph.

Proposition 13 *Let C be a class in a schema S with acyclic reference graph G_{ref} such that there exist uniqueness constraints for C and each C_i such that C_i occurs as a label in G_{ref}. Then C is value-representable.*

Proof. We use induction on the maximum length of a path in G_{ref}. If there are no references in the structure expression S of C the type T_C is a proper value type. Since there exists a uniqueness constraint on C, the identity function id on T_C also defines a uniqueness constraint. Hence $V_C = T_C$ satisfies the requirements of Definition 10.

If there are references $r_i : C_i$ in the structure expression S of C, then the induction hypothesis holds for each such C_i, because G_{ref} is acyclic. Let V_C result from S by replacing each $r_i : C_i$ by V_{C_i}. Then V_C satisfies the requirements of Definition 10. □

Corollary 14 *Let C be a class in a schema S such that there exist an acyclic identification graph G_{id} and uniqueness constraints for C and each C_i occuring as a label in G_{id}. Then C is value-identifiable.*

4.3 Computation of Value Representation Types

We want to address the more general case where cyclic references may occur in the schema $S = \{C_1, \ldots, C_n\}$. In this case a simple induction argument as in the proof of Proposition 13 is not applicable. So we take another approach. We define algorithms to compute types V_C and I_C that turn out to be proper value types under certain conditions. In the next subsection we then show that these types are the value representation type and the value identification type required by Definition 10.

Algorithm 15 *Let $F(C_i) = T_i$ provided there exists a uniqueness constraint on C_i defined by $c_i : T_{C_i} \to T_i$, otherwise let $F(C_i)$ be undefined. If ID occurs in some $F(C_i)$ corresponding to $r_j : C_j$ $(j \neq i)$, we write ID_j.*

Then iterate as long as possible using the following rules:

1. *If $F(C_j)$ is a proper value type and ID_j occurs in some $F(C_i)$ $(j \neq i)$, then replace this corresponding ID_j in $F(C_i)$ by $F(C_j)$.*

2. *If ID_i occurs in some $F(C_i)$, then let $F(C_i)$ be recursively defined by $F(C_i) == S_i$, where S_i is the result of replacing ID_i in $F(C_i)$ by the type name $F(C_i)$.*

This iteration terminates, since there exists only a finite collection of classes. If these rules are no longer applicable, replace each remaining occurrence of ID_j in $F(C_i)$ by the type name $F(C_j)$ provided $F(C_j)$ is defined. □

Note that the the algorithm computes (mutually) recursive types. Now we give a sufficient condition for the result of Algorithm 15 to be a proper value type.

Lemma 16 *Let C be a class in a schema S such that there exists a uniqueness constraint for all classes C_i occurring as a label in some identification graph G_{id} of C. Let I_C be the type $F(C)$ computed by Algorithm 15 with respect to the uniqueness constraints used in the definition of G_{id}. Then I_C is a proper value type.*

Proof. Suppose I_C were not a proper value type. Then there exists at least one occurrence of ID in I_C. This corresponds to a class C_i without uniqueness constraint occurring as a label in G_{id}, hence contradicts the assumption of the lemma. □

4.4 The Finiteness Property

Let us now address the general case. The basic idea is that there is always only a finite number of objects in a database. Assuming the database being consistent with respect to inclusion and referential constraints yields that there can not exist infinite cyclic references. This will be expressed by the *finiteness property*. We show that this property allows the computation of value representation types.

Definition 17 *Let C be a class in a schema S and let $g_{k,l}$ denote a path in G_{ref} from v_{C_k} to v_{C_l} provided there is a reference $r_l : C_l$ in the structure expression of C_k. Then a* cycle *in G_{ref} is a sequence $g_{0,1} \cdots g_{n-1,n}$ with $C_0 = C_n$ and $C_k \neq C_l$ otherwise.*

Note that we use paths instead of edges, because the edges in G_{ref} do not always correspond to references. According to our definition of a class there exists a referential constraint on C_k, C_l defined by $o_{k,l} : T_{C_k} \times ID \rightarrow BOOL$ corresponding to $g_{k,l}$. Therefore, to each cycle there exists a corresponding sequence of functions $o_{0,1} \cdots o_{n-1,n}$. This can be used as follows to define a function $cyc : ID \times ID \rightarrow BOOL$ corresponding to a cycle in G_{ref}.

Definition 18 *Let C be a class in a schema S and let $g_{0,1} \cdots g_{n-1,n}$ be a cycle in G_{ref}. The corresponding* cycle relation *$cyc : ID \times ID \rightarrow BOOL$ is defined by $cyc(i,j) = true$ iff there exists a sequence $i = i_0, i_1, \dots, i_n = j$ $(n \neq 0)$ such that $(i_l, v_l) \in C_l$ and $o_{l,l+1}(i_{l+1}, v_l) = true$ for all $l = 0, \dots, n-1$.*

Given a cycle relation cyc, let cyc^m the m-th power of cyc.

Lemma 19 *Let C be a class in a schema S. Then C satisfies the finiteness property, i.e. for each instance $\mathcal{D}$ of S and for each cycle in G_{ref} the corresponding cycle relation cyc satisfies*

$$\forall i \in dom(C). \exists n. \forall j \in dom(C). \exists m < n. \left(cyc^n(i,j) = true \Rightarrow cyc^m(i,j) = true \right) .$$

Proof. Suppose the finiteness property were not satisfied. Then there exist an instance $\mathcal{D}$, a cycle relation cyc and an object identifier i_0 such that

$$\forall n. \exists j \in dom(C). \forall m < n. \left(cyc^n(i_0, j) = true \wedge cyc^m(i_0, j) = false \right)$$

holds. Let such a j corresponding to $n > 0$ be i_n. Then the elements $i_0, i_1, i_2, \dots$ are pairwise distinct. Hence there would be infinitely many objects in $\mathcal{D}$ contradicting the finiteness of a database. □

Lemma 20 *Let $\mathcal{D}$ be an instance of schema $S = \{C_1, \dots, C_n\}$. Then $\mathcal{D}$ satisfies at each stage of Algorithm 15 uniqueness constraints for all $i = 1, \dots, n$ defined by some $c_i^f : T_{C_i} \rightarrow F(C_i)$.*

Proof. It is sufficient to show that whenever a rule is applied replacing $F(C_i)$ by $F(C_i)'$, then $F(C_i)'$ also defines a uniqueness constraint on C_i.

Suppose that $(i, v) \in C_i$ holds in $\mathcal{D}$. Since it is possible to apply a rule to $F(C_i)$, there exists at least one value $j :: ID$ occurring in $c_i(v)$. Replacing ID_j in $F(C_i)$ corresponds to replacing j by some value $v_j :: F(C_j)$. Because of the finiteness property such a value must exist. Moreover, due to the uniqueness constraint defined by c_j the function $f : F(C_i) \to F(C_i)'$ representing this replacement must be injective on $c_i(codo(\mathcal{D}(C_i)))$. Hence, $c_i' = f \circ c_i$ defines a uniqueness constraint on C_i. □

Now assume that we use only trivial uniqueness constraints in Algorithm 15. In order to distinguish this situation from the general case we write $G(C_i)$ instead of $F(C_i)$ to refer to this special case.

Lemma 21 *Let $\mathcal{D}$ be an instance of schema $S = \{C_1, \ldots, C_n\}$. Then at each stage of Algorithm 15 (applied with arbitrary uniqueness constraints and in parallel with trivial ones) there exists for all $i = 1, \ldots, n$ a function $\bar{c}_i : G(C_i) \to F(C_i)$ that is unique on $c_i(codom(\mathcal{D}(C_i)))$ with $c_i' = \bar{c}_i \circ c_i$.*

Proof. As in the proof of Lemma 20 it is sufficient to show that the required property is preserved by the application of a rule from any of the two versions of Algorithm 15. Therefore, let $\bar{c}_i$ satisfy the required property and let $g : G(C_i) \to G(C_i)'$ and $f : F(C_i) \to F(C_i)'$ be functions corresponding to the application of a rule to $G(C_i)$ and $F(C_i)$ respectively. Such functions were constructed in the proofs of Lemma 20 and Lemma 20 respectively.

Then $f \circ \bar{c}_i$ satisfies the required property with respect to the application of f. In the case of applying g we know that g is injective on $c_i(codom(\mathcal{D}(C_i)))$. Let $h : G(C_i)' \to G(C_i)$ be any continuation of $g^{-1} : g(c_i(codom(\mathcal{D}(C_i)))) \to G(C_i)$. Then $\bar{c}_i \circ h$ satisfies the required property. □

Theorem 22 *Let C be a class in a schema S such that there exists a uniqueness constraint for all classes C_i occurring as a label in the reference graph G_{ref} of C. Let V_C be the type $G(C)$ computed by Algorithm 15 with respect to trivial uniqueness constraints and let I_C be the type $F(C)$ computed by Algorithm 15 with respect to arbitrary uniqueness constraints. Then C is value-representable with value representation type V_C and each such I_C is a value identification type.*

Proof. V_C is a proper value type by Lemma 16. From Lemma 20 it follows that if $\mathcal{D}$ is an instance of S, then there exists a function $c : T_C \to V_C$ such that the uniqueness constraint defined by c holds for $\mathcal{D}$. The same applies to I_C.

If V_C' is another proper value type and $\mathcal{D}$ satisfies a uniqueness constraint defined by $c' : T_C \to V_C'$, then V_C' is some value-identification type I_C. Hence by Lemma 21 there exists a function $c'' : V_C \to V_C'$ that is unique on $c(codom(\mathcal{D}(C)))$ with $c' = c'' \circ c$. This proves the Theorem. □

Corollary 23 *Let S be a schema such that all classes C in S are value-identifiable. Then all classes C in S are also value-representable.*

□

4.5 Weak Value-Representability

Let us now ask whether there exist also weaker identification mechanisms other than value-representability. In several papers, e.g. [42] a navigational approach on the basis of the reference structure has been favoured. This leads to dependent classes similar to "weak entities" in the entity-relationship model [62]. We shall show that such an approach requires at least a value-identifiable "entrance" of some path and the hard restriction on references to be representable by surjective functions.

Definition 24 *Let S be some schema.*

1. If r is a reference from class C to D in S and $o : T_C \times ID \rightarrow BOOL$ is the function of Definition 4 expressing the corresponding referential constraint, then r satisfies the *(SF)-condition* iff

   (a) $o(v,i) \wedge o(v,j) \Rightarrow i = j$ and

   (b) $j \in dom(x_D) \Rightarrow \exists v :: T_C . v \in codom(x_C) \wedge o(v,j)$

   hold for all $i, j :: ID, v :: T_C$.

2. An *(SF)-chain* from class D to C in S is a sequence of classes $D = C_0, \ldots, C_n = C$ such that for all i $(i = 1, \ldots, n)$ either C_i is a subclass of C_{i-1} or there exists a reference r_i from C_{i-1} to C_i satisfying the (SF)-condition.

3. A class C in S is called *weakly value-identifiable* iff there exists a value-identifiable class D and an (SF)-chain from D to C.

The notation (SF)-condition has been chosen to emphasize that such a reference represents a surjective function. It is easy to see taking $n = 0$ that each value-identifiable class is also weakly value-identifiable.

Lemma 25 *If C is a weakly value-identifiable class in a schema S, then there exists a proper value type I_C such that for each instance $\mathcal{D}$ of S there exists a function $c : ID \rightarrow I_C$ such that c is injective on $dom(\mathcal{D}(C))$.*

Call I_C a *weak value-identification type* of the class C.

Proof. Let $D = C_0, \ldots, C_n = C$ be an (SF)-chain from the value-identifiable class D to C with corresponding references r_i $(i = 1, \ldots, n)$. If r_i satisfies the (SF)-condition, there exists a function $c_i : ID \rightarrow ID$ such that $j \in dom(\mathcal{D}(C_i)) \Rightarrow (c_i(j), v) \in x_{C_{i-1}}$ for some v with $o_i(v,j)$ (just take some inverse image of j under the surjective reference function). Since r_i defines a function, c_i is clearly injective. If C_i is a subclass of C_{i-1}, then take $c_i = id$.

If $c' : ID \rightarrow I_D$ is the function defined by the uniqueness constraint on D and $c'' : ID \rightarrow ID$ is the concatenation $c_1 \circ \ldots \circ c_n$, then $c = c' \circ c''$ satisfies the required property. □

Definition 26 *A class C in a schema S is called* weakly value-representable *iff there exists a proper value type V_C such that for each instance $\mathcal{D}$ of S the following properties hold.*

1. *There is a function $c : ID \rightarrow V_C$ that is injective on $dom(\mathcal{D}(C))$.*

2. *For each proper value type V_C' and each function $c' : ID \rightarrow V_C'$ that is injective on $dom(\mathcal{D}(C))$ there exists a function $c'' : V_C \rightarrow V_C'$ that is unique on $c(dom(\mathcal{D}(C)))$ with $c' = c'' \circ c$.*

We call V_C the weak value-representation type *of the class C.*

Note that the weak value-representation type is unique provided it exists. Again it is easy to see that value-representability implies weak value-representability. Moreover, due to Lemma 25 each weakly value-representable class is also weakly value-identifiable. We shall see that also the converse of this fact is true.

We want to compute weak value representation types. This can be done using a slight modification of Algorithm 15 that completely ignores uniqueness constraints. We refer to this algorithm as the *blind* version of Algorithm 15 and to emphasize this, we write $H(C_i)$ instead of $F(C_i)$. Analogous to Lemmata 16 and 20 the following results holds.

Lemma 27 *Let C be a class in a schema S and let I_C be the type $H(C)$ computed by the blind version of Algorithm 15. Then I_C is a proper value type.*

Lemma 28 *Let $\mathcal{D}$ be an instance of the schema $S = \{C_1, \ldots, C_n\}$. Let C, D be classes such that C is weakly value-identifiable, D is value-identifiable and there exists some (SF)-chain from D to C. Let $c : ID \to I_C$ be the function of Lemma 25 corresponding to this chain. Let $c' : ID \to H(D)$ be a function corresponding to the uniqueness constraint on D and the instance $\mathcal{D}$. Then at each stage of the blind version of Algorithm 15 there exists a function $\bar{c} : H(D) \to I_C$ that is unique on $c'(dom_{\mathcal{D}}(C))$ with $c = \bar{c} \circ c'$.*

Based on these two lemmata we can now state the main result on weak value representability.

Theorem 29 *Let C be a weakly value-identifiable class in a schema S andlet V_C be the product of all types $H(D)$, where D is the leading value-identifiable class in some maximal (SF)-chain corresponding to C and $H(D)$ is the result of the blind version of Algorithm 15. Then C is weakly value-representable with weak value-representation type V_C.*

Proof. V_C is a proper value type by Lemma 27. From Lemmata 20 and 25 it follows that there exists a function $c' : ID \to V_C$ that is injective on $dom_{\mathcal{D}}(C)$.

From Lemma 28 it follows that there exists a function $\bar{c} : V_C \to I_C$ that is unique on $c'(dom(\mathcal{D}(C)))$ with $c = \bar{c} \circ c'$. This proves the Theorem. □

5 The Genericity Problem

The preservation of advantages of relational databases requires generic operations for querying and for the insertion, deletion and update of single objects. While querying [1,12,30,55] is per se a set-oriented operation, i.e. it is not necessary to select just one single object, and hence does not raise any specific problems with object identifiers, things change completely in case of updates. If an object with a given value is to be updated (or deleted), this is only defined unambigously, if there does not exist another object with the same value. If more than one object exists with the same value or more generally with the same value and the same references to other objects, then the user has to decide, whether an update- or delete-operation is applied to *all* these objects, to only *one* of these objects selected non-deterministically or to *none* of them, i.e. to reject the operation. However, it is not possible to specify a priori such an operation that works in the same way for all objects in all situations. The same applies to insert-operations. Hence the problem, in which cases operations for the insertion, deletion and update of objects can be defined generically.

Some authors [43] have chosen the solution to abandon generic operations. Others [6,7,9] use identifying values to represent object identity, thus embody a strict concept of surrogate keys to avoid the problem. Our approach is different from both solutions in that we use the concept of hidden abstract identifiers, but at the same time formally characterize those classes for which unique generic methods for the insertion, deletion and update of single objects exist. At the same time inclusion and referential integrity have to be enforced. We show that these classes are the value-representable ones.

5.1 Generic Update Methods

The requirement that object-identifiers have to be hidden from the user imposes the restriction on canonical update operations to be value-defined in the sense that the identifier of a new object has to be chosen by the system whereas all input- and output-data have to be values of proper value types.

We now formally define what we mean by generic update methods. For this purpose regard an instance $\mathcal{D}$ of a schema $\mathcal{S}$ as a set of objects. For each recursively defined type T let $\bar{T}$ denote by replacing each occurrence of a recursive type T' in T by $UNION(T', ID)$.

Definition 30 *Let C be a class in a schema $\mathcal{S}$.* Generic update methods *on C are $insert_C$, $delete_C$ and $update_C$ satisfying the following properties:*

1. *Their input types are proper value types; their output type is the trivial type $\perp$.*

2. *In the case of insert applied to an instance $\mathcal{D}$ there exists some $o :: U_C$ such that*

   (a) *the result is an instance $\mathcal{D}'$ with $o \in \mathcal{D}'$ and $\mathcal{D} \subseteq \mathcal{D}'$ hold and*

   (b) *if $\bar{\mathcal{D}}$ is any instance with $\mathcal{D} \subseteq \bar{\mathcal{D}}$ and $o \in \bar{\mathcal{D}}$, then $\mathcal{D}' \subseteq \bar{\mathcal{D}}$.*

3. *In the case of delete applied to an instance $\mathcal{D}$ there exists some $o :: U_C$ such that*

   (a) *the result is an instance $\mathcal{D}'$ with $o \notin \mathcal{D}'$ and $\mathcal{D}' \subseteq \mathcal{D}$ hold and*

   (b) *if $\bar{\mathcal{D}}$ is any instance with $\bar{\mathcal{D}} \subseteq \mathcal{D}$ and $o \notin \bar{\mathcal{D}}$, then $\bar{\mathcal{D}} \subseteq \mathcal{D}'$.*

4. *In the case of update applied to an instance $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$, where $\mathcal{D}_2 = \{o\}$ if $o \neq o'$ and $\mathcal{D}_2 = \emptyset$ otherwise there exist $o, o' :: U_C$ with $o = (i, v)$ and $o' = (i, v')$ such that*

   (a) *the result is an instance $\mathcal{D}' = \mathcal{D}_1 \cup \mathcal{D}_2'$ with $\mathcal{D}_2 \cap \mathcal{D}_2' = \emptyset$,*

   (b) *$o \in \mathcal{D}$, $o' \in \mathcal{D}'$,*

   (c) *if $\bar{\mathcal{D}}$ is any instance with $\mathcal{D}_1 \subseteq \bar{\mathcal{D}}$ and $o' \in \bar{\mathcal{D}}$, then $\mathcal{D}' \subseteq \bar{\mathcal{D}}$.*

Canonical update methods *on C are $insert'_C$, $delete'_C$ and $update'_C$ defined analogously with the only difference of their output type being ID and their input-type being $\bar{T}$ for some value-type T.*

Note that this definition of genericity includes the consistency with respect to the implicit constraints on $\mathcal{S}$. We show that value-representability is necessary and sufficient for the existence and uniqueness of such operations.

Lemma 31 *Let C be a class in a schema $\mathcal{S}$ such that there exist canonical update methods on C. Then also generic update methods exist on C.*

Proof. In the case of *insert* define $insert_C(V :: V_C) == I \leftarrow insert'_C(V)$, i.e. call the corresponding canonical operation and ignore its output. The same argument applies to *delete* and *update*. □

Theorem 32 *Let C be a class in a schema $\mathcal{S}$ such that there exist generic update methods on C. Then C is value-representable. Moreover, all super- and subclasses of C are also value-representable.*

Proof. First consider the *delete* method with input type I_C which is by definition a proper value type. We show that it is already a value identification type.

If not, then for all instances $\mathcal{D}$ and all functions $c : T_C \rightarrow I_C$ there exist $i, j :: ID$ and $v, w :: T_C$ with

$$i \neq j \wedge (i, v) \in \mathcal{D}(C) \wedge (j, w) \in \mathcal{D}(C) \wedge c(v) = c(w) \ . \tag{12}$$

Now take $o = (i, v)$ and $o' = (j, w)$. Then there exist two distinct instances $\mathcal{D}'$ and $\mathcal{D}''$ satisfying the conditions of Definition 30(iii) with respect to o and o' respectively, hence contradict the assumption of a unique generic *delete*-method on C.

The same argument applies to the input-type V_C. Moreover, since insertion requires all values of referenced object to be provided, we derive from Algorithm 15 and Theorem 22 that V_C is a value representation type. Therefore, C is value-representable.

The value-representability on superclasses is implied, since *insert* (and *update*) on C involve the corresponding method on each superclass. The value-representability of subclasses follows from the propagation of *update* through them. We omit the technical details. □

5.2 Generic Updates in the Case of Value-Representability

Our next goal is to reduce the existence problem of canonical update operations to schemata without IsA relations.

Lemma 33 *Let C, D be value-representable classes in a schema $\mathcal{S}$ such that C is a subclass of D with subtype function $g : T_C \rightarrow T_D$. Then there exists a function $h : V_C \rightarrow V_D$ such that for each instance $\mathcal{D}$ of $\mathcal{S}$ with corresponding functions $c : T_C \rightarrow V_C$ and $d : T_D \rightarrow V_D$ we have $h(c(v)) = d(g(v))$ for all $v \in codom(\mathcal{D}(C))$.*

Proof. By Definition 10 c is injective on $codom(\mathcal{D}(C))$, hence any continuation h of $d \circ g \circ c^{-1}$ satisfies the required property.

It remains to show that h does not depend on $\mathcal{D}$. Suppose $\mathcal{D}_1$, $\mathcal{D}_2$ are two instances such that $w = c_1(v_1) = c_2(v_2) \in V_C$, where c_1, d_1, h_1 correspond to $\mathcal{D}_1$ and c_2, d_2, h_2 correspond to $\mathcal{D}_2$. Then there exists a permutation π on ID such that $v_2 = \pi(v_1)$. We may extend π to a permutation on any type. Since ID has no non-trivial supertype, g permutes with π, hence $g(v_2) = \pi(g(v_1))$. From Definition 10 it follows $d_2(g(v_2)) = d_1(g(v_1))$, i.e. $h_2(w) = h_1(w)$. □

In the following let $\mathcal{S}_0$ be a schema derived from a schema $\mathcal{S}$ by omitting all IsA relations.

Lemma 34 *Let C be a value-representable class in $\mathcal{S}$ such that all its superclasses and subclasses $D_1 \dots D_n$ are also value-representable. Then canonical update operations exist on C in $\mathcal{S}$ iff they exist on C and all D_i in $\mathcal{S}_0$.*

Proof. By Theorem 22 the value-representation type V_C is the result of Algorithm 15, hence V_C does not depend on the inclusion constraints of $\mathcal{S}$. Then we have

$$I :: ID \leftarrow insert'_C(V :: V_C) ==$$
$$I \leftarrow insert'_{D_1}(h_1(V)); \dots ; I \leftarrow insert'_{D_n}(h_n(V)); I \leftarrow insert^0_C(V)$$

where $h_i : V_C \rightarrow V_{D_i}$ is the function of Lemma 33 and $insert^0_C$ denotes a canonical insert on C in $\mathcal{S}_0$. Hence in this case the result for the *insert* follows by structural induction on the IsA-hierarchy.

If the subtype function g required in Lemma 33 does not exist for some superclass D then simply add V_D to the input type. We omit the details for this case.

The arguments for *delete* and *update* are analogous. The value-representability of subclasses is required for the update case. □

From now on we use a global operation *NewId* that produces a fresh identifier $I :: ID$. This can be represented as a method using projection.

Lemma 35 *Let C be a value-representable class in $\mathcal{S}_0$. Then there exist unique quasi-canonical update operations on C.*

Proof. Let $r_i : C_i$ $(i = 1 \dots n)$ denote the references in the structure expression of C. If V be a value of type V_C, then there exist values $V_{i,j} :: V_{C_i}$ $(i = 1 \dots n, j = 1 \dots k_i)$ occurring in V. Let $\bar{V} = \{V_{i,j}/J_{i,j} \mid i = 1 \dots n, j = 1 \dots k_i\}.V$ denote the value of type T_C that results from replacing each $V_{i,j}$ by some $J_{i,j} :: ID$. Moreover, for $I :: ID$ let

$$V^{(I)}_{i,j} = \begin{cases} \{V/I\}.V_{i,j} & \text{if } V \text{ occurs in } V_{i,j} \\ V_{i,j} & \text{else} \end{cases}$$

Then the canonical insert operation can be defined as follows:

$$\begin{aligned}
& I :: ID \leftarrow insert'_C(V :: \bar{V}_C) == \\
& \quad \exists I' :: ID,\, V' :: T_C.\, (Pair(I', V') \in C \wedge c(V') = V) \;\rightarrow\; I := I' \\
& \boxtimes\, \exists V' :: T_C.\, V = V' \;\rightarrow\; I \leftarrow NewId;\, x_C := x_C \cup \{(I, V)\} \\
& \boxtimes\, I \leftarrow NewId;\, J_{1,1} \leftarrow insert'_{C_1}(V^{(I)}_{1,1});\, \dots;\, J_{n,k_n} \leftarrow insert'_{C_n}(V^{(I)}_{n,k_n}); \\
& \qquad x_C := x_C \cup \{(I, \bar{V})\}
\end{aligned}$$

It remains to show that this operation is indeed canonical. Apply the method to some instance $\mathcal{D}$. If there already exists some $o = (I', V')$ in C with $c(V') = V$, the result is $\mathcal{D}' = \mathcal{D}$ and the requirements of Definition 30 are trivially satisfied. Otherwise let $o = (I, \bar{V})$. If $\bar{\mathcal{D}}$ is an instance with $\mathcal{D} \subseteq \bar{\mathcal{D}}$ and $o \in \bar{\mathcal{D}}$, we have $J_{i,j} \in dom(C_i)$ for all $i = 1 \dots n$, $j = 1 \dots k_i$, since $\bar{\mathcal{D}}$ satisfies the referential constraints. Hence $\bar{\mathcal{D}}$ contains the distinguished objects corresponding to the involved quasi-canonical operations $insert'_{C_i}$. By induction on the length of call-sequences $\mathcal{D}_{i,j} \subseteq \bar{\mathcal{D}}$ for all $i = 1 \dots n$, $j = 1 \dots k_i$, where $\mathcal{D}_{i,j}$ is the result of $J_{i,j} \leftarrow insert'_{C_i}(V^{(I)}_{i,j})$. Hence $\mathcal{D}' = \bigcup_{i,j} \mathcal{D}_{i,j} \cup \{o\} \subseteq \bar{\mathcal{D}}$. The uniqueness follows from the uniqueness of V_C.

The definitions and proofs for *delete* and *update* are analogous. □

Theorem 36 *Let C be a value-representable class in a schema S such that all its super- and subclasses are also value-representable. Then there exist unique generic update operations on C.*

Proof. By Lemma 31 and Lemma 34 it is sufficient to show the existence of canonical update operations on C and all its super- and subclasses in the schema S_0. This follows from Lemma 35. □

In [50] it has been shown, how linguistic reflection [56] can be exploited to generate the generic update operations for value-representable classes in an OODM schema.

6 The Consistency Problem

In general a database may be considered as a triplet $(\mathcal{S}, \mathcal{O}, \mathcal{C})$, where $\mathcal{S}$ defines a structure, $\mathcal{O}$ denotes a collection of state changing operations and $\mathcal{C}$ is a set of constraints. Then the *consistency problem* is to guarantee that each specified operation $o \in \mathcal{O}$ will never violate any constraint $\mathcal{I} \in \mathcal{C}$. *Integrity enforcement* aims at the derivation of a new set $\mathcal{O}'$ with $|\ O'\ | = |\ O\ |$ of operations such that $(\mathcal{S}, \mathcal{O}', \mathcal{C})$ satisfies this property.

Suppose we are given a database schema $\mathcal{S}$ and a static integrity constraint $\mathcal{I}$ on that schema. Regard $\mathcal{I}$ as a logical formula defined on $\mathcal{S}$. Consistency requires that only those instances $\mathcal{D}$ of $\mathcal{S}$ are allowed that satisfy $\mathcal{I}$. Call the set of such instances $sat(\mathcal{S}, \mathcal{I})$. Each transaction is a *database transformation*. Such a database transformation $\mathcal{T}$ takes an arbitrary instance $\mathcal{D}$ and possibly some input values $v_1, \dots, v_n$ and produces a new instance $\mathcal{D}'$ and possibly some output values $v'_1, \dots, v'_m$. $\mathcal{T}$ is *consistent* with respect to $\mathcal{I}$ iff for each $\mathcal{D} \in sat(\mathcal{S}, \mathcal{I})$ we also have $\mathcal{D}' \in sat(\mathcal{S}, \mathcal{I})$.

Classically consistency is maintained at run-time by transaction monitors. Whenever an inconsistent instance is produced the transaction that caused the inconsistency will be rolled back. This "everything or nothing" approach has been critized, since it causes enormous run-time overhead for consistency checking and rollback. Moreover, it leaves the burden of writing consistent transactions to the user. In principle the first problem vanishes, if verification techniques are used at design time [44,57,58], whereas the second one still remains.

As an alternative a lot of attention has been paid to integrity enforcement. In most cases the envisioned solution is an active database [18,27,59,64,65], where production rules are used to repair inconsistencies instead of rolling back. Although this is sometimes coupled with design time (or even run-time) analysis of the rules [18,27,33,63], the approach is not always successfull. Moreover, a satisfying theory for rule triggering systems with respect to the integrity enforcement problem is still missing. Therefore, we favour an operational approach [51,48,52,53], which aims at replacing inconsistent database transactions by consistent specializations.

6.1 Greatest Consistent Specializations

In general non-deterministic partial state transitions S as used in our method language can be described by a subset of $\mathcal{D} \times \mathcal{D}_\perp$, where $\mathcal{D}$ denotes the set of possible states and $\mathcal{D}_\perp = \mathcal{D} \cup \{\perp\}$, where $\perp$ is a special symbol used to indicate non-termination. It can be shown [20,41,46,44] that this is equivalent to defining two predicate transformers $wp(S)$ and $wlp(S)$ associated with S satisfying the *pairing condition* $wp(S)(\mathcal{R}) \Leftrightarrow wlp(S)(\mathcal{R}) \wedge wp(S)(true)$ and the *universal conjunctivity* of $wlp(S)$,i.e.

$$wlp(S)(\forall i \in I. \mathcal{R}_i) \Leftrightarrow \forall i \in I. wlp(S)(\mathcal{R}_i) .$$

The predicate transformers assign to some postcondition $\mathcal{R}$ the *weakest (liberal) precondition* of S to establish $\mathcal{R}$. Clearly, pre- and postconditions are X-constraints. Informally these conditions can be characterized as follows:

- $wlp(S)(\mathcal{R})$ characterizes those initial states such that all terminating executions of S will reach a final state characterized by $\mathcal{R}$ provided S is defined in that initial state, and

- $wp(S)(\mathcal{R})$ characterizes those initial states such that all executions of S terminate and will reach a final state characterized by $\mathcal{R}$ provided S is defined.

The use of these predicate transformers for the definition of language semantics is usually called "axiomatic semantics". Based on this consistency and specialization can be formally defined and used for the formal description of the consistency problem. For this purpose we define "extended operations" and therefore need to know for each operation S the set of classes $\mathcal{S}'$ such that S does neither read nor change the class variables x_C with $C \notin \mathcal{S}'$. In this case we call S a $\mathcal{S}'$-operation. We omit the formal definition [41,51].

Definition 37 *Let $\mathcal{S}$ be a schema, $\mathcal{I}$ a constraint and S, T methods defined on $\mathcal{S}_1 \subseteq \mathcal{S}$ and $\mathcal{S}_2 \subseteq \mathcal{S}$ respectively with $\mathcal{S}_1 \subseteq \mathcal{S}_2$.*

1. *S is* consistent *with respect to $\mathcal{I}$ iff $\mathcal{I} \Rightarrow wlp(S)(\mathcal{I})$ holds.*

2. *T* specializes *S iff $wp(S)(true) \Rightarrow wp(T)(true)$ and $wlp(S)(\mathcal{R}) \Rightarrow wlp(T)(\mathcal{R})$ hold for all constraints $\mathcal{R}$ with free variables x_C such that $C \in \mathcal{S}_1$ (denoted $T \sqsubseteq S$).*

Hence the following definition of a *greatest consistent specialization*:

Definition 38 *Let $\mathcal{S}$ be a schema, $\mathcal{I}$ a constraint and S a method defined on $\mathcal{S}_1 \subseteq \mathcal{S}$. A method $S_{\mathcal{I}}$ is a* Greatest Consistent Specialization *(GCS) of S with respect to $\mathcal{I}$ iff*

1. *$S_{\mathcal{I}} \sqsubseteq S$,*

2. *$S_{\mathcal{I}}$ is consistent with respect to $\mathcal{I}$ and*

3. *for each method T satisfying properties (i) and (ii) (instead of $S_{\mathcal{I}}$) we have $T \sqsubseteq S_{\mathcal{I}}$.*

If only properties (i) and (ii) are satisfied, we simply talk of a consistent specialization.

Let us first state the main results from [48].

Theorem 39 *Let $\mathcal{S}$ be a schema, $\mathcal{I}$, $\mathcal{J}$ constraints and S a method defined on $\mathcal{S}_1 \subseteq \mathcal{S}$.*

1. *There exists a greatest consistent specialization $S_{\mathcal{I}}$ of S with respect to $\mathcal{I}$. Moreover, $S_{\mathcal{I}}$ is uniquely determined (up to semantic equivalence) by S and $\mathcal{I}$.*

2. *The GCSs $(S_{\mathcal{I}})_{\mathcal{J}}$ and $S_{(\mathcal{I} \wedge \mathcal{J})}$ coincide on initial states satisfying $\mathcal{I} \wedge \mathcal{J}$.*

The proof of these results heavily uses predicate transformers and is therefore omitted here.

In [51] it has been shown that a GCS—that is in general non-deterministic—can be written as a finite choice of *maximal quasi-deterministic specializations* (MQCSs), where quasi-determinism means determinism up to the selection of some values. In most cases this value selection can be shifted to the input, but the selection of object identifiers should be left to the system.

Next, we formally define quasi-determinism and then present the main result from [51], an algorithm for the computation of MQCSs.

Definition 40 *A method S is called* quasi-deterministic *iff there exist types $T_1, \dots, T_n$ such that S is semantically equivalent to*

$$y_1 :: T_1 \mid \dots y_n :: T_n \mid S' \ ,$$

where S' is a deterministic method.

Algorithm 41 In: *An X-operation S and constraints $I_1, \dots, I_n$ defined on extensions $Y_1, \dots, Y_n$ of X.*

Let ℓ be the list of the constraints. As long as $\ell \neq nil$ proceed as follows:

1. *Set $S'_I = S$.*
2. *Choose and remove one constraint I_i from ℓ.*
3. *Check whether S'_I is I_i-reduced. If not, stop with no result, otherwise continue.*
4. *Make S'_I $\boxtimes$-free by replacing each occurring $S_1 \boxtimes S_2$ by $S_1 \square wlp(S_1)(false) \rightarrow S_2$.*
5. *Replace each basic assignment in S'_I by some (subsumption-free) MQCS with respect to I_i.*
6. *Compute $P(S_I)$ as*

$$P(\bar{S_I}) \equiv \{z_1/x_1, \dots, z_n/x_n\}.wlp(\{x_1/z_1, \dots \\ \dots, x_n/z_n\}.\bar{S_I})(\neg wlp(S)(z_1 \neq x_1 \vee \dots \vee z_n \neq x_n)) \ ,$$

   where the x_i are the class variables occurring in I or in S and the z_i are used as a disjoint copy of these.

7. *Set $S = P(S_I) \rightarrow S'_I$.*

Set $S'_I = S$.

Out: *An operation $I \rightarrow S'_I$, where S'_I is a (subsumption-free) MQCS of the original S with respect to the conjunction I of the constraints.*

□

An extension of the GCS algorithm to compute all (subsumption-free) MQCSs is easy.

It has been shown in [51] that Algorithm 41 is correct. However, it depends on checking a very technical condition, I-reducedness. We omit this condition here.

6.2 Enforcing Integrity in the OODM

Since Algorithm 41 allows integrity enforcement to be reduced to the case of assignments, we may restrict ourselves to the case of a single explicit constraint in addition to the trivial uniqueness constraints that are required to assure value-representability and that are used to construct *generic update operations*. In the following we describe MQCSs with respect to the constraints introduced in Definition 5.

6.2.1 Inclusion Constraints.

Let I be an inclusion constraint on C_1, C_2 defined via $c_i : T_{C_i} \rightarrow T$ $(i = 1, 2)$. Then each insertion into C_1 requires an additional insertion into C_2 whereas a deletion on C_2 requires a deletion on C_1. Update on one of the C_i requires an additional update on the other class.

Let us first concentrate on the insert-operation on C_1 (for an insert on C_2 there is nothing to do). Insertion into C_1 requires an input-value of type V_{C_1}; an additional insert on C_2 then requires an input-value of type V_{C_2}. However, these input-values are not independent, because the corresponding values of type T_{C_1} and T_{C_2} must satisfy the general inclusion constraint. Therefore we first show that the constraint can be "lifted" to a constraint on the value-representation types. Note that this is similar to the handling of IsA-constraints in Lemma 33.

Lemma 42 *Let C_1, C_2 be classes, $c_i : T_{C_i} \rightarrow T$ functions and let V_{C_i} be the value-representation type of C_i $(i = 1, 2)$. Then there exist functions $f_i : V_{C_i} \rightarrow T$ such that for all database instances $\mathcal{D}$*

$$f_1(d_1^{\mathcal{D}}(v_1)) = f_2(d_2^{\mathcal{D}}(v_2)) \Leftrightarrow c_1(v_1) = c_2(v_2) \tag{13}$$

for all $v_i \in codom(\mathcal{D}(x_{C_i}))$ $(i = 1, 2)$ holds. Here $d_i^{\mathcal{D}} : T_{C_i} \rightarrow V_{C_i}$ denotes the function used in the uniqueness constraint on C_i with respect to $\mathcal{D}$.

Proof. Due to Definition 10 we may define $f_i = c_i \circ (d_i^{\mathcal{D}})^{-1}$ on $c_i(codom(\mathcal{D}(x_{C_i})))$ $(i = 1, 2)$.

Then we have to show that this definition is independent of the instance $\mathcal{D}$. Suppose $\mathcal{D}_1$, $\mathcal{D}_2$ are two different instances. Then there exists a permutation π on ID such that $d_i^{\mathcal{D}_2} = d_i^{\mathcal{D}_1} \circ \pi$, where π is extended to T_{C_i}. Then

$$c_i \circ (d_i^{\mathcal{D}_2})^{-1} = c_i \circ \pi^{-1} \circ (d_i^{\mathcal{D}_1})^{-1} = \pi^{-1} \circ c_i \circ (d_i^{\mathcal{D}_1})^{-1} ,$$

since c_i permutes with π^{-1}. Then the stated equality follows. □

Now let $V_{C_1,C_2} = V_{C_1} \times V_{C_2}$ and define the new insert-operation on C_1 by $(insert_{C_1})_I((v_1, v_2) :: V_{C_1,C_2}) ==$

$$f_1(v_1) = f_2(v_2) \rightarrow insert_{C_1}(v_1) \; ; \; insert_{C_2}(v_2) \; , \tag{14}$$

where the f_i are the functions of Lemma 42. Note there there is no need to require $C_1 \neq C_2$. Delete- and update-operations can be defined analogously.

6.2.2 Functional and Uniqueness Constraints.

Now let I be a functional constraint on C defined via $c^1 : T_C \to T_1$ and $c^2 : T_C \to T_2$. In this case nothing is required for the delete operation whereas for inserts (and updates) we have to add a postcondition. Moreover, let $c^{\mathcal{D}} : T_C \to V_C$ denote the function associated with the value-representability of C and the database instance $\mathcal{D}$ and let all other notations be as before. Let us again concentrate on the insert-operation. Let $insert'_C$ denote the *canonical insert* on C. Then we define

$$
\begin{aligned}
&(insert_C)_I(V :: V_C) == \\
&\quad I :: ID \mid I \leftarrow insert'_C(V) ; \\
&\quad V' :: T_C \mid (I, V') \in x_C \to \\
&\quad\ (\forall J :: ID, W :: T_C . \, ((J, W) \in x_C \\
&\quad\ \wedge c^1(W) = c^1(V') \Rightarrow c^2(W) = c^2(V')) \ \to \ skip \ . \qquad (15)
\end{aligned}
$$

Note that in this case there is no change of input-type. For delete- and update-operations we have analogous definitions.

A uniqueness constraint defined via $c^1 : T_C \to T_1$ is equivalent to a functional constraint defined via c^1 and $c^2 = id : T_C \to T_C$ plus the trivial uniqueness constraint. Since trivial uniqueness constraints are already enforced by the canonical update operations, there is no need to handle separately arbitrary uniqueness constraints.

6.2.3 Exclusion Constraints.

The handling of exclusion constraints is analogous to the handling of inclusion constraints. This means that an insert (update) on one class may cause a delete on the other, whereas delete-operations remain unchanged.

We concentrate again on the insert-operation. Let I be an exclusion constraint on C_1 and C_2 defined via $c_i : T_{C_i} \to T$ $(i = 1, 2)$. Let $f_i : V_{C_i} \to T$ denote the functions from Lemma 42. Then we define a new insert-operation on C_1 by

$$
\begin{aligned}
&(insert_{C_1})_I(V :: V_{C_1}) == \\
&\quad insert_{C_1}(V) ; \\
&\quad \mu S. \, ((I :: ID \mid V' :: T_{C_2} \mid (I, V') \in x_{C_2} \\
&\qquad \wedge c^2(V') = f_1(V) \ \to \ delete_{C_2}(V') \ ; \ S \ ) \boxtimes skip \ ) \ . \qquad (16)
\end{aligned}
$$

For delete- and update-operations an analogous result holds.

Theorem 43 *The methods S_I in (14), (15) and (16) are MQCSs of generic insert-methods with respect to inclusion, functional and exclusion constraints respectively.*

The proof involves detailed use of predicate transformers and is therefore omitted here [48,49]. Analogous results hold for *delete* and *update*.

7 Conclusion

In this paper we describe first results concerning the formal foundations of object oriented database concepts. For this purpose we introduced a formal object oriented datamodel (OODM) with the following characteristics.

- Objects are considered to be abstractions of real world entities, hence they have an immutable identity. This identity is encoded by abstract identifiers that are assumed to form some type ID. This identifier concept eases the modelling of shared data and cyclic references, however, it does not relieve us from the problem to provide unique identification mechanisms for objects in a database.

- In our approach there is not only one value of a given type that is associated with an object. In contrast we allow several values of possibly different types to belong to an object, and even this collection of types may change.

- Classes are used to structure objects. At each time a class corresponds to a collection of objects with values of the same type and references to objects in a fixed set of classes. Inheritance is based on IsA relations that express an inclusion at each time of the sets of objects. Moreover, referential integrity is supported.

- We associate with each class a collection of methods. Methods are specified by guarded commands, hence the method language is computationally complete. In order to allow the handling of identifiers that are always hidden from the user as well as user-accessible transactions a hiding operator on methods is introduced. Generic update operations, i.e. insert, delete and update on a class are assumed to be automatically derived whenever this is possible.

- We associate integrity constraints to schemata. Certain kinds of such constraints can be obtained by generalizing corresponding constraints in the relational model. We assume that methods are automatically changed in order to enforce integrity.

On this basis of this formal OODM we study the problems of identification, genericity and integrity. We show that the unique identification of objects in a class requires the class to be value-representable.

An advantage of database systems is to provide generic update operations. We show that the unique existence of such generic methods requires also value-representability. However, in this case referential and inclusion integrity can be enforced automatically. This result can be generalized with respect to distinguished classes of user-defined integrity constraints. Given some arbitrary method S and some constraint I there exists a *greatest consistent specialization* (GCS) S_I of S with respect to I. Such a GCS behaves nice in that it is compatible with the conjunction of constraints. For the GCS construction of a user-defined transaction we apply the GCS algorithm developped in [48,51,52,53].

This work on mathematical foundations of OODB concepts is not yet completed. A lot of problems are still left open and are the matter of current investigations and future research.

- In our approach classes are sets. What are other bulk types? Does it make sense to abstract from classes in this way?

- The problem of updatable views is still open.

- Our approach to genericity only handles the worst case expressed by the value representation type. We assume that polymorphism will help to generalize our results to the general case. Moreover, we must integrate communication aspects at least with respect to the user.

- The usual axiomatic semantics for guarded commands abstracts from an execution model. All results are true for semantic equivalence classes. However, we also need optimization, especially with respect to the derived GCSs.
- We only presented a formal OODM without looking into methodological aspects such as the characterization of good designs.

We express the hope that others will also contribute to solve open problems in OODB foundation or in the implementation of more sophisticated object oriented database languages on a sound mathematical basis.

Acknowledgement

We would like to thank Catriel Beeri, Joachim W. Schmidt, and Ingrid Wetzel for many stimulating discussions especially concerning object identification. We also want to thank David Stemple and Kasimierz Subieta for questioning the theme from an engineering point of view.

References

[1] S. Abiteboul: *Towards a deductive object-oriented database language*, Data & Knowledge Engineering, vol. 5, 1990, pp. 263 – 287

[2] S. Abiteboul, R. Hull: *IFO: A Formal Semantic Database Model*, ACM ToDS, vol. 12 (4), December 1987, pp. 525 – 565

[3] S. Abiteboul, P. Kanellakis: *Object Identity as a Query Language Primitive*, in Proc. SIGMOD, Portland Oregon, 1989, pp. 159 – 173

[4] A. Albano, G. Ghelli, R. Orsini: *Types for Databases: The Galileo Experience*, in Type Systems and Database Programming Languages, University of St. Andrews, Dept. of Mathematical and Computational Sciences, Research Report CS/90/3, 27 – 37

[5] A. Albano, A. Dearle, G. Ghelli, C. Marlin, R. Morrison, R. Orsini, D. Stemple: *A Framework for Comparing Type Systems for Database Programming Languages*, in Type Systems and Database Programming Languages, University of St. Andrews, Dept. of Mathematical and Computational Sciences, Research Report CS/90/3, 1990

[6] A. Albano, G. Ghelli, R. Orsini: *Objects and Classes for a Database Programming Language*, FIDE technical report 91/16, 1991

[7] A. Albano, G. Ghelli, R. Orsini: *A Relationship Mechanism for a Strongly Typed Object-Oriented Database Programming Language*, in A. Sernadas (Ed.): *Proc. VLDB 91*, Barcelona 1991

[8] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik: *The Object-Oriented Database System Manifesto*, Proc. 1st DOOD, Kyoto 1989

[9] F. Bancilhon, G. Barbedette, V. Benzaken, C. Delobel, S. Gamerman, C. Lécluse, P. Pfeffer, P. Richard, F. Velez: *The Design and Implementation of O_2, an Object-Oriented Database System*, Proc. of the ooDBS II workshop, Bad Münster, FRG, September 1988

[10] C. Beeri: *Formal Models for Object-Oriented Databases*, Proc. 1st DOOD 1989, pp. 370 – 395

[11] C. Beeri: *A formal approach to object-oriented databases*, Data and Knowledge Engineering, vol. 5 (4), 1990, pp. 353 – 382

[12] C. Beeri, Y. Kornatzky: *Algebraic Optimization of Object-Oriented Query Languages*, in S. Abiteboul, P. C. Kanellakis (Eds.): Proc. ICDT '90, Springer LNCS 470, pp. 72 – 88

[13] C. Beeri: *New Data Models and Languages - the Challange* in Proc. PODS '92

[14] L. Cardelli, P. Wegner: *On Understanding Types, Data Abstraction and Polymorphism*, ACM Computing Suerveys 17,4, pp 471 – 522

[15] L. Cardelli: *Typeful Programming*, Digital Systems Research Center Reports 45, DEC SRC Palo Alto, May 1989

[16] M. Carey, D. DeWitt, S. Vandenberg: *A Data Model and Query Language for EXODUS*, Proc. ACM SIGMOD 88

[17] M. Caruso, E. Sciore: *The VISION Object-Oriented Database Management System*, Proc. of the Workshop on Database Programming Languages, Roscoff, France, September 1987

[18] S. Ceri, J. Widom: *Deriving Production Rules for Constraint Maintenance*, Proc. 16th Conf. on VLDB, Brisbane (Australia), August 1990, pp. 566 – 577

[19] A. Dearle, R. Connor, F. Brown, R. Morrison: *Napier88 - A Database Programming Language?*, in Type Systems and Database Programming Languages, University of St. Andrews, Dept. of Mathematical and Computational Sciences, Research Report CS/90/3, 10 – 26

[20] E. W. Dijkstra, C. S. Scholten: *Predicate Calculus and Program Semantics*, Springer-Verlag, 1989

[21] H.-D. Ehrich, M. Gogolla, U. Lipeck: *Algebraische Spezifikation abstrakter Datentypen*, Teubner-Verlag, 1989

[22] H.-D. Ehrich, A. Sernadas: *Fundamental Object Concepts and Constructors*, in G. Saake, A. Sernadas (Eds.): Information Systems – Correctness and Reusability, TU Braunschweig, Informatik Berichte 91-03, 1991

[23] H. Ehrig, B. Mahr: *Fundamentals of Algebraic Specification*, vol.1, Springer 1985

[24] L. Fegaras, T. Sheard, D. Stemple: *The ADABTPL Type System*, in Type Systems and Database Programming Languages, University of St. Andrews, Dept. of Mathematical and Computational Sciences, Research Report CS/90/3, 45 – 56

[25] L. Fegaras, T. Sheard, D. Stemple: *Uniform Traversal Combinators: Definition, Use and Properties*, University of Massachusetts, 1992

[26] D. Fishman, D. Beech, H. Cate, E. Chow et al.: *IRIS: An Object-Oriented Database Management System*, ACM ToIS, vol. 5(1), January 1987

[27] P. Fraternali, S. Paraboschi, L. Tanca: *Automatic Rule Generation for Constraint Enforcement in Active Databases*, in U. Lipeck (Ed.): Proc. 4th Int. Workshop on Foundations of Models and Languages for Data and Objects "MODELLING DATABASE DYNAMICS", Volkse (Germany), October 19-22, 1992

[28] G. Gottlob, G. Kappel, M. Schrefl: *Semantics of Object-Oriented Data Models - The Evolving Algebra Approach*, in J. W. Schmidt, A. A. Stognij (Eds.): Proc. Next Generation Information Systems Technology, Springer LNCS, vol. 504, 1991

[29] M. Hammer, D. McLeod: *Database Description with SDM: A Semantic Database Model*, J. ACM, vol. 31 (3), 1984, pp. 351 – 386

[30] A. Heuer, P. Sander: *Classifying Object-Oriented Results in a Class/Type Lattice*, in B. Thalheim et al. (Ed.): *Proceedings MFDBS 91*, Springer LNCS 495, pp. 14 – 28

[31] R. Hull, R. King: *Semantic Database Modeling: Survey, Applications and Research Issues*, ACM Computing Surveys, vol. 19(3), September 1987

[32] R. Hull, M. Yoshikawa: *ILOG: Declarative Creation and Manipulation of Object Identifiers*, in Proc. 16th VLDB, Brisbane (Australia), 1990, pp. 455 – 467

[33] A. P. Karadimce, S. D. Urban, *Diagnosing Anomalous Rule Behaviour in Databases with Integrity Maintenance Production Rules*, in Proc. 3rd Int. Workshop on Foundations of Models and Languages for Data and Objects, Aigen (Austria), September 1991, pp. 77 – 102

[34] S. Khoshafian, G. Copeland: *Object Identity*, Proc. 1st Int. Conf. on OOPSLA, Portland, Oregon, 1986

[35] M. Kifer, J. Wu: *A Logic for Object-Oriented Logic Programming (Maier's O-Logic Revisited)*, in PODS'89, pp. 379 – 393

[36] W. Kim, N. Ballou, J. Banerjee, H. T. Chou, J. Garza, D. Woelk: *Integrating an Object-Oriented Programming System with a Database System*, in Proc. OOPSLA 1988

[37] D. Maier, J. Stein, A. Ottis, A. Purdy: *Development of an Object-Oriented DBMS*, OOPSLA, September 1986

[38] F. Matthes, J. W. Schmidt: Bulk Types – Add-On or Built-In?, in Proc. DBPL III, Nafplion 1991

[39] J. Mylopoulos, P. A. Bernstein, H. K. T. Wong: *A Language Facility for Designing Interactive Database-Intensive Applications*, ACM ToDS, vol. 5 (2), April 1980, pp. 185 – 207

[40] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis: *Telos: Representing Knowledge About Information Systems*, ACM ToIS, vol. 8 (4), October 1990 pp. 325 – 362

[41] G. Nelson: *A Generalization of Dijkstra's Calculus*, ACM TOPLAS, vol. 11 (4), October 1989, pp. 517 – 561

[42] A. Ohori: *Representing Object Identity in a Pure Functional Language*, Proc. ICDT 90, Springer LNCS, pp. 41 – 55

[43] G. Saake, R. Jungclaus: *Specification of Database Applications in the TROLL Language*, in Proc. Int. Workshop on the Specification of Database Systems, Glasgow, 1991

[44] K.-D. Schewe, I. Wetzel, J. W. Schmidt: *Towards a Structured Specification Language for Database Applications*, in D. Harper, M. Norrie (Eds.): Proc. Int. Workshop on the Specification of Database Systems, Springer WICS, 1991, pp. 255 – 274 (an extended version appeared as FIDE technical report 1991/30, October 1991)

[45] K.-D. Schewe, B. Thalheim, I. Wetzel, J. W. Schmidt: *Extensible Safe Object-Oriented Design of Database Applications*, University of Rostock, Preprint CS-09-91, September 1991

[46] K.-D. Schewe: *Spezifikation datenintensiver Anwendungssysteme* (in German), lecture manuscript, University of Hamburg, Winter 1991/92

[47] K.-D. Schewe, J. W. Schmidt, I. Wetzel: *Identification, Genericity and Consistency in Object-Oriented Databases*, in J. Biskup, R. Hull (Eds.): Proc. ICDT '92, Springer LNCS 646, pp. 341-356

[48] K.-D. Schewe, B. Thalheim, J. W. Schmidt, I. Wetzel: *Integrity Enforcement in Object-Oriented Databases*, in U. Lipeck, B. Thalheim (Eds.): Proc. 4th Int. Workshop on Foundations of Models and Languages for Data and Objects "MODELLING DATABASE DYNAMICS", Volkse (Germany), October 19-22, 1992

[49] K.-D. Schewe, B. Thalheim, I. Wetzel: *Foundations of Object Oriented Database Concepts*, University of Hamburg, Report FBI-HH-B-157/92, October 1992

[50] K.-D. Schewe, J. W. Schmidt, D. Stemple, B. Thalheim, I. Wetzel: *A Reflective Approach to Method Generation in Object Oriented Databases*, University of Rostock, Rostocker Informatik Berichte, no. 14, 1992

[51] K.-D. Schewe, B. Thalheim: *Computing Consistent Transactions*, University of Rostock, Preprint CS-08-92, December 1992

[52] K.-D. Schewe, B. Thalheim, I. Wetzel: *Integrity Preserving Updates in Object Oriented Databases*, in M. Orlowska, M. Papazoglou (Eds.) : Proc. 4th Australian Database Conference, Brisbane, February 1993, World Scientific, pp. 171-185

[53] K.-D. Schewe, B. Thalheim: *Exceeding the Limits of Rule Triggering Systems to Achieve Consistent Transactions*, submitted for publication

[54] M. H. Scholl, H.-J. Schek: *A Relational Object Model*, in Proc. ICDT 90, Springer LNCS, pp. 89 – 105

[55] G. M. Shaw, S. B. Zdonik: *An Object-Oriented Query-Algebra*, IEEE Data Engineering, vol. 12 (3), 1989, pp. 29 – 36

[56] D. Stemple, T. Sheard, L. Fegaras: *Reflection: A Bridge from Programming to Database Languages*, in Proc. HICSS '92

[57] D. Stemple, S. Mazumdar, T. Sheard: *On the Modes and Meaning of Feedback to Transaction Designer*, in Proc. SIGMOD 1987, pp. 375 – 386

[58] D. Stemple, T. Sheard: *Automatic Verification of Database Transaction Safety*, ACM ToDS vol. 14 (3), September 1989

[59] M. Stonebraker, A. Juingran, J. Goh, S. Potaminos: *On Rules, Procedures, Caching and Views in Database Systems*, in Proc. SIDMOD 1990, pp. 281 – 290

[60] S. Y. W. Su: *SAM*: A Semantic Association Model for Corporate and Scientific-Statistical Databases*, Inf. Sci., vol. 29, 1983, pp. 151 – 199

[61] B. Thalheim: *Dependencies in Relational Databases*, Teubner Leipzig, 1991

[62] B. Thalheim: *The Higher-Order Entity-Relationship Model*, in J. W. Schmidt, A. A. Stognij (Eds.): Proc. Next Generation Information Systems Technology, Springer LNCS, vol. 504, 1991

[63] S. D. Urban, L. Delcambre: *Constraint Analysis: a Design Process for Specifying Operations on Objects*, IEEE Trans. on Knowledge and Data Engineering, vol. 2 (4), December 1990

[64] J. Widom, S. J. Finkelstein: *Set-oriented Production Rules in Relational Database Systems*, in Proc. SIGMOD 1990, pp. 259 – 270

[65] Y. Zhou, M. Hsu: *A Theory for Rule Triggering Systems*, in Proc. EDBT '90, Springer LNCS 416, pp. 407 – 421

(Received April 7, 1993)

Acta Cybernetica, Vol. 11, No. 1-2, Szeged, 1993

On the characterization of the integers: The hidden function problem revisited

R. Berghammer*

Abstract

In this paper the hidden function problem studied so far only for equational (e.g., in [9] and [11]) or conditional equational (e.g., in [3]) algebraic specifications is considered for arbitrary first-order theories. It is shown that a unique characterization of the integers with zero, successor and predecessor as term-generated model of a finite first-order theory needs at least one hidden function or relation.
Keywords: Hidden function problem, algebraic specifications, first-order theories.

1 Introduction

In mathematical logic, a structure for a first-order language is said to be a model for a set T of sentences over the same language, if each sentence of T holds in it. The algebraic specification approach of computer science uses a restricted definition. Here it is often additionally demanded that each element of the carrier sets can finitely be "described" by a closed term, i.e., that the model of the specification is term-generated (see e.g., [1], [12], [6], or [13]). The main reason for the restriction to term-generated models of specifications is the necessity of finite descriptions of algorithms. As an essential advantage one obtains the proof principle of term induction. Furthermore, by using only term-generated models one is able to extend the expressiveness of first-order theories (resp. algebraic specifications).

In this paper we deal with the question, whether and how the structure $Z := (\mathbb{Z}, 0, \text{succ}, \text{pred})$ can – up to isomorphism – be characterized as the only term-generated model of a set of first-order sentences over a first-order language with symbols for 0, the successor function $\text{succ}(u) := u+1$, and the predecessor function $\text{pred}(u) := u-1$. First, we give a positive answer using an infinite set of sentences. Then we show, and this is the main result of the paper, that there is no finite set of first-order sentences with the same property. Finally, we extend the language by a symbol for the "usual" ordering relation on the integers $\mathbb{Z}$ and present a finite set of sentences, which has the structure $\overline{Z} := (\mathbb{Z}, 0, \text{succ}, \text{pred}, \leq)$ as – up to isomorphism – only term-generated model.

The relation $\leq$ simplifies the specification of the constant and operations of interest 0, succ, and pred. In the terminology of algebraic specifications it is called

*Fakultät für Informatik, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, D-85579 Neubiberg

a "hidden function", since the way to specify Z is structured by first specifying $\overline{Z}$ and then to forget or hide the auxiliary relation $\leq$.

Strictly speaking, $\leq$ is a hidden relation. The term "hidden function" (which we will use in the remainder of the paper, too) results from the fact that the algebraic specification approach considers relations as functions to the truth values.

Given a class C of first-order formulae and a semantic mechanism S which determines the meaning of a specification, the so-called hidden function problem for C and S asks whether the use of hidden functions extends the expressiveness of specifications. All the known examples deal with the following question: Is there a structure that fails to possess a unique characterization (this notion depends on S) using finite subsets of C only, but the same is not true if auxiliary functions may be used? In the case of C being the class of universally quantified equations and S being initial algebra semantics, a solution - the first example which requires the use of a hidden function – can be found in [9]. This paper contains no formal proof, but based on Majster's example in [11] a simple structure, called "toy stack", is constructed and carefully proved that it cannot be specified using initial algebra semantics and finitely many equations unless hidden functions are permitted. This proof is mainly based on regular sets and their properties. Independently of [11], in [2] another solution of the hidden function problem for equational specifications and initial algebra semantics is given. It is shown that the structure $N := (\mathbb{N}, 0,$ succ, sqr), where succ is again the successor function and $\mathrm{sqr}(u) := u^2$, does not possess a finitary equational specification without the use of hidden machinery. The (rather complicated) proof can also be found in [3]. Obviously, N admits a very natural finite equational specification involving addition and multiplication as auxiliary functions. Using the so-called sparsity property of predicates on natural numbers, in the same paper [3] the hidden function problem is also solved for conditional equational specifications and initial algebra semantics.

Our examples Z and $\overline{Z}$ solve also a hidden function problem for certain C and S. In comparison to the papers just mentioned, we do not restrict the class of formulae and consider all term-generated models. This means that C is the class of all first-order formulae and that a structure M is (uniquely) characterized by a set T of sentences under S if and only if M is a term-generated model of T and all these models are isomorphic. Furthermore, we use proof principles from "classical" model theory, viz. the use of the compactness theorem and elimination of quantifiers.

2 Preliminaries

Throughout this paper we use first-order logic with the equality symbol $\approx$ as a logical symbol. In this section, we briefly recall some basic definitions of first-order logic. Further details can be found in, for instance, [7] or [10].

Assume L to be a first-order language. A *structure* M for L (also called L-structure) consists of a non-empty carrier set $|M|$, an n-ary function $f_M : |M|^n \rightarrow |M|$ for every n-place function symbol f, and an n-ary predicate $p_M : |M|^n \rightarrow \mathbb{B}$ for every n-place predicate symbol p, where $\mathbb{B}$ denotes the set $\{0,1\}$ representing truth values. If $n = 0$, then f_M is an element of $|M|$ and p_M is a truth value.

Assume M and N to be two structures for the same first-order language. A bijective function $\Phi : |M| \rightarrow |N|$ is said to be an *isomorphism* from M to N, if

$$\Phi(f_M(u_1,\ldots,u_n)) = f_N(\Phi(u_1),\ldots,\Phi(u_n))$$

for all n-place function symbols f and all $u_1,\ldots,u_n \in |M|$ and

$$p_M(u_1,\ldots,u_n) = 1 \Leftrightarrow p_N(\Phi(u_1),\ldots,\Phi(u_n)) = 1$$

for all n-place predicate symbols p and all $u_1,\ldots,u_n \in |M|$. If there is an isomorphism from M to N, then we say that M and N are *isomorphic*.

Let M be a structure for a first-order language L and $\Psi : V \to |M|$ be an assignment for the variables $x \in V$ with values from $|M|$. Furthermore, let t be a term and A be a formula built up over L. By t^M_Ψ we denote the value of t in M under Ψ; by $M \models A[\Psi]$ we denote that A holds in M under Ψ. Both notations are inductively defined as usual. In particular, we have $M \models t_1 \approx t_2[\Psi]$ if and only if $t^M_{1\Psi} = t^M_{2\Psi}$. If both t and A are closed, then t^M_Ψ as well as $M \models A[\Psi]$ do not depend on the assignment Ψ. Therefore, in this case we use the notations t^M and $M \models A$ instead. The notation $M \models A$ is also used to indicate that $M \models A[\Psi]$ for every assignment Ψ.

Let L be a first-order language. A set T of sentences (i.e., closed formulae) built up over L is called a *theory* over L. A structure M for L is said to be a *model* of T, if $M \models A$ for all sentences $A \in T$. In addition, M is called *term-generated,* if for every element $u \in |M|$ there exists a closed term t (also built up over L) such that $u = t^M$.

3 An infinite characterization of the integers without hidden functions

In the following, we give a characterization of the integers with 0, succ, and pred as – up to isomorphism – only term-generated model of an infinite first-order theory. This result will also be used in the next section.

Let L_Z be the first-order language consisting of a 0-place function symbol (constant symbol) z and two 1-place function symbols s, p, and let T_Z denote the following infinite theory:

$$
\begin{array}{rll}
(1) & \forall x(s(p(x)) \approx x) & \\
(2) & \forall x(p(s(x)) \approx x) & \\
(3.1) & \forall x(\neg(s(x) \approx x)) & \\
(3.2) & \forall x(\neg(s(s(x)) \approx x)) & \\
 & \cdots\cdots & \\
(3.\mathrm{n}) & \forall x(\neg(s(s(\ldots s(s(x))\ldots)) \approx x) & \text{(exactly n occurrences of s).} \\
 & \cdots\cdots &
\end{array}
$$

Obviously, we have: The structure $Z := (\mathbb{Z}, 0, \text{succ}, \text{pred})$ is a term-generated model of T_Z. We call Z the *standard model* of the theory T_Z. In the following, we show that it is – up to isomorphism – the only term-generated model of T_Z. To this end, we assume for the rest of this Section 3 an arbitrarily chosen (but fixed) term-generated model $M := (|M|, z_M, s_M, p_M)$ of T_Z and construct an isomorphism from M to the standard model.

Define s_M^n (resp. p_M^n) as n^{th} power of s_M (resp. p_M). Fundamental for the construction of the just mentioned isomorphism is the following representation of the elements of $|M|$.

Lemma 3.1 *Let $u \in |M|$. Then there exists exactly one natural number $n \in \mathbb{N}$ such that $u = s_M^n(z_M)$ or $u = p_M^n(z_M)$.*

Proof. a) In the first step we prove the existence of the number n.

As the model M is term-generated, for all $u \in |M|$ there exists a closed term t built up over L_Z such that $t^M = u$. Thus, it suffices to show that for all closed terms t built up over L_Z there exists a natural number $n \in \mathbb{N}$ such that $t^M = s_M^n(z_M)$ or $t^M = p_M^n(z_M)$. This can be done by term induction.

Induction base: The case of t being the symbol z is trivial; choose $n = 0$.

Induction step: By the induction hypothesis, $t^M = s_M^n(z_M)$ or $t^M = p_M^n(z_M)$. First, suppose $t^M = s_M^n(z_M)$. Then we have

$$s(t)^M = s_M(t^M) = s_M(s_M^n(z_M)) = s_M^{n+1}(z_M).$$

Furthermore, due to the validity of sentence (2) in M,

$$p(t)^M = p_M(t^M) = p_M(s_M^n(z_M)) = p_M(s_M(s_M^{n-1}(z_M))) = s_M^{n-1}(z_M),$$

provided $n > 0$. Finally, in the case $n = 0$ we obtain

$$p(t)^M = p_M(t^M) = p_M(z_M).$$

This shows that also $s(t)^M$ and $p(t)^M$ have the stated representation.

The remaining case $t^M = p_M^n(z_M)$ is handled similarly using the validity of (1) in M.

b) In a second step, now we prove the uniqueness of the representation. To this end, suppose $u = s_M^m(z_M) = s_M^n(z_M)$ and $m \neq n$. W.l.o.g, let $m < n$. Then there exists a positive natural number k fulfilling the equation $m + k = n$. Sentence (2) is true in M. Thus,

$$s_M^m(z_M) = s_M^n(z_M) = s_M^m(s_M^k(z_M)) \Leftrightarrow z_M = s_M^k(z_M).$$

However, $s_M^k(z_M) = z_M$ contradicts the validity of sentence (3.k) in M. In the same manner one deals with the remaining cases. □

With the help of this lemma, we are able to define a function Φ from the carrier set $|M|$ to the integers by

$$\Phi : |M| \to \mathbb{Z} \qquad \Phi(u) := \begin{cases} n & \text{if } u = s_M^n(z_M) \\ -n & \text{if } u = p_M^n(z_M). \end{cases}$$

We then have the following property:

Lemma 3.2 *The function Φ is an isomorphism from the fixed model M to the standard model Z.*

Proof. Bijectivity of Φ is obvious; the inverse Φ^{-1} from the integers to $|M|$ is given as

$$\Phi^{-1} : \mathbb{Z} \to |M| \qquad \Phi^{-1}(n) := \begin{cases} s^n_M(z_M) & \text{if } n \geq 0 \\ p^n_M(z_M) & \text{if } n \leq 0. \end{cases}$$

It remains to prove that Φ preserves the interpretations of the three symbols $z, s,$ and p. This is done in the following. Note, that we have $z_Z = 0, s_Z = \text{succ}$, and $p_Z = \text{pred}$.

Obviously, $\Phi(z_M) = 0$ holds. Now, assume $u \in |M|$. For a proof of $\Phi(s_M(u)) = \text{succ}(\Phi(u))$ we distinguish two cases. If $u = s^n_M(z_M)$, then we obtain

$$\Phi(s_M(u)) = \Phi(s^{n+1}_M(z_M)) = n + 1 = \Phi(s^n_M(z_M)) + 1 = \Phi(u) + 1 = \text{succ}(\Phi(u)).$$

In the case $u = p^n_M(z_M)$ we have

$$\Phi(s_M(u)) = \Phi(p^{n-1}_M(z_M)) = -n + 1 = \Phi(p^n_M(z_M)) + 1 = \Phi(u) + 1 = \text{succ}(\Phi(u)),$$

provided $n > 0$ (here we have used that sentence (1) is true in M), and

$$\Phi(s_M(u)) = \Phi(s_M(z_M)) = 1 = \Phi(z_M) + 1 = \Phi(u) + 1 = \text{succ}(\Phi(u)),$$

provided $n = 0$. Equation $\Phi(p_M(u)) = \text{pred}(\Phi(u))$ is proved analogously to the latter one. □

Summing up, we have the desired result that the structure Z is characterized by the theory T_Z:

Theorem 3.3 *The standard model Z is – up to isomorphism – the only term-generated model of T_Z.* □

4 There is no finite characterization of the integers without hidden functions

In this section we show (Theorem 4.3 below) that there is no finite theory of arbitrary sentences built up over the language L_Z of Section 3 which has Z as - up to isomorphism - only term-generated model. The crucial point of this proof is the use of the compactness theorem of first-order logic which implies that a theory T has a model if every finite subset of T has a model. However, to conclude the proof it is additionally necessary to get a term-generated model for the chosen theory. Here elimination of quantifiers plays an important role.

A theory T over a first-order language L admits *elimination of quantifiers* if and only if for every formula A built up over L there is a quantifier-free formula B built up over the same language such that $M \models A \leftrightarrow B$ for every model M of T. In model theory elimination of quantifiers is one of the methods for proving theories decidable. Some examples can e.g., be found in [10], Section 13. The next lemma shows that the theory T_Z of Section 3 admits elimination of quantifiers, whereby no additional free variables are introduced.

Lemma 4.1 *Assume A to be a formula built up over the language L_Z. Then there exists a quantifier-free formula B, also built up over L_Z, such that $M \models A \leftrightarrow B$ for every model M of T_Z and, furthermore, the set of the free variables of B is contained in the set of the free variables of A.*

Proof. a) In a first step we prove the existence of a quantifier-free formula B over L_Z such that $M \models A \leftrightarrow B$ for every model M of T_Z.

We are allowed to assume the given formula A to be of the form $\exists x(A_1 \wedge \ldots \wedge A_m)$, where each $A_i, 1 \leq i \leq m$, is an atomic formula or the negation of an atomic formula. A proof of this well-known fact can e.g., be found in [7], Section 3.1. Furthermore, we may suppose that the variable x occurs in each A_i. For, if x does not occur in some A_{i_0}, then we use the equivalence of $\exists x(A_1 \wedge \ldots \wedge A_m)$ and $A_{i_0} \wedge \exists x(A_1 \wedge \ldots \wedge A_{i_0-1} \wedge A_{i_0+1} \wedge \ldots \wedge A_m)$.

Assume $y_i, 1 \leq i \leq k$, to denote the free variables of $\exists x(A_1 \wedge \ldots \wedge A_m)$. For a being an element from $\{z, x, y_1, \ldots, y_k\}$, we abbreviate the term $s(\ldots s(a) \ldots)$ (resp. $p(\ldots p(a) \ldots)$) with $n \geq 0$ occurrences of s (resp. p) by $s^n(a)$ (resp. $p^n(a)$). Particularly, we have $s^0(a) := p^0(a) := a$.

Now, suppose M to be a model of the theory T_Z. Each atomic sub-formula of A is an equation $t_1 \approx t_2$, where the terms $t_i, 1 \leq i \leq 2$, are built up using the variables $y_i, 1 \leq i \leq k$, the variable x, and the function symbols z, s, and p. Since the variable x occurs in at least one of the terms and the sentences (1) and (2) are true in M, there exist natural numbers m and n and $a \in \{z, x, y_1, \ldots, y_k\}$ such that $t_1 \approx t_2$ is equivalent to one of the following equations:

$$\begin{array}{llll} \text{(i)} & s^m(x) \approx s^n(a) & \text{(ii)} & s^m(x) \approx p^n(a) \\ \text{(iii)} & p^m(x) \approx s^n(a) & \text{(iv)} & p^m(x) \approx p^n(a). \end{array}$$

In the case $m \leq n$, the first equation is equivalent to $x \approx s^{n-m}(a)$; otherwise it is equivalent to $s^{m-n}(x) \approx a$, i.e., to $x \approx p^{m-n}(a)$. The proofs that also for the remaining equations there exist equivalent formulae of this specific form are identical and follow likewise from the validity of (1) and (2) in M.

Hence, we may suppose that every atomic formula occurring in A is of the form $x \approx s^n(a)$ or $x \approx p^n(a)$, where $a \in \{z, x, y_1, \ldots, y_k\}$. However, we may further suppose that a is different from x. This is due to the fact that $x \approx s^n(x)$ as well as $x \approx p^n(x)$ can be replaced by $z \approx z$ if $n = 0$, and by $\neg(z \approx z)$ if $n \neq 0$, and that the latter closed formulae can again be moved out-side of quantification.

Summing up, we may assume the given formula A to be of the form $(1 \leq m, 1 \leq j \leq m)$

$$\exists x(x \approx t_1 \wedge \ldots \wedge x \approx t_{j-1} \wedge \neg(x \approx t_j) \wedge \ldots \wedge \neg(x \approx t_m)),$$

where the terms $t_i, 1 \leq i \leq m$, are of the form $s^n(a)$ or $p^n(a)$ and $a \in \{z, y_1, \ldots, y_k\}$. Now, we distinguish three cases:

Case 1: $j = 1$, i.e, the formula A has the form $\exists x(\neg(x \approx t_1) \wedge \ldots \wedge \neg(x \approx t_m))$. It can easily be shown that the carrier set of each model of the theory T_Z is infinite. Now

$$\begin{array}{lll} M \text{ is a model of } T_Z & \Rightarrow & |M| \text{ is infinite} \\ & \Rightarrow & M \models \forall y_1 \ldots \forall y_m (\exists x(\neg(x \approx y_1) \wedge \ldots \wedge \neg(x \approx y_m))) \\ & \Rightarrow & M \models \exists x(\neg(x \approx t_1) \wedge \ldots \wedge \neg(x \approx t_m)) \end{array}$$

implies that A is true in M. Since $M \models z \approx z$ holds, too, we may choose B as $z \approx z$ and obtain, thus, $M \models A \leftrightarrow B$.

Case 2: $j > 1$ and $m = 1$, i.e., A has the form $\exists x(x \approx t_1)$. Then A is also valid in M and we may again choose B as formula $z \approx z$.

Case 3: $j > 1$ and $m \geq 2$, i.e., A contains an equation and there is at last a further equation and/or negation of an equation:

$$\exists x(x \approx t_1 \wedge \ldots \wedge x \approx t_{j-1} \wedge \neg(x \approx t_j) \wedge \ldots \wedge \neg(x \approx t_m)).$$

In this case, first, we delete the equation $x \approx t_1$ from A and then replace in the resulting formula every occurrence of the variable x by the term t_1. Since x does not occur in the terms t_i, $1 \leq i \leq m$, this leads to

$$\exists x(t_1 \approx t_2 \wedge \ldots \wedge t_1 \approx t_{j-1} \wedge \neg(t_1 \approx t_j) \wedge \ldots \wedge \neg(t_1 \approx t_m)),$$

a formula, which is equivalent to the original one. (Note, that the matrix of the original formula A is quantifier-free.) We have now a formula in the matrix of which x no longer occurs, so the quantifier may be omitted. Now, we choose B as formula

$$t_1 \approx t_2 \wedge \ldots \wedge t_1 \approx t_{j-1} \wedge \neg(t_1 \approx t_j) \wedge \ldots \wedge \neg(t_1 \approx t_m).$$

With this choice, we have again that $M \models A \leftrightarrow B$ holds.

b) The additional property is an immediate consequence of the construction of B. Either B is closed (cases 1 and 2) or the sets of the free variables of A and B are identical (case 3). □

Let L be a first-order language with at least one constant symbol. Furthermore, let T be a theory over L such that each sentence of T is a *prenex universal formula,* i.e., of the form $\forall x_1 \ldots \forall x_n A$, where $n \geq 0$ and A (the "matrix" of the formula) is quantifier-free. If T has a model, then it has also a term-generated one. For a logic without equality a proof of this well-known fact can e.g., be found in [8], p. 19; the generalization of this proof to a logic with equality is trivial.

As an immediate consequence, we obtain:

Lemma 4.2 *Assume A to be a sentence built up over the language L'_Z. If there is a model of the theory $T_Z \cup \{A\}$, then there is also a term-generated one.*

Proof. We use Lemma 4.1 and obtain that for every sentence A over L_Z there exists a quantifier-free sentence B over the same language such that the class of all models of $T_Z \cup \{A\}$ equals the class of all models of $T_Z \cup \{B\}$. Each sentence of T_Z is a prenex universal formula. Since B is a prenex universal formula, too, the above mentioned property of the class of these formulae applies. □

After these preparations, we are now able to prove the desired result.

Theorem 4.3 *There is no finite theory over the first-order language L_Z which has the structure Z as – up to isomorphism – only term-generated model.*

Proof. Suppose, for a contradiction, that we are given a finite theory $\{A_1, \ldots, A_m\}$ over the language L_Z which has - up to isomorphism - the structure Z as only term-generated model. We define the sentence A by $A := A_1 \wedge \ldots \wedge A_m$.

Claim: Each finite subset S of the theory $T_Z \cup \{\neg A\}$ has a model.

Proof: If $\neg A \notin S$, then Z is a model. Otherwise, let $k := \max\{n : (3.n) \in S\}$. We define a structure M for the language L_Z as a "loop of size $k+1$", i.e., by $|M| := \{0, \ldots, k\}$ and

$$z_M := 0 \qquad s_M(u) := \begin{cases} u+1 & \text{if } u \neq k \\ 0 & \text{if } u = k \end{cases} \qquad p_M(u) := \begin{cases} u-1 & \text{if } u \neq 0 \\ k & \text{if } u = 0. \end{cases}$$

It is obvious that the sentences (1), (2) and (3.n), where $1 \leq n \leq k$, are true in M. Also $M \models \neg A$ holds. Otherwise, we would have $M \models A$ which implies (the structure M is term-generated) that M and Z were isomorphic. Thus, we have a contradiction. Summing up, M is a model of S.

Now, we use the compactness theorem of first-order logic to deduce that the theory $T_Z \cup \{\neg A\}$ has a model. In combination with Lemma 4.2 this implies the existence of a term-generated model M of $T_Z \cup \{\neg A\}$. M is also a term-generated model of T_Z. From this fact and Theorem 3.3 we obtain that the two models M and Z of T_Z are isomorphic. As a consequence, $M \models A$ holds. But this is a contradiction to $M \models \neg A$. □

Consider the sub-theory of T_Z containing the two sentences (1) and (2) only. It can be shown that each term-generated model of this theory is either isomorphic to Z or to a "loop of size n". In the manner of speaking of algebraic specifications or universal algebra, Z is initial in the class of all term-generated models of $\{(1), (2)\}$. To obtain this model as – up to isomorphism – only term-generated model, one has to extend the theory in such a way that loops are prevented, i.e., infinitely many inequalities can be derived. Theorem 4.3 states that the language used so far is too "poor" to do this in a finite manner.

5 A finite characterization of the integers using a hidden function

As just mentioned, a finite extension of the theory $\{(1), (2)\}$ which prevents loops requires an extension of the language L_Z, i.e., the use of hidden machinery. In this section we show, that a symbol for the usual ordering on the integers suffices. To this end, we extend the language L_Z to $\overline{L}_Z := L_Z \cup \{\ll\}$, where $\ll$ is a 2-place predicate symbol. Furthermore, we consider the three sentences (the symbol $\ll$ is used in infix notation)

$$(3)\ \forall x(\neg(s(x) \ll x)) \quad (4)\ \forall x(x \ll x) \quad (5)\ \forall x \forall y(s(x) \ll y \rightarrow x \ll y).$$

And, finally, we define the finite theory $\overline{T}_Z$ over $\overline{L}_Z$ to consist of the sentences (1) and (2) of T_Z and the sentences (3), (4), and (5).

Clearly, the structure $\overline{Z} := (\mathbb{Z}, 0, \text{succ}, \text{pred}, \leq)$ for $\overline{L}_Z$ is a term-generated model of $\overline{T}_Z$. In the rest of this section we prove that each other term-generated model is isomorphic to this model. As in Section 3, therefore, we assume in the sequel an arbitrarily chosen (but fixed) term-generated model $M := (|M|, z_M, s_M, p_M, \ll_M)$ of $\overline{T}_Z$. In the following, we write $u \ll_M v$ (resp. $u \not\ll_M v$) instead of $\ll_M (u,v) = 1$ (resp. $\ll_M (u,v) = 0$). As in the case of T_Z we obtain:

Lemma 5.1 *Let $u \in |M|$. Then there exists exactly one natural number $n \in \mathbb{N}$ such that $u = s_M^n(z_M)$ or $u = p_M^n(z_M)$.*

Proof. As the existence of n follows from the validity of (1) and (2) in M (cf. the proof of Lemma 3.1), it remains to show uniqueness.

If $u = s_M^m(z_M) = s_M^n(z_M)$ and $m + k = n$ (where $k > 0$), then

$$\begin{aligned} s_M^m(z_M) = s_M^m(s_M^k(z_M)) &\Rightarrow s_M^k(Z_M) = Z_M \Rightarrow s_M^k(z_M) \ll_M z_M \\ &\Rightarrow s_M(z_M) \ll_m z_M, \end{aligned}$$

since (2), (4), and (5) are true in M. However, $s_M(z_M) \ll_M z_M$ is a contradiction to the validity of formula (3) in M.

The remaining cases are handled similarly. □

The proof of the fact that the function Φ of the third section is an isomorphism from M to $\overline{Z}$, too, is prepared by a simple

Lemma 5.2 *If $u \in |M|$ and $n \in \mathbb{N}$, then $n > 0$ implies $s_M^n(u) \not\ll_M u$.*

We use induction on n. The induction base $n = 1$ holds since sentence (3) is true in M; the induction step proceeds as follows: From the validity of (5) in M we obtain

$$s_M^{n+1}(u) \ll_M u \Rightarrow s_M^n(u) \ll_M u$$

and, thus, contraposition in conjunction with the induction hypothesis applies. □

Now, we are able to prove:

Lemma 5.3 *The function Φ of Section 3 is also an isomorphism from the fixed model M to the model $\overline{Z}$.*

Proof. Due to Lemma 3.2 of the third section, we have only to prove that Φ preserves the two interpretations $\ll_M$ and $\leq$ of the predicate symbol $\ll$, i.e., that for all $u, v \in |M|$

$$u \ll_M v \Leftrightarrow \Phi(u) \leq \Phi(v).$$

Assume $u = s_M^n(z_M)$ and $v = s_M^m(z_M)$. For a proof of direction "$\Rightarrow$" we show that $\Phi(u) \not\leq \Phi(v)$ implies $u \not\ll_M v$. From $\Phi(u) \not\leq \Phi(v)$ we get $m > n$, hence $m = k + n$, where $k > 0$. Thus,

$$u = s_M^{k+n}(z_M) = s_M^k(s_M^n(z_M)) = s_M^k(v).$$

Due to this result, $u \not\ll_M v$ is equivalent to $s_M^k(v) \not\ll_M v$ and Lemma 5.2 applies. Now, we prove direction "$\Leftarrow$". From $\Phi(u) \leq \Phi(v)$ we obtain that $m \leq n$ holds, i.e., $k + m = n$, where $k \geq 0$. This shows the equation

$$s_M^k(u) = s_M^k(s_M^m(z_M)) = s_M^{k+m}(z_M) = v.$$

In combination with the validity of (4) in M, this result yields $s_M^k(u) \ll_M v$ which in turn implies (since (5) is true in M) that $u \ll_M v$.

Next, let $u = s_M^m(z_M)$ and $v = p_M^n(z_M)$. For a proof of "$\Rightarrow$" we distinguish between $m+n=0$ and $m+n>0$. The first case is trivial. In the second case we use that (1) is true in M and get

$$u \ll_M v \Leftrightarrow s_M^m(s_M^n(p_M^n(z_M))) \ll_M p_M^n(z_M) \Leftrightarrow s_M^{m+n}(p_M^n(z_M)) \ll_M p_M^n(z_M).$$

Now, Lemma 5.2 shows that the premise of the implication to be proven does not hold. A proof of "$\Leftarrow$" is trivial.

The remaining cases can be shown analogously. □

We now have that the structure $\overline{Z}$ is characterized by the theory $\overline{T}_Z$:

Theorem 5.4 *The model $\overline{Z}$ is – up to isomorphism – the only term-generated model of $\overline{T}_Z$.* □

It is obvious that the use of a predicate symbol for the ordering (in combination with an extension of the theory $\{(1),(2)\}$) is not the only way to prevent loops. E.g., one can also extend the langauge L_Z by a predicate symbol n and $\{(1),(2)\}$ by the four sentences

$$\begin{array}{llll} (6) & \neg n(z) & (7) & n(p(z)) \\ (8) & \forall x(n(x) \rightarrow n(p(x))) & (9) & \forall x(n(s(x)) \rightarrow n(x)) \end{array}$$

which specify the interpretation of n to test a given integer for being negative or not. Another possibility is to introduce inductively (using z, s, and p) a 2-place function symbol f that describes the repeated application of the symbols s and p, resp. A natural way to specify f is

$$\begin{array}{ll} (10) & \forall x(f(x,z) \approx x) \\ (11) & \forall x \forall y(f(x,s(y)) \approx s(f(x,y))) \\ (12) & \forall x \forall y(f(x,p(y)) \approx p(f(x,y))). \end{array}$$

We may then substitute in the theory T_Z the infinite set (3.n), $n \geq 1$, of sentences by a single one, viz.

$$(13)\ \forall x \forall y(\neg(y \approx z) \rightarrow \neg(f(x,y) \approx z)).$$

In both cases, the proof of isomorphism is mainly a consequence of (the validity of) Lemma 3.1.

We finish this section with a remark concerning our proof method. Certain, our "model-oriented" approach is not the only way to solve the given problem. For instance, a proof which argues algebraically can proceed as follows: One shows that the initial term-generated model Z of the theory $\{(1),(2)\}$ can be extended by the ordering relation $\leq$ in such a way that the resulting structure $\overline{Z}$ for $\overline{L}_Z$ is initial wrt. T_Z. Since the truth values 0 and 1 are different, the ordering relation cannot identify elements. Now, the desired isomorphism result is an immediate consequence of the initiality of $\overline{Z}$. This remark shows also: For a translation of the proof of this section into the notation of algebraic specifications a specification of the truth values is required which has – up to isomorphism – the two element Boolean algebra as only model.

6 Concluding remarks

From a theoretical point of view, hidding machinery is used to overcome the lack of expressive power. In the present paper we have shown its necessity even in the case of full first-order specifications. To this end, first, we have presented an infinite first-order theory T_Z whose term-generated models are exactly the structures isomorphic to $Z = (\mathbb{Z}, 0, \text{succ}, \text{pred})$. Then we have shown that there is no finite set of first-order sentences which has the same property. And, finally, we have given unique characterizations of Z using hidding machinery.

For the proof of the main result (Theorem 4.3) we have used the argument that the theory $T_Z \cup \{\neg A\}$ has a term-generated model if every of its finite subsets has a model. It seems that this argument (an extension of the compactness theorem of first-order logic) can also be used to prove that there is no finite characterization of more complex data types without hidden functions.

For the description of large structures and systems it is necessary to compose specifications in a modular way from smaller ones to master complexity. Hidding is one of these so-called specification-building operations and contained in almost all modern specification languages; see [13] for an overview. Frequently, its use makes specifications more readable and understandable. Furthermore, in various case studies it has proven advantageous to use hidding if specifications are transformed, e.g., into versions which provide algorithmic solutions. As two examples for this latter application we mention the papers [5] and [4]. In all these cases the decisive question is how to find suitable hidden functions and their defining formulae. This aspect of hidding was not addressed here, but some work can be found in the literature. However, it seems that a general methodology for the *practical* use of hidden machinery remains to be developed.

Acknowledgement. This paper benefited from valuable discussions with Ulf Schmerl. I am also grateful to the referees for their helpful comments.

References

[1] Bauer F.L., Wössner H.: Algorithmic Language and Program Development. Springer Verlag, Berlin-Heidelberg-New York (1982)

[2] Bergstra J.A., Tucker J.V.: Algebraic Specifications of Computable and Semicomputable Data Structures. Research Report IW 115, Department of Computer Science, Mathematical Centre, Amsterdam (1979)

[3] Bergstra J.A., Tucker J.V.: Algebraic Specifications of Computable and Semicomputable Data Types. *Theoretical Computer Science* 50, 137-181 (1987)

[4] Broy M.: Deductive Program Development: Evaluation in Reverse Polish Notation as an Example. In: Broy M., Wirsing M. (eds.): Methods of Programming. LNCS 544, Springer Verlag, Berlin-Heidelberg-New York, 79-99 (1991)

[5] Dosch W., Wirsing M., Ausiello G., Mascari G.F.: Polynomials – The Specification, Analysis and Development of an Abstract Data Type. In: Wilhelm R. (ed.): GI - 10. Jahrestagung, Informatik Fachberichte 33, Springer Verlag, Berlin-Heidelberg-New York, 306-320 (1991)

[6] Ehrig H., Mahr B.: Fundamentals of Algebraic Specifications 1. Equations and Initial Semantics. EATCS Monographs in Theoretical Computer Science, Vol. 6, Springer Verlag, Berlin-Heidelberg-New York (1985)

[7] Enderton H.B.: A Mathematical Introduction to Logic. Academic Press, London (1972)

[8] Kreisel G., Krivine J.-L.: Modelltheorie – Eine Einführung in die mathematische Logik. Hochschultexte, Springer Verlag, Berlin-Heidelberg-New York (1972)

[9] Majster M.: Data Types, Abstract Data Types and their Specification Problem. Report TUM-I7740, Institut für Informatik, TU München (1977) Also in: *Theoretical Computer Science 8,* 89-127 (1979)

[10] Monk J.D.: Mathematical Logic. Springer Verlag, New York-Heidelberg-Berlin (1976)

[11] Thatcher J.W., Wagner E. G., Wright J.B.: Data Type Specification: Parameterization and the Power of Specification Techniques. Proc. 10^{th} SIGACT Symp. Theory of Computing, 119-132 (1978)

[12] Wirsing M., Pepper P., Partsch H., Dosch D., Broy M.: On Hierarchies of Abstract Data Types. *Acta Informatica* 20, 1-33 (1983)

[13] Wirsing M.: Algebraic Specifications. In: van Leeuwen J. (ed.): Handbook of Theoretical Computer Science B, North-Holland, 675-788 (1990)

Received October 22, 1992

Acta Cybernetica, Vol. 11, No. 1-2, Szeged, 1993

On codes concerning bi-infinite words

Do Long Van* Nguyen Huong Lam* Phan Trung Huy*

Abstract

In this paper we consider a subclass of circular codes called Z-codes. Some tests of Sardinas-Patterson type for Z-codes are given when they are finite or regular languages. As consequences, we prove again the results of Beal and Restivo, relating regular Z-codes to circular codes and codes with finite synchronization delay. Also, we describe the structure of two-element Z-codes.

1 Preliminary

In this paper only very basic notions of free monoids and formal languages are needed. As a general reference we mention [7], and for the facts concerning codes we always refer to [3] silently. In addition to this we use also notions concerning infinite and bi-infinite words without very formal definitions because of a wide availability of papers on the subject. To fix our notations we want to specify the following. Throughout this paper A denotes a finite alphabet. The free monoid generated by A, or the set of finite words, is denoted by A^* and its neutral element, the empty word, by ε. As usual we set $A^+ = A^* - \varepsilon$. For a word x in A^*, $|x|$ means the length of x. We call a nonempty word x *primitive* if it is not a proper power of any word, otherwise x is *imprimitive*. We call two words x and y *copower* if they are powers of the same word. For example, as well known two different words are copower if and only if the set they form is not a code. For two finite words x and y the notation yx^{-1} and $x^{-1}y$ are used to denote the right and the left quotient of y by x respectively. Naturally, the quotient and the product of two words can be extended to languages, i.e. subsets of A^* :

$$\begin{aligned} X^{-1}Y &= \{x^{-1}y : x \in X, y \in Y\},\ YX^{-1} = \{yx^{-1} : x \in X, y \in Y\}. \\ XY &= \{xy : x \in X, y \in Y\}, X^2 = XX, \ldots; \end{aligned}$$

and $X^* = \bigcup_{n \geq 0} X^n$ (the Kleene closure of X).

In the following, our consideration is mainly based on the notion of infinite and bi-infinite words on A. Let ${}^N\!A$, A^N, A^Z be the sets of left infinite, right infinite and bi-infinite words on A respectively. For a language X of A^*, we denote ${}^\omega X$, X^ω and ${}^\omega X^\omega$ the left infinite, the right infinite and the bi-infinite product of nonempty words of X respectively, i.e. their elements are obtained by concatenation of words of $X - \varepsilon$ carried out infinitely to the left, to the right or infinitely in both directions. For example,

$${}^\omega X = \{\ldots u_2 u_1 : u_i \in X - \varepsilon, i = 1, 2, \ldots\}.$$

*Institute of Mathematics, P.O.Box 631, 10000 Hanoi, Vietnam

Factorizations in elements of X (over X, on X) of a left or right infinite word are understood customarily (see [10] for details), but factorizations of a bi-infinite word need a special treatment as follows. Let $w \in A^Z$ be in the form:

$$w = \dots a_{-2}a_{-1}a_0a_1a_2\dots$$

with $a_i \in A$. A *factorization* on elements of X of the bi-infinite word w is a strictly increasing function $\mu : Z \longrightarrow Z$ satisfying $x_i = a_{\mu(i)+1}\cdots a_{\mu(i+1)} \in X$ for all $i \in Z$. Two factorizations μ and λ are said to be *equal,* denoted $\mu = \lambda$ if there is $t \in Z$ such that $\lambda(i+t) = \mu(i)$ for all $i \in Z$. Otherwise, λ and μ are *distinct,* denoted $\mu \neq \lambda$. It is easy to verify that $\mu \neq \lambda$ iff $\mu(Z) \neq \lambda(Z)$, or equivalently, there exist a word $u \in A^+$, two bi-infinite sequences of words of $X: \dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots$ and $\dots, y_{-2}, y_{-1}, y_0, y_1, y_2, \dots$ such that

$$\begin{aligned} \dots x_{-2}x_{-1}u &= \dots y_{-1}y_0, \quad |u| \le |x_0|, \\ x_0x_1\dots &= uy_1y_2\dots, \quad |u| \le |y_0| \end{aligned}$$

with $u \neq x_0$ or $u \neq y_0$.

If every rigth infinite word of A^N has at most one factorization on elements of X then X is said to be an N-*code* (see [10], where in a wider context N-code is called *strict* code). Analogously, if every left infinite word possesses this property, we call X an $\overline{N}$-*code.* Obviously, X is an N-code iff $\overline{X} = \{\overline{x} : x \in X\}$ is an $\overline{N}$-code, where $\overline{x}$ is the mirror image of the word x. For the bi-infinite words, we have our basic

Definition 1 *A language X of A^+ is a Z-code if all factorizations on X of every bi-infinite word are equal.*

Example 1 Every singleton $\{u\}$ is always both an N-code and an $\overline{N}$-code but it is a Z-code if and only if u is primitive. The two-word language $X = \{ab, ba\}$ is both an N-code and an $\overline{N}$-code, but it is not a Z-code since the word ${}^{\omega}(ab)^{\omega}$ has two factorizations $\dots ab.ab.ab\dots$ and $\dots ba.ba.ba\dots$, which are verified directly to be distinct.

The family of Z-codes is closely connected with the so-called circular code [3]. A language X of A^* is said to be *circular* if for any $x_0, x_1, \dots x_m, y_0, y_1, \dots y_n$ of X and s, t of A^* the equalities

$$\begin{aligned} x_1x_2\dots x_m &= ty_0\dots y_m s, \\ x_0 &= st \end{aligned}$$

imply $s = \varepsilon, m = n$ and $x_0 = y_0, \dots, x_m = y_m$.

It is easy to see that every circular language is a code and that every Z-code is a circular code. But not always a circular code is a Z-code, as the following code [4] $X = \{ab\} \cup \{ab^iab^{i+1}, i = 0, 1, 2, \dots\}$ shows that. Nevertheless, every regular circular code is a Z-code i.e. the families of regular Z-codes and regular circular codes coincide, as shown by Beal [2]. Therefore, results and algorithms invented for circular codes can be applied to Z-codes. However, in the next section we work independently with Z-codes, proposing some tests for regular and finite Z-codes. As consequences of that, we can obtain a result of A. Restivo on codes with finite (bounded) synchronization delay [11] and the aforementioned Beal's result. Also, for completeness, as an easy consequence of [1], we describe the structure of two-word Z-codes.

2 Tests for Z-codes

We develop now a criterion to verify whether a finite subset X of A^+ is a Z-code. Our procedure is something like the Sardinas- Patterson one (cp. [10]), but actually instead of one sequence of subsets associated to X we need two sequences associated to each overlap of elements of X. Precisely, we define first the subset:

$$W(X) = \{w \in A^+ : \exists u, v \in A^*; \exists x, y \in X : uw = x, wv = y, uv \neq \varepsilon\}$$

whose element is called an *overlap* of elements of X. For each $w \in W(X)$, we define two sequences $U_i(w, X)$ an $V_i(w, X)$ of subsets of A^* as follows

$$\begin{aligned} U_0(w,X) &= w^{-1}X - \{\varepsilon\}, \\ U_{i+1}(w,X) &= U_i(w,X)^{-1}X \cup X^{-1}U_i(w,X), \\ V_0(w,X) &= Xw^{-1} - \{\varepsilon\}, \\ V_{i+1}(w,X) &= XV_i(w,X)^{-1} \cup V_i(w,X)X^{-1}, \end{aligned}$$

$i = 0, 1, 2, \ldots$. Further, if there is no risk of confusion, instead of $W(X)$, $U_i(w, X)$, $V_j(w, X)$ we write simply W, U_i, V_j. The following property of $U_i(w, X)$, $V_j(w, X)$ is useful in the sequel.

Lemma 1 *For every $N \geq 0$ and for any word $u, u \in U_N(w, X)$ iff there exist $x_1, \ldots, x_n, y_1, \ldots, y_m \in X$ such that $m + n - 1 = N$ and either*

$$wx_1 \ldots x_n = y_1 \ldots y_m u, \quad |u| \leq |x_n|, |w| < |y_1|$$

or

$$wx_1 \ldots x_n u = y_1 \ldots y_m, \quad |u| \leq |y_m|, |w| < |y_1|.$$

Remark. Similarly, the symmetrical statement holds for V_j.

Proof. By induction on N. For $N = 0$ we have

$$u \in U_0 \Leftrightarrow (\exists y_1 \in X : w^{-1}y_1 = u \Leftrightarrow wu = y_1, |u| < |y_1|, |w| < |y_1|).$$

Suppose the lemma is true for some $N \geq 0$, we prove it true for $N + 1$. We have

$$u \in U_{N+1} \Leftrightarrow \exists u' \in U_N, \exists x \in X : u'u = x \vee xu = u'.$$

By induction hypothesis, $u' \in U_N$ iff there exist $x_1, \ldots, x_n, y_1, \ldots, y_m \in X$ such that $n + m - 1 = N$ and either

$$wx_1 \ldots x_n u' = y_1 \ldots y_m, \quad |u'| \leq |y_m|, \quad |w| < |y_1| \tag{1}$$

or

$$wx_1 \ldots x_n = y_1 \ldots y_m u', \quad |u'| \leq |x_n|, \quad |w| < |y_1|. \tag{2}$$

Therefore $u \in U_{N+1}$ is equivalent to the fact that there exist $x_1, \ldots, x_n$, x, $y_1, \ldots, y_m$ in X such that

$$((u'u = x)\&((1) \vee (2))) \vee ((xu = u')\&((1) \vee (2)))$$

or equivalently

$$((u'u = x)\&(1)) \vee (u'u = x)\&(2)) \vee$$
$$((xu = u')\&(1)) \vee ((xu = u')\&(2)).$$

The last, in its turn, as it is easy to verify, is equivalent to the fact that there exist $x_1, \ldots, x_{n'}, y_1, \ldots, y_{m'}$ in X such that $n' + m' - 1 = N + 1$ and

$$wx_1 \ldots x_{n'} = y_1 \ldots y_{m'}u, \quad |u| \leq |x_{n'}|, \quad |w| < |y_1|$$

or

$$wx_1 \ldots x_{n'}u = y_1 \ldots y_{m'}, \quad |u| \leq |x_{m'}|, \quad |w| < |y_1|,$$

i.e. the lemma is true also for $N + 1$.

Now we state a sufficient condition for a language to be a Z-code.

Proposition 1 *A finite subset X of A^+ is a Z-code if for every overlap w of elements of X, the following conditions hold:*

(i) if $w \in W \cap X$ then $U_i = \emptyset$ and $V_j = \emptyset$ for some $i, j \geq 0$;

(ii) if $w \in W - X$ then $U_i = \emptyset$ or $V_j = \emptyset$ for some $i, j \geq 0$.

Proof. We suppose that X is not a Z-code, i.e. at least one word of A^Z possesses two distinct factorizations on X, therefore we have two equalities:

$$\ldots x_{-2}x_{-1}w = \ldots y_{-1}y_0 \tag{1}$$

$$x_0x_1 \ldots = wy_1y_2 \ldots \tag{2}$$

for some $w \in A^+, |w| \leq |y_0|$ and $|w| \leq |x_0|, w \neq x_0$ or $w \neq y_0$, hence $w \in W$.

If $w \in W \cap X$ and, say, $w \neq x_0$, then $U_0 \neq \emptyset$. By (2), for every $N > 0$ there is the least integer $n \geq 0$ such that $|x_0 \ldots x_n| \geq |wy_1 \ldots y_N|$, that is

$$x_0x_1 \ldots x_n = wy_1 \ldots y_N u$$

for some word $u \in A^*, |u| < |x_n|$. By Lemma 1, $u \in U_{N+n}$. Thus $U_N \neq \emptyset$: (i) does not hold. For the case $w \neq y_0$, by (1) and the symmetrical version of Lemma 1 we get $V_N \neq \emptyset$ for all $N \geq 0$: (i) does not hold again.

Now let $w \in W - X$ then we have both $w \neq x_0$ and $w \neq y_0$. By the same argument as above we obtain $U_i \neq \emptyset$ and $V_j \neq \emptyset$ for all $i, j \geq 0$: (ii) does not hold. The proof is completed.

In order to make a converse of Proposition 1 for finite languages we prove a lemma, which places an upperbound on the least i such that $U_i = \emptyset$. For a finite subset $X = \{x_1, x_2, \ldots, x_n\}$ of A^* we define $\| X \| = \sum_{i=1}^{n} |x_i|$. Note that each U_i consists only of right factors (i.e. suffices) of words in X and if $U_k = U_l \neq \emptyset$ for $k \neq l$ then $U_i \neq \emptyset$ for all $i \geq 0$. Since the set of right factors of words in X is of cardinality at most $\| X \|$, such an upperbound obviously exists and we can take it as $2^{\|X\|}$. In the following lemma a more refined estimation is given.

Lemma 2 *For any finite subset X of A^* and $w \in W$, the following assertions are equivalent*

(i) $U_i(w, X) \neq \emptyset$ for some $i \geq \| X \|$;

(ii) $U_i(w, X) \neq \emptyset$ for all $i \geq 0$;

(iii) There exist infinite sequences $x_1, x_2, \ldots; y_1, y_2, \ldots$ of words in X such that

$$wx_1x_2 \cdots = y_1y_2 \cdots$$

with $|w| < |y_1|$.

Remark. The symmetrical statement holds for $V_j(w, X)$.

Proof. (iii) $\Rightarrow$ (ii): already done in the proof of Proposition 1.

(ii) $\Rightarrow$ (i): obvious.

(i) $\Rightarrow$ (iii): Let $u_N \in U_N(w, X), N \geq \| X \|$. Then there exist $u_i \in U_i(w, X)$ such that $u_0 = w, u_{i+1} \in u_i^{-1}X$ or $X^{-1}u_i, i = 0, 1, \ldots, N-1$. It is easy to see that $u_0, u_1, \ldots, u_N$ are suffices of words in X and the cardinality of the set of the suffices of the finite set X does not exceed $\| X \|$ and thus is less than $N+1$. Therefore, there are p and $q, 0 \leq p < q \leq N$ such that $u_p = u_q$. Let l be the largest number not exceeding $q-p$ such that $u_{p+1} = y_1^{-1}u_p, u_{p+2} = (y_1y_2)^{-1}u_p, \ldots, u_{p+l} = (y_1 \ldots y_l)^{-1}u_p$, where $y_1, \ldots, y_l \in X$; otherwise $l = 0$. Then $u_{p+l+1} \in u_{p+l}^{-1}X$ and we apply Lemma 1 to the case $u_q \in U_{q-p-l}(u_{p+l}, X)$ to obtain some words $x_1, \ldots, x_n$ and $z_1, \ldots, z_m$ of X such that

$$u_{p+l}x_1 \ldots x_n = z_1 \ldots z_m u_q$$

or

$$u_{p+l}x_1 \ldots x_n u_q = z_1 \ldots z_m.$$

Whence

$$u_p x_1 \ldots x_n = y_1 \ldots y_l z_1 \ldots z_m u_q$$

or

$$u_p x_1 \ldots x_n u_q = y_1 \ldots y_l z_1 \ldots z_m.$$

Since $u_p = u_q$, these equalities lead respectively to the infinite words

$$u_p(x_1 \ldots x_n)^\omega = (y_1 \ldots y_l z_1 \ldots z_m)^\omega \tag{1}$$

or

$$u_p(x_1 \ldots x_n y_1 \ldots y_l z_1 \ldots z_m)^\omega = (y_1 \ldots y_l z_1 \ldots z_m x_1 \ldots x_n)^\omega. \tag{2}$$

On the other hand, since $u_p \in U_p(w, X)$, again by Lemma 1 we have

$$wx = y'yu_p, \quad |w| < |y'| \tag{3}$$

or

$$wxu_p = y'y, \quad |w| < |y'| \tag{4}$$

where $y' \in X, x, y \in X^*$. Combining (3) and (4) with (1) and (2), we get four possibilities that all lead to the desired infinite equality in (iii). Lemma 2 is proved.

Now we are ready to state our criterion.

Theorem 1 *A finite subset x of A^+ is a Z-code if and only if for every overlap w of elements of X, the following conditions hold:*

(i) if $w \in W \cap X$ then $U_i(w, X) = \emptyset$ and $V_j(w, X) = \emptyset$ for some $i, j < \| X \|$;

(ii) if $w \in W - X$ then $U_i(w, X) = \emptyset$ or $V_j(w, X) = \emptyset$ for some $i, j < \| X \|$.

Proof. The sufficient part is Proposition 1, we have to prove only the necessary one. Suppose that (i) or (ii) does not hold. We shall derive from this two equalities which show that X is not a Z-code. In fact, by Lemma 2 and its symmetrical version, we have two cases: there exist

(1) $w \in W \cap X$ and $x_i, y_j \in X, i, j = 0, 1, 2, \ldots$ such that

$$x_0 x_1 \cdots = w y_0 y_1 \ldots, \quad |w| < |x_0|$$

or

$$\ldots x_1 x_0 = \ldots y_1 y_0 w, \quad |w| < |x_0|;$$

(2) $w \in W - X$ and $x_i, y_j \in X, i, j = \cdots -2, -1, 0, 1, 2, \ldots$ such that

$$x_0 x_1 \cdots = w y_0 y_1 \ldots, \quad |w| < |x_0|$$

and

$$\ldots x_{-1} x_0 = \ldots y_{-1} y_0 w, \quad |w| < |x_0|$$

regarding (i) or (ii) does not hold.

The first case together with the obvious equalities $\ldots ww = \ldots ww$ and $ww \ldots = ww \ldots$ show that X is not a Z-code.

The equalities in the second case themselves ensure that X is not a Z-code. The proof is completed.

We give now some examples illustrating the execution of the algorithm.

Example 2 (a) Consider $X = \{a^2b, b^2a\}$. We apply Theorem 1 to show that X is a Z-code.

$$\begin{aligned} W &= \{a, b\}, \\ U_0(a, X) &= \{ab\}, \ U_1(a, X) = \emptyset, \\ U_0(b, X) &= \{ba\}, \ U_1(b, X) = \emptyset. \end{aligned}$$

Since $a, b \notin X$, we conclude that X is a Z-code.

(b) Let $X = \{u\}$ with u imprimitive, $u = \lambda^n (n \geq 2)$. Clearly $\lambda \in W - X$, $U_0(\lambda, X) = \{\lambda^{n-1}\}$, which implies $\lambda \in U_1(\lambda, X), \lambda^{n-1} \in U_2(\lambda, X), \ldots$. Thus $U_i(\lambda, X) \neq \emptyset$ for all $i \geq 0$. So $\{u\}$ is not a Z-code.

Conversely, let $X = \{u\}$ not be a Z-code and let λ be an overlap of X such that $U_i(\lambda, X) \neq \emptyset$ for all $i \neq 0$. Since λ is an overlap of u, we have $x\lambda = u$ for some $x \in A^+$. Further, if $\lambda_0 \in U_0(\lambda, X)$ then $\lambda\lambda_0 = u$. Hence $U_0(\lambda, X) = \{\lambda_0\}$. Let $\lambda_1 \in U_1(\lambda, X)$ then $\lambda_0\lambda_1 = u$. Thus $|\lambda_1| = |\lambda|$ and from $x\lambda = u$ it follows $\lambda = \lambda_1$. Consequently $\lambda_0\lambda = \lambda\lambda_0 = u$, which with $\lambda_0, \lambda_1 \neq \varepsilon$ yield that u is imprimitive. Thus $\{u\}$ is a z-code if and only if u is primitive.

The main setback of Theorem 1 is that it is unfit for infinite (even regular) languages.

Example 3 Consider $X = \{a, cab, c, bc^+d\}$ on the alphabet $A = \{a, b, c, d\}$. It is an infinite regular Z-code, but for all $i \geq 0 : U_i(c, X) \neq \emptyset$.

Nevertheless, for the important class of regular languages we can work out another algorithm close to the previous one, also of Sardinas-Patterson type. Let

X be a regular language and as before W be the set of overlaps. First, for each overlap $w \in W$ we construct two sequences:

$$\overline{U}_0 = w^{-1}X - \{\varepsilon\}, \qquad \overline{U}_{i+1} = \overline{U}_i^{-1}X^*,$$
$$\overline{V}_0 = Xw^{-1} - \{\varepsilon\}, \qquad \overline{V}_{i+1} = X^*\overline{V}_i^{-1}$$

for all $i \geq 0$, which, if needed, will be referred to as $\overline{U}_i(w, X)$ and $\overline{V}_j(w, X)$. Of course there is no need to compute $\overline{U}_i(w, X), \overline{V}_j(w, X)$ for all $w \in W$, it is sufficient to take representatives modulo the right and left principal congruence defined by X^* or X. Recall that for a subset X of A^* the following equivalence relation

$$u \equiv_R v \Leftrightarrow u^{-1}X = v^{-1}X, \quad u, v \in A^*,$$

called *right principal congruence* defined by X. Analogously is defined the *left principal congruence* $\equiv_L$. When X is regular, the number of right (left) principal congruence classes, called *right index* (resp. *left index*) of X, is finite and equal to the number of states of the minimal automaton recognizing X. Now we state

Theorem 2 *Let X be a regular subset of A^+ and m, e be the right and left index of X^*. Then X is a Z-code if and only if for all $w \in W$ the following conditions hold*
(i) $w \in W \cap X$ implies $\overline{U}_i(w, X) = \emptyset$ and $\overline{V}_j(w, X) = \emptyset$ for some $i < m, j < e$;
(ii) $w \in W - X$ implies $\overline{U}_i(w, X) = \emptyset$ or $\overline{V}_j(w, X) = \emptyset$ for some $i < m, j < e$.

Remark. As seen from the proof below, (i) and (ii) are sufficient for any language of A^* to be a Z-code.

Proof. In fact, we prove an equivalent statement: X is not a Z-code iff (i) or (ii) does not hold.

First, let X not be a Z-code. Then there exist two equalities:

$$\ldots x_{-2}x_{-1}w = \ldots y_{-1}y_0, \tag{1}$$

$$x_0x_1 \ldots = wy_1y_2 \ldots \tag{2}$$

with $|w| \leq |x_0|, |w| \leq |y_0|, x_i, y_j \in X, w \neq x_0$ or $w \neq y_0$, hence $w \in W$.

If $w \in W \cap X$, we assume for certainty that $w \neq y_0$ and consider (1), putting $v_0 = y_0w^{-1} \in \overline{V}_0$. From (1) we get

$$\ldots x_{-2}x_{-1} = \ldots y_{-2}y_{-1}v_0.$$

Choose $n \in N$ such that $|x_{-n} \ldots x_{-2}x_{-1}| \geq |v_0|$ and put again $v_1 = (x_{-1} \ldots x_{-1})v_0^{-1}$, hence $v_1 \in X^*v_0^{-1} \subseteq X^*\overline{V}_0^{-1} = \overline{V}_1$ and

$$\ldots x_{-(n+2)}x_{-(n+1)}v_1 = \ldots y_{-2}y_{-1}.$$

We apply this argument over and over again to see that $\overline{V}_j \neq \emptyset$ for all $j \geq 0$, i.e (i) does not hold.

If now $w \in W - X$, we have both $w \neq x_0$ and $w \neq y_0$. Similarly, we apply the argument above to (1) and (2) to verify $\overline{U}_i \neq \emptyset$ and $\overline{V}_j \neq \emptyset$ for all $i, j \geq 0$: (ii) does not hold.

Conversely, let $\overline{U}_i \neq \emptyset$ for all $i \geq 0$ and N be any integer not less than m, and $u_N \in \overline{U}_N$. There exist $u_i \in \overline{U}_i, i = 0, 1, \ldots, N-1$ such that $u_0 \in w^{-1}X$, $u_{i+1} \in u_i^{-1}X^*, i = 0, 1, \ldots, N-1$, or equivalently, $wu_0 \in X, u_iu_{i+1} \in X^*$, $i = 0, 1, \ldots, N-1$. Among $u_0, u_1, \ldots, u_N$ we can pick out u_q and u_p such that $p < q$ and $u_q \equiv_R u_p \mod X^*$. We define now an infinite sequence of words $u'_0, u'_1, \ldots$ by putting

$$u'_i = u_i, \quad 0 \leq i \leq q-1$$

and

$$u'_{q+i} = u_{p+t}, \quad i = 0, 1, \ldots,$$

where t is the least nonnegative residue of $i \bmod q-p$.

It is easy to verify that

$$x'_i = u'_iu'_{i+1} \in X^*$$

for $i = 0, 1, 2, \ldots$ and

$$x' = wu'_0 = wu_0 \in X.$$

Consider now the infinite product $wu'_0u'_1 \ldots$ written in two ways

$$(wu'_0)(u'_1u'_2) \cdots = w(u'_0u'_1)(u'_2u'_3) \ldots$$

or

$$x_0x_1 \cdots = wy_1y_2 \ldots \tag{3}$$

with $x_0 \in X, |w| < |x_0|; x_i, y_j \in X^*$.

Analogously, if $\overline{V}_j \neq \emptyset$ for all $i \geq 0$, we have the equality

$$\ldots x_{-2}x_{-1}w = \ldots y_{-1}y_0, \tag{4}$$

where $y_0 \in X, |w| < |y_0|; x_i, y_j \in X^*$.

If now $w \in W \cap X$ and (i) does not hold, for instance, $\overline{U}_i \neq \emptyset$ for all i. Then (3) together with the obvious equality $\ldots ww = \ldots ww$ show that X is not a Z-code.

If $w \in W - X$ and (ii) does not hold, i.e. $\overline{U}_i, \overline{V}_j \neq \emptyset$ for all $i, j \geq 0$. Then (3) and (4) will give rise to two distinct factorizations on X of some bi-infinite word: X is not a Z-code and the theorem follows.

Example 4 We use Theorem 2 to show that the language $X = \{a, cab, c, bc^+d\}$ given in Example 3 is in fact a Z-code.

$$\begin{aligned}
W &= \{c, b\}, \\
\overline{U}_0(c, X) &= \{ab\}, \overline{U}_1(c, X) = c^+dX^*, \overline{U}_2(c, X) = \emptyset, \\
\overline{V}_0(c, X) &= \emptyset, \\
\overline{U}_0(b, X) &= c^+d, \overline{U}_1(b, X) = \emptyset.
\end{aligned}$$

Since $c \in W \cap X, b \in W - X, X$ is a Z-code.

In general Theorem 2 is not true for arbitrary languages, as shown in the following

Example 5 Consider $X = \{a^{i+2}ba^ib : i = 0,1,2,\ldots\} \cup \{ba^{2i+1}b : i = 0,1,2,\ldots\} \subseteq \{a,b\}^*$. Clearly, b is an overlap and for all $i \geq 0$, we have $ab \in \overline{U}_i(b,X)$, $a^{2(i+1)}b \in \overline{V}_i(b,X)$, i.e. $\overline{U}_i, \overline{V}_j \neq \emptyset$ for all $i,j \geq 0$, but a simple verification ensures that X is a Z-code.

We should mention two other algorithms to verify whether a regular code X is a Z-code. Both of them consist in checking the emptiness problem for some automata (Devolder and Timmerman [4], Beal [2]) that has as well known a polynomial time complexity in the number of states of automata.

Using Theorem 2 we give alternative proofs of the results of M.P. Beal and A. Restivo. First, we prove

Corollary 1 (M.P. Beal [1]) *Let X be a regular code. Then X is a Z-code if and only if it is a circular code.*

Proof. First, observe that if X is a code then

(1) for any $w \in W \cap X$: $\overline{U}_i(w,X) \cap X^* = \emptyset$ and $\overline{V}_i(w,X) \cap X^* = \emptyset$ for all $i = 0,1,2,\ldots$;

(2) for any $w \in W - X$: $\overline{U}_i(w,X) \cap X^* = \emptyset$ or $\overline{V}_i(w,X) \cap X^* = \emptyset$ for all $i = 0,1,2,\ldots$

that are trivially to be verified using Lemma 1 or its symmetrical version.

Let now X be a regular circular code, hence a code: (1) and (2) are satisfied.

Suppose that for some $w \in W \cap X$ we have, say, $\overline{U}_i \neq \emptyset$ for all $i = 0,1,2,\ldots$. For any $N \geq 0$ there exist $u_0, u_1, \ldots, u_{N-1}, u_N$ such that $u_1 \in u_0^{-1}X^*, \ldots, u_N \in u_{N-1}^{-1}X^*$. Since X^* is of finite right index m, if we take N sufficiently large, we can find $i,j : 0 \leq i < j$, such that $u_i^{-1}X^* = u_j^{-1}X^*$ and $j-1$ is even. Consider the words

$$u = u_{i+1}\ldots u_j, \quad v = u_{i+2}\ldots u_{j-1},$$

it follows $u_j u_{i+1} \in X^*, v \in X^*$ and $u = u_{i+1} v u_j \in X^*$. By circularity of X we get $u_j, u_{i+1} \in X^*$, in particular, $u_j \in \overline{U}_j \cap X^* \neq \emptyset$ contradicting (1). Therefore for any $w \in W \cap X$ we have $\overline{U}_i = \emptyset$ for some i and analogously $\overline{V}_j = \emptyset$ for some j.

As for any $w \in W - X$, by the same way, we can conclude that either $\overline{U}_i = \emptyset$ for some i or $\overline{V}_j = \emptyset$ for some j.

By virtue of Theorem 2, X is a Z-code. The proof is completed.

We now deduce another statement concerning codes with bounded synchronization delay. Recall that a subset X of A^* is said to be a *code with bounded synchronization delay* provided it is a code and for some integer $p \geq 0$, for all $u,v \in X^p$, and for all $g,f \in A^*$,

$$gu, vf \in X^*$$

whenever

$$guvf \in X^*.$$

The least number p satisfying this condition is the *synchronization delay* of X. The fact that every code with bounded synchronization delay is a Z-code is obvious, but the reverse conclusion is not always valid. A lot of interesting properties of these codes have been discovered, for example, in the finite case, these codes are exactly the very pure codes, i.e. circular codes (see [11], [12]). We have the following

Corollary 2 (A. Restivo [11]) *Let X be a regular subset of A^+, X is a code with bounded delay if and only if it is a Z-code satisfying $A^*X^dA^* \cap X = \emptyset$ for some positive integer d.*

Proof. "Only if" part: first, the fact that each code with bounded synchronization delay is a Z-code is easy. Further, we show that $A^*X^dA^* \cap X = \emptyset$ for all d exceeding the right index of X. Suppose on the contrary that

$$ux_1 \dots x_d v \in A^*X^dA^* \cap X$$

for some $x_1, x_2, \dots, x_d \in X$ and $u, v \in A^*$. Then, indeed, there exist i and $j, i < j \leq d$, such that $ux_1 \dots x_i \equiv_R ux_1 \dots x_j \bmod X$ which implies that for all $k = 0, 1, 2, \dots$:

$$ux_1 \dots x_i (x_{i+1} \dots x_j)^k \equiv_R ux_1 \dots x_i (x_{i+1} \dots x_j)^{k+1} \bmod X$$

and consequently

$$ux_1 \dots x_i (x_{i+1} \dots x_j)^k x_{j+1} \dots x_d v \in X.$$

Hence the synchronization delay of X cannot be bounded.

Conversely, let X be a regular Z-code and $A^*X^dA^* \cap X = \emptyset$ for some positive integer d, hence $d \geq 2$. By Theorem 2, for all overlaps $w \in W, \overline{U}_m(w, X) = \emptyset$ or $\overline{V}_e(w, X) = \emptyset$, where m and e are the right and left index of X^*, respectively. We show that X is of bounded synchronization delay not greater than $p = (m+1)d$ (the value in [11] is $2(m+1)d$). If that is not so, there must exist some words $g, h \in A^*, x_1, \dots, x_p, x_{p+1}, \dots x_{2p}, y_1, \dots, y_q \in X$ such that

$$gx_1 \dots x_p x_{p+1} \dots x_{2p} h = y_1 \dots y_q \tag{1}$$

and for all $k = 1, 2, \dots, q$

$$gx_1 \dots x_p \neq y_1 \dots y_k.$$

Thus, it has to exist a unique positive integer $l \leq q$ such that

$$y_1 \dots y_{l-1} < gx_1 \dots x_p < y_1 \dots y_l$$

and the largest positive integer $i \leq p-1$ and the smallest positive integer $j \geq p+1$ satisfying

$$gx_1 \dots x_i \leq y_1 \dots y_{l-1} < gx_1 \dots x_p < y_1 \dots y_l \leq gx_1 \dots x_j \tag{2}$$

(abusing language, we write for words $x, y, x \leq y, x < y$ to indicate that x is a prefix, a proper prefix of y, respectively). Since $y_l \notin A^*X^dA^*, j \leq d+p$ and $i \geq p-d$.

Further, if in (2) $gx_1 \dots x_i = y_1 \dots y_{l-1}$ and $gx_1 \dots x_j = y_1 \dots y_l$ then

$$y_l = x_{i+1} \dots x_j, \quad j - i \geq 2$$

that is a contradiction with the fact that X is a code.

Alternatively, assume that $gx_1 \dots x_j \neq y_1 \dots y_l$ which gives rise to

$$gx_1 \dots x_{j-1} w = y_1 \dots y_l, \tag{3.1}$$

$$x_j x_{j+1} \dots x_{2p} h = w y_{l+1} \dots y_q, \tag{3.2}$$

where $w \in W$ and $|w| < |y_l|, |w| < |x_j|$. Similarly, the case $gx_1 \dots x_i \neq y_1 \dots y_{l-1}$ gives rise to

$$gx_1 \dots x_{i+1} = y_1 \dots y_{l-1} w, \tag{4.1}$$

$$w x_{i+2} \dots x_{2p} h = y_l y_{l+1} \dots y_q, \tag{4.2}$$

where $w \in W$ and $|w| < |y_l|, |w| < |x_{i+1}|$.

We will show that (3.2) or (4.2) equally leads to $\overline{U}_{2m}(w, X) \neq \emptyset$ and (3.1) or (4.1) – to $\overline{V}_k(w, X) \neq \emptyset$ with k abitrarily large, in particular $k \geq e$ that is quite a contradiction.

First, suppose that we have (3.2), setting

$$u_1 = x_{j+1} \dots x_{j+d}, \dots, u_m = x_{j+(m-1)d+1} \dots x_{j+md}$$

and let $q(k)$ the smallest integer such that for $k = 0, 1, \dots, m$

$$x_j u_1 \dots u_k \leq w y_{l+1} \dots y_{q(k)} \tag{5}$$

(for compactness, we set by convention that $x_j u_1 \dots u_k = x_j$ when $k = 0$). Since $A^* X^d A^* \cap X = \emptyset, w < x_j$ and $u_1, \dots, u_m \in X^d$, it follows $l+1 \leq q(0) < q(1) < \dots < q(m)$. Putting $v_k = y_{l+1} \dots y_{q(k)}, k = 0, 1, 2, \dots, m$, by (5) and $A^* X^d A^* \cap X = \emptyset$ we get

$$x_j u_1 \dots u_k \leq w v_k < x_j u_1 \dots u_{k+1} \tag{6}$$

for $k = 1, 2, \dots, m-1$ and

$$w v_{k-1} \leq x_j u_1 \dots u_k \leq w v_k \tag{7}$$

for $k = 1, \dots, m$.

It is easy to verify that (6), (7) together with $w < x_j$ yield

$$(w v_0)^{-1}(x_j u_1) \in \overline{U}_2, \dots, (w v_{m-1})^{-1}(x_j u_1 \dots u_m) \in \overline{U}_{2m},$$

i.e. $\overline{U}_{2m} \neq \emptyset$.

Likewise, since $i + 2 \leq j$, (4.2) leads also to $\overline{U}_{2m} \neq \emptyset$.

Now, as far as $\overline{V}_e$ is concerned, we treat (3.1) and (4.1) as above, only in the symmetrical way. Directly, (3.1) or (4.1) cannot lead to $\overline{V}_e \neq \emptyset$, but we can "pump" them up to some equalities "long" enough by proceeding as follows. Suppose, for example, that we have (4.1). Among $m + 1$ numbers $1, d + 1, \dots, md + 1$ there must exist a, b such that $gx_1 \dots x_a \equiv_R gx_1 \dots x_b \bmod X^*$ with $a < b$. Note that $b - a \geq d \geq 2$ and $a, b \leq md + 1 \leq p - d + 1 \leq i + 1$. Further, for some integer $s \leq t \leq l$ we must have

$$\begin{aligned} y_1 \dots y_{s-1} u_a &= gx_1 \dots x_a, \\ gx_1 \dots x_a v_a &= y_1 \dots y_s, \\ u_a v_a &= y_s \end{aligned}$$

and

$$\begin{aligned} y_1 \dots y_{t-1} u_b &= g x_1 \dots x_b, \\ g x_1 \dots x_b v_b &= y_1 \dots y_t, \\ u_b v_b &= y_t, \end{aligned}$$

where $u_a, v_a, u_b, v_b \in A^*$. Hence $x_{a+1} \dots x_{i+1} \in v_a X^* w$. From $gx_1 \dots x_a \equiv_R gx_1 \dots x_b \bmod X^*$ it follows

$$g x_1 \dots x_a \equiv_R g x_1 \dots x_a (x_{a+1} \dots x_b)^k \bmod X^*$$

for all $k = 0, 1, 2, \dots$. Since $gx_1 \dots x_a v_a \in X^*$ we have $gx_1 \dots x_a(x_{a+1} \dots x_b)^k v_a \in X^*$. Therefore

$$g x_1 \dots x_a (x_{a+1} \dots x_b)^k x_{a+1} \dots x_{i+1} \in X^* w, \tag{8}$$

where, as before, $|w| < |x_{i+1}|$.

Looking into (8) we see that the left-hand side of (4.1) is pumped up by a product of $k(b - a - 1)$ words. We take k large enough to obtain a sufficiently "long" equality of the form (4.1). Now proceeding as is done for $\overline{U}_{2m}$, we conclude that $\overline{V}_e$ is nonempty. This contradiction with Theorem 2 completes the proof.

The regularity condition is essential for Theorem 3 to be valid. Indeed, consider the following

Example 6 The Z-code $X = \{a^{i+1} b a^i b : i = 0, 1, 2, \dots\} \subseteq \{a, b\}^*$ is not a regular language. It is not a code with bounded synchronization delay, although $A^* X^2 A^* \cap X = \emptyset$.

Concluding, from [8] or [1] we deduce the following statement.

Theorem 3 *Let $X = \{x, y\}(|x| > |y|)$ be a two-word language of A^* then X is not a Z-code if and only if one of the following assertions holds*

(i) x or y is imprimitive;

(ii) x and y are conjugate;

(iii) xy^n is imprimitive for some positive integer $n < \frac{|x|}{|y|} + 1$;

(iv) $x^2 y$ is a square.

Proof. Obviously, if one of (i)-(iv) holds, X is not a Z-code.

Conversely, suppose that X is not a Z-code (thus not a circular code, not a very pure code) and besides x and y are primitive and not conjugate. We show that (iii) or (iv) must occur.

Indeed, by [8] or [1], $x^* y \cup x y^*$ contains an imprimitive word $u = v^m, m \geq 2$:

– if $u = xy^n$ then $(n-1)|y|$ cannot excceed $|v| - 1$ otherwise by Fine and Wilf Theorem (see [9] or [5]) x and y are copower that contradicts the assumption. Thus $(n-1)|y| < |v|$, or $2(n-1)|y| < 2|v| \leq |x| + n|y| = |xy^n|$ i.e. $|n| < \frac{|x|}{|y|} + 1$;

– if $u = x^n y = v^m$ we can suppose $n \geq 2$. Further, if the inequality

$$m \geq \frac{n+1}{n-1}$$

holds, then $m(n-1)|x| \geq (n+1)|x| \geq n|x|+|y| = m|v|$. Therefore, $(n-1)|x| \geq |v|$, or, $n|x| \geq |x|+|v|$. Again by Fine and Wilf Theorem x, v and thus x, y are copower that contradicts the assumption. So, we always have $m < \frac{n+1}{n-1}$. Since $m, n \geq 2$ it follows $m = n = 2$.

Acknowledgements The authors acknowledge their great indebtedness to the referees for their valuable comments and suggestions.

References

[1] E. Barbin Le Rest and M. Barbin Le Rest, Sur la combinatoire des codes à deux mots, *Theoretical Computer Science* 41 (1985), 61-81.

[2] M.P. Beal, "Codages, automates locaux et entropie," Publications du LITP (Paris), No. 38 (1988).

[3] J. Berstel and D. Perrin, "Theory of Codes", Academic Press, New York, 1985.

[4] J. Devolder and E. Timmerman, "Codes for Bi-infinite Words", Publications du LIFL (Lille), No I.T. 16 (1990).

[5] N.J. Fine and H.S. Wilf, Uniqueness Theorems for Periodic Functions, *Proceedings of the American Mathematical Society* 16 (1965), 109-114.

[6] S. Golomb and B. Gordon, Codes with Bounded Synchronization Delay, *Information and Control* 8 (1965), 355-372.

[7] G. Lallement, "Semigroups and Combinatorial Applications," John Wiley, New York, 1979.

[8] A. Lentin and M.P. Shützenberger, A Combinatorial Problem in the Theory of Free Monoids, "Combinatorial Mathematics and Its Applications," Proceedings of the Conference held at the University of North Carolina (R.C. Bose and T.A. Dowlings, eds.), Chapell Hill, pp. 128-144.

[9] M. Lothaire, "Combinatorics on Words," Addison-Wesley, Reading, Massachusetts, 1983.

[10] Nguyen Huong Lam and Do Long Van, On a Class of Infinitary Codes, *Theoretical Informatics and Applications* 24 (1990), 441-458.

[11] A. Restivo, A Combinatorial Property of Codes Having Finite Synchronization Delay, *Theoretical Computer Science* 1 (1975), 95-101.

[12] A. Restivo, On a Question of McNaughton and Papert, *Information and Control* 25 (1974), 93-101.

Received May 9, 1992

Market Oriented Integration of MS-Windows-Based Tools for Distributed Decision Support*

M. Biró[†] P. Danyi[‡] P. Gelléri[‡]

Abstract

In this paper, we discuss the meaningfulness of value added systems integration for distributed decision support from a market oriented primary perspective. The issues to be analysed are derived from all pairwise interrelationships of the entities involved in a decision situation. These are the task logic, the decision culture, and the decision environment. Keeping these considerations in focus, we summarize experiments with commercially available products for the Microsoft Windows environment which is undisputably the most popular operating environment for personal computers.

Keywords: Systems integration, Decision support systems, Model design, Distributed decisions.

1 Introduction

The purpose of this paper is to give a structured guide to the design of distributed decision support systems. Since our primary objective is the supply of the market, we are concentrating on Microsoft Windows based tools which can be used on the most popular type of personal computers worldwide.

Our approach is derived from practical experiences in building and installing decision support systems to orders. One of our recent observations is that users are less keen on accepting a clever but custom made software tool than well established commercial products. We also see however, that commercial products alone are most of the time inappropriate for the support of specific decision circumstances. Our answer is *value added systems integration.*

The universal validity of our conclusion is supported by the report of a coloquium held by the U.S. Computer Science and Telecommunications Board, the Commission on Physical Sciences, Mathematics, and Applications, and the National Research Council in 1991. There, "systems integration was identified as a

*Supported by OTKA grants No. 2571 and 2575.

†Computer and Automation Institute, Hungarian Academy of Sciences, Budapest, Kende u. 13-17. H-1111 Hungary

‡Dept. of Information Engineering, Technical University of Budapest, Budapest, Műegyetem rkp. 3. H-1111 Hungary

large and rapidly growing market in which the United States was a clear leader" [1] [Keeping the U.S. Computer Industry Competitive... 1992].

In this paper, we are not going into the details of systems integration issues in general. We are rather concentrating on the structuring of ideas based on our practical experiences in building and installing decision support systems and our pioneering role in introducing object- oriented windows based software and decision support technology in Hungary [2], [3].

2 A Model for Mapping Decision Situations

A DSS must always refer to the particular decision situation. However, decision situations are not only determined by the decision problem itself, but also by the problem owner and the available decision techniques. Let us formulate a model which, according to our experiences, provides an appropriate guidance for our analysis (Figure 1.). A DSS stands in the intersection of the entities of the basic model which means that a DSS can only be built if we bring together the contexts of these entities.

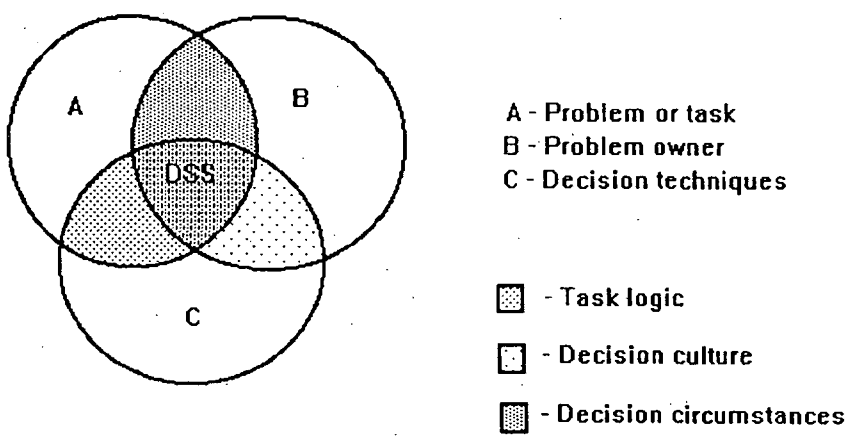


Figure 1.

Examples could be brought from an infinitely wide range of areas including money allocation, tender evaluation, personnel selection. Let us consider the following specific example. A DSS is being designed for managing *catastrophe situations* in a power plant. The system must not only contain decision techniques in themselves, as e.g. MCDM, fuzzy logic or AHP. Decision models should be set up concerning

- different kinds of problem (or task) situations, e.g. earthquake, computer virus, etc...
- problem owners with different levels of decision authority ranging from a guard to the president.

The entities do never occur apart but in a colorful amalgamation which we are interested in. Let us consider the intersections of all pairs of entities:

Task logic. The intersection of decision technique and problem (or task) is related to the abstract *types of decision problems* which reflects different decision models and have logically different solution algorithms. The most typical task logics are as follows: selection among discrete (well defined) alternatives, task monitoring, resource allocation, etc. It is obvious that any DSS supports some of the possible task logics but not all of the logics. Different DSSs must be built for the catastrophe example in the different warning phases with dissimilar levels of danger.

Decision culture. The intersection of problem owner and decision technique is related to the decision culture. This means the problem owner's experience in solving decision problems that is capability of using various kinds of decision methods and tools, and his skill level at using them. More specifically, e.g. some people prefer using probabilities, others do odds or utilities. On the other hand, some people are risk-averse, some are risk-prone. Japanese and American managers hardly ever look similarly at the very same problem. What kind of presumptions can we have about cultures? First, some people may be homogeneous as far as their decision thinking is concerned. Second, if they think differently, classes must be defined. Our goal is to help the problem owner in finding his real role i.e. his class.

Decision circumstances. Finally, let us consider the concept of decision circumstances, which is related to the intersection of problem and problem owner in the model. First, decision circumstances include the constraints and goals of the problem owner together with his or her attitude to the task. This also means time and resource constraints, and considerations coming from personal interests on the other hand. Second, the environment of the given problem has a huge influence on the design of the DSS. Some of the important issues are the individual or group nature of the decision environment, the chance for a compromise in the group case, the equality or inequality of voting powers, etc.

3 The MS-WINDOWS Based Toolkit Approach

In our opinion the most effective way of building a DSS satisfying particular requirements is using a toolkit. When we build a DSS from parts, we can excellently track the needs of the user, the environment, etc. In addition, the toolkit approach provides some technical advantages:

- modularity
- all of the pieces are exchangeable
- interfaces between units must be precisely elaborated.

The tools that we shall inspect are commercially available products for the Microsoft Windows operating environment. The interface between the units is naturally provided by DDE (Dynamic Data Exchange) and OLE (Object Linking and Embedding) which are defined in general within the environment.

However, while the above features significantly facilitate systems integration, we have to extend the commercial tools with new capabilities in order to supporting specific decision circumstances. These extensions will be highlighted below as well.

Tools covering significant task logics

The tools that can be mentioned here must include at least group scheduling capabilities which are necessary for monitoring the group decision making process and for allocating the necessary resources. There are many Windows based products belonging to this category. One of them is Schedule+ included with Windows for Workgroups and the future Windows NT as well.

Windows for workgroups has another important characteristic from the task logic point of view, which differentiates it from other groupware tools like Lotus Notes available today. It supports peer-to-peer networking with network dynamic data exchange facility as opposed to the client- server paradigm inherent to other tools. This feature opens new possibilities for distributed decision support where each personal computer on the network can operate both as a client and a server, obviating the need for a dedicated server. These networks are not only inexpensive but also easy to set up. A useful exploitation of this technology for distributed negotiation support (DINE) is described in [4]. This application was based on a prototype network dynamic data exchange facility developed with the participation of one of the authors one year before the release of the commercial Microsoft tool.

Tools covering significant decision cultures

Experts participating in a distributed decision making process may have different professional backgrounds which basically determine their decision culture. Different professional backgrounds implie that their professional cognitive patterns are different as well. A tool supporting distributed decision making must provide support for each individual expert and for the group as a whole. Thus, the model representations offerred by the system must be appealing to all of the participants, which implies that they must be as close as possible to everyday cognitive patterns. Tabular (relational) representations in *spreadsheets* for example satisfy this requirement, since tables are incorporated among our cognitive patterns at the elementary school level. This is in fact the fundamental reason of their general success [5].

Spreadsheet products for Windows are numerous again. They include Lotus 1-2-3, Borland Quattro, and Microsoft Excel.

The already mentioned application (DINE) [4], [6] is based on Microsoft Excel, which was extended with several features in order to accomodating experts from various professional backgrounds and still providing a high level of decision support. These features include optimisation and multiple criteria decision making capabilities in an environment where dynamically changing data originating from shared data bases or other members of the decision making group are permanently taken into account.

Tools covering significant decision circumstances

Groupsystems and Lotus Notes are commercial tools that are relevent to different decision circumstances. Groupsystems provides anonymous, real-time interaction with the help of a facilitator in an electronic meeting room. Lotus Notes provides workgroup electronic mail, distributed databases, bulletin boards, document management, etc... in an environment distributed in time and space.

It is a characteristic property of DINE that it provides integrated support for both the group as a whole and the individual user while privately evaluating the positions of other group members. This support is independent on the cooperative or competitive nature of the decision circumstances.

4 DINE

The DINE model supports simultaneous, multiple issue, independent peer-to-peer negotiations. It allows the integration of existing negotiation support techniques which, as opposed to DINE, mostly focus on scenarios where the negotiation issues are shared by all negotiators. The latter techniques are used to support the independent peer- to-peer negotiations in DINE. Negotiators may in fact use any tool even without DINE, as long as it supports the same peer-to-peer information sharing protocol. At the same time, DINE is a generalized multiple criteria decision making model where the alternatives to be ranked are compound subsets of negotiated offers. DINE naturally integrates asynchronous and synchronous communication requirements, intuitive judgement and deep knowledge based techniques. The implementation is based on the Microsoft Windows environment and some of its value added features have already been mentioned.

Our objective here is the critical description of the value- added features related to model-based deep knowledge generation wich bring the Microsoft Excel commercial tool closer to a wide range of task logics, decision cultures and decision circumstances.

The cunstruction of models in general is well supported in spreadsheet environments. There is even integrated support for the specification and solution of optimization models within a spreadsheet (What's Best, IFPS/Optimum, Microsoft Excel Solver). The advantages of such systems over algebraic languages have been analysed in detail [8], [18], we will not go into these issues here.

What difficulties do arise however with existing tools and what kind of further support can be provided for optimization modeling and model experiments in spreadsheets which can improve their scope of usability? Let us list some of these below.

1. The first problem is that while changing most parameters of the model is natural and easy, *changing the size of the model* involves spreadsheet manipulations which are error prone and external to the world of the model itself.

2. The second problem is also related to the size of the model. There are two major reasons why *large models* are increasingly difficult to handle with spreadsheets. The first reason is *memory limitation* which is a question of money and technology scaling only. The other reason is our *cognitive limitation*. The power of the spreadsheet in visualizing data relationships may decrease with larger models unless appropriate data are stored in relational databases and the display structures of the model are carefully chosen in the beginning.

3. The third problem is that existing spreadsheet model building schemes are essentially algebraic which means that a transformation of real world objects and relationships into *algebraic entities and expressions* is necessary. A remarkable possibility for integrating iconic and other representation schemes including spreadsheets is described in [14]. This issue is not discussed any further in this paper, it will be the subject of a further study.

The purpose of the meta-model building tool in DINE is the provision of relief to the first two difficulties above. The solutions provided by DINE are best illustrated in the light of an example.

An example

The example is a simple multiperiod investment problem similar to the one provided as a sample application for Microsoft Excel Solver. The point is not on the validity of the assumptions, but on the new spreadsheet representation and underlying meta-model building tool which solves the first two problems above.

Determine how to invest cash into certificates of deposit (CD) with fixed interest rate and fixed term, so as to maximize interest income while meeting given periodical cash requirements (plus a safety margin). The algebraic formulation of this problem is a typical textbook exercise. The spreadsheet formulation provided as a sample application for Microsoft Excel Solver has its advantages, however it strongly suffers from the above listed difficulties. The DINE approach will preserve the advantages, while resolving the problems.

The *primary concepts* that appear to be necessary for the formulation of the model are the following:

- Date
- Cash requirement
- CD
- Interest
- Term
- Investment

These concepts will be extended during meta-model building with a few secondary quantities which contribute to a better visualization of the data relationships.

The meta-model building tool

The quantities in our example which are appropriate for database storage are the cash requirements with the corresponding dates (a private database) and the CD's with their interest rates and terms (public database). The decision variables are clearly the amounts invested into different CD's at the specified dates (Investment).

CD	Interest rate	Term

Date	Cash requirement

The meta-model of the problem is placed into the first line of a table whose field headings are the primary concepts and some further interesting secondary quantities. On request, our macros interpret the meta-model and replace the line with a table which is then the final model still hot linked to the underlying databases and automatically responding to any intuitive or optimization based changes.

The purpose of the meta-model is the definition of the way the actual model will be automatically built as soon as the underlying databases are available and the user requests it. The meta-model by consequent is independent on the sizes of any databases which determine the size of the model itself, it depends however on the fields of those databases.

The functional decomposition of the model into databases and meta-model provides a solution to the first problem above. It allows an easy reconstruction of the model any time the size of any database changes. The use of the relational database paradigm means a solution to the second problem (keeping a clear view of relationships) from the side of the primary data the model refers to. The databases may even reside on remote servers.

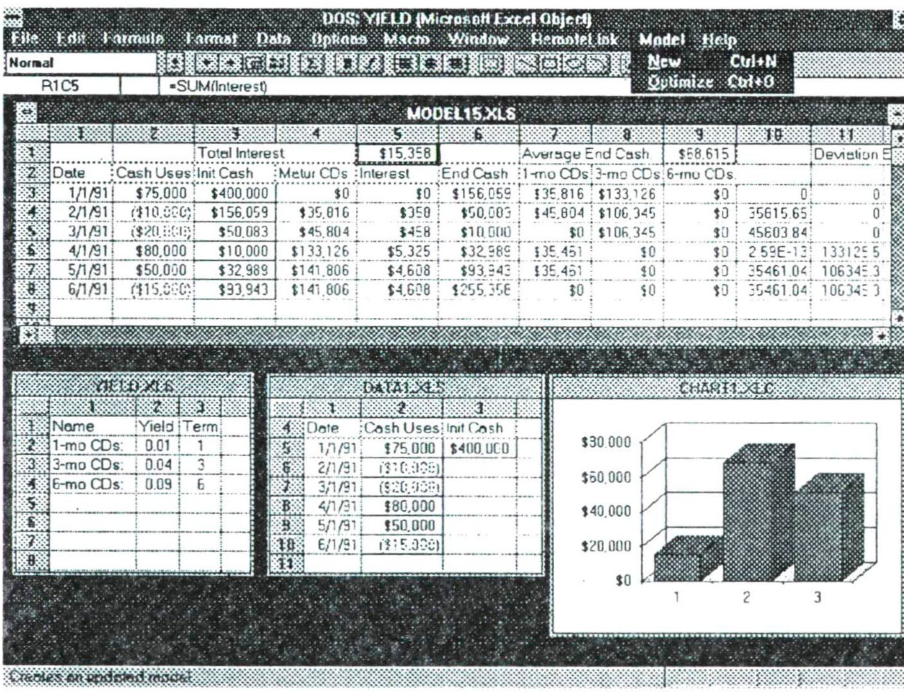


Figure 2. Model, underlying databases, and chart showing model characteristics.

The solution to the second problem from the side of the model, that is keeping a clear view of relationships within the model, is a question of careful design of the model structure in the spreadsheet, and of the most useful decomposition of calculations into secondary result tables. The secondary result tables should in particular include quantities which will serve as constraints to the optimization problem, and should at the same time be useful for the evaluation of the effect of intuitive changes made with the decision variables. From the technical point of view, the primary and secondary result tables have to be defined in such a way that the same spreadsheet formula can provide all required quantities in any given column of the table when the meta-model is expanded into the final model.

5 CONCLUSION

In this paper, we gave a structured guide to the design of distributed decision support systems from a market oriented perspective. We concentrated on Microsoft Windows based tools which can be used on the most popular type of personal computers worldwide.

We illustrated the power of value added systems integration with new features incorporated into a prototype distributed negotiation support application exploiting the advanced capabilities of the Microsoft Excel commercial spreadsheet environment.

References

[1] L.A.Belady: Software Engineering: Beyond Software and Beyond Engineering. In: Shifting Paradigms in Software Engineering (ed. R. Mittermeir). (Springer-Verlag, Wien, New York, 1992) pp.2.

[2] Hernádi,A; Biró,M.; Horváth,T.; Hutter,O.; Király.L.; Knuth,E.; Remzső,T. Window Systems. (Typotex Kft, Budapest, 1990). (in Hungarian)

[3] Biró,M.; Csáki,P.; Vermes,M. WINGDSS Group Decision Support System under MS-Windows. In: Proceedings of the Second Conference on Artificial Intelligence (ed. by I.Fekete and P.Koch). (John von Neumann Society for Computer Sciences, Budapest, Hungary, 1991) pp.263-274.

[4] M. Biró, E. Bodroghy, A. Bor, E. Knuth, L. Kovács, The Design of DINE: a DIstributed NEgotiation Support Shell. In: Decision Support Systems: Experiences and Expectations, (ed. by T. Jelassi, M.R. Klein, W.M. Mayon-White). IFIP Transactions A-9 (North- Holland, 1992) pp.103-114.

[5] M. Biró and I. Maros, The Use of Deep Knowledge from the Perspectives of Cooperative Problem Solving, Systems Modeling, and Cognitive Psychology. In: Shifting Paradigms in Software Engineering (ed. R. Mittermeir). (Springer-Verlag, Wien, New York, 1992) pp.56- 67.

[6] M. Biró, A. Bor, E. Knuth, T. Remzso, A. Szilléry: Spreadsheet-Based Model Building and Multiple Criteria Group Decision Support, MTA SzTAKI Working Paper, 1993.

[7] J. Bisschop and A. Meeraus, On the Development of a General Algebraic Modeling System in a Strategic Planning Environment, Mathematical Programming Study, 20(1982)1-20.

[8] S.E. Bodily, Spreadsheet Modeling as a Stepping Stone, Interfaces, 16, no.5(1986)34-52.

[9] J.M. Bowers and S.D. Benford, Studies in Computer Supported Cooperative Work, North- Holland, 1991.

[10] Brookes, C.H.P. (1986) Requirements Elicitation for Knowledge Based Decision Support Systems, in:McLean, E.R. and Sol, H.G.(eds): Decision Support Systems: A Decade in Perspective, Elsevier Science Publ. B.V. (North-Holland)

[11] T.X. Bui, Co-oP - A Group Decision Support System for Cooperative Multiple Criteria Group Decision Making. Lecture Notes in Computer Science 290. Springer-Verlag, 1987.

[12] S.I. Gass, Model World: Danger, Beware the User as Modeler, Interfaces, 20, no.3(1990)60-64.

[13] Gelleri, P. and Martinez, F. (1988): How to Handle Differences in Importance among Participants with GDSS, in: Lee, R.M., McCosh, A.M. and Migliarese, P.: Organizational Decision Support Systems, North-Holland.

[14] J. Gerlach and F. Kuo, An Approach to Dialog Management for Presentation and Manipulation of Composite Models in Decision Support Systems, Decision Support Systems, 6(1990)227-242.

[15] Holt, C.C. Conceptual Environment for Organization Support Systems: Design and Use of Information Technology in Organization, in:Conf. 'Environments for Supporting Decision Processes', Budapest, Hungary, 18-21 lune, 1990.

[16] Huber, G.P. (1984): Issues in the Design of Group Decision Support Systems, MIS Quarterly, 8.pp.195- 204.

[17] T. Hürlimann, LPL, A Structured Language for Modeling Linear Programs (Verlag Peter Lang AG, Bern, 1987). Keeping the U.S. Computer Industry Competitive: Systems Integration. (National Academy Press, Washington, D.C., 1992).

[18] A. Roy, L. Lasdon and D. Plane, End-user optimization with spreadsheet models, European Journal of Operational Research, 39(1989)131-137.

[19] Sol, H.G. Information Systems to Support Decision Processes, in:Conf. 'Environments for Supporting Decision Processes', Budapest, Hungary, 18-21 lune, 1990.

[20] Yazdani M. Shells Versus Toolkits. Expert System User, January 1989, pp.13.

Received June 28, 1993

Készítette a JATEPress
6722 Szeged, Petőfi Sándor sugárút 30—34.
Felelős vezető: Szőnyi Etelka
Méret: B/5, példányszám: 350, munkaszám: 27/94.

Subscription information and mailing address for editorial correspondence:

Acta Cybernetica
Árpád tér 2.
Szeged
H-6720 Hungary

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Gécseg Ferenc
A kézirat a nyomdába érkezett: 1993. december
Terjedelem: 7,12 (B/5) ív