Frequent Itemset Mining in Multirelational Databases

Aída Jiménez, Fernando Berzal, and Juan-Carlos Cubero

Dept. Computer Science and Artificial Intelligence,
ETSIIT - University of Granada, 18071, Granada, Spain
{aidajm,jc.cubero,fberzal}@decsai.ugr.es

Abstract. This paper proposes a new approach to mine multirelational databases. Our approach is based on the representation of a multirelational database as a set of trees. Tree mining techniques can then be applied to identify frequent patterns in this kind of databases. We propose two alternative schemes for representing a multirelational database as a set of trees. The frequent patterns that can be identified in such set of trees can be used as the basis for other multirelational data mining techniques, such as association rules, classification, or clustering.

1 Introduction

Data mining techniques have been developed to extract potentially useful information from databases. Classification, clustering, and association rules have been widely used. However, most existing techniques usually require all the interesting data to be in the same table.

Several alternatives have been proposed in the literature to handle with more than one table. There are algorithms that have been developed in order to explore tuples that, albeit in the same table, are somehow related [1] [2]. Other algorithms have been devised to extract information from multirelational databases, i.e., taking into account not only a single table but also the tables that are related to it [3]. For instance, these algorithms have been used for classification [4] and clustering [5] in multirelational databases.

In this paper, we propose two alternative representations for multirelational databases. Our representation schemes are based on trees, so that we can apply existing tree mining techniques to identify frequent patterns in multirelational databases. We also compare the proposed representation schemes in order to determine which one is better to use depending on the information we want to obtain from the database.

Our paper is organized as follows. We introduce some standard terms in Section 2. Section 3 presents two different schemes for representing multirelational databases using trees. We explain the kind of patterns that can be identified in the trees derived from a multirelational database in Section 4. We present some experimental results in Section 5 and, finally, we end our paper with some conclusions in Section 6.

2 Background

We will first review some basic concepts related to labeled trees before we address our multirelational data mining problem.

A **tree** is a connected and acyclic graph. A tree is rooted if its edges are directed and a special node, called root, can then be identified. The root is the node from which it is possible to reach all the other nodes in the tree. In contrast, a tree is said to be free if its edges have no direction, that is, when it is an undirected graph. A free tree, therefore, has no predefined root.

Rooted trees can be classified as **ordered trees**, when there is a predefined order within each set of sibling nodes, or **unordered trees**, when there is not such a predefined order among sibling nodes. textbfPartially-ordered trees contain both ordered and unordered sets of sibling nodes. They can be useful when the order within some sets of siblings is important but it is not necessary to establish an order relationship within all the sets of sibling nodes.

Figure 1 shows an example dataset with different kinds of rooted trees. In this figure, ordered sibling nodes are joined by an arc, while unordered sets of sibling nodes do not share an arc.

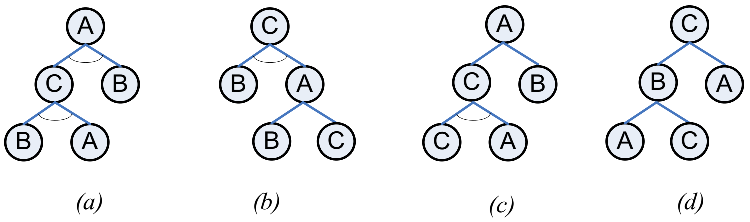


Fig. 1. Example dataset with different kinds of rooted trees (from left to right): *(a)* completely-ordered tree, *(b) (c)* partially-ordered trees, *(d)* completely-unordered tree

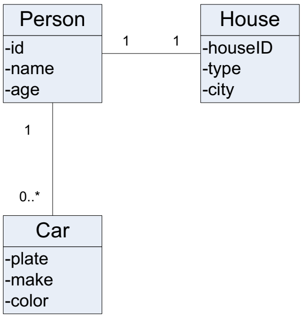
3 Tree-Based Multirelational Database Representation

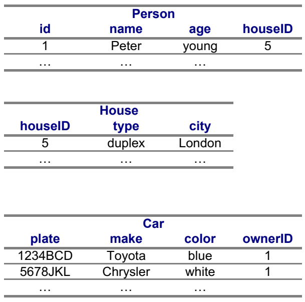
Multirelational data mining techniques look for patterns that involve multiple relations (tables) from a relational database [3].

The schema of a multirelational database can be represented using an UML diagram [6]. Figure 2 shows the UML diagram for a multirelational database as well as its tables with some example tuples.

We will call target relation (or target table) to the main relation in the multirelational database. Let x be a relation, A be an attribute belonging to this relation, and a the value of the attribute. We will use the notation $x.A = a$ to represent the node which contains the value a for the attribute A in the relation x.

We present two different schemes for representing multirelational databases as sets of trees. The main idea behind both of them is building a tree from each tuple in the target table and following the links between tables (i.e. the foreign keys) to collect all the information related to each tuple in the target table. In both representation schemes, the root of all trees will be the name of the target table.




Person

id	name	age	houseID
1	Peter	young	5
...	...	...	

House

houseID	type	city
5	duplex	London
...	...	...

Car

plate	make	color	ownerID
1234BCD	Toyota	blue	1
5678JKL	Chrysler	white	1
...	...	...	

Fig. 2. MultiRelational database example

3.1 Key-Based Tree Representation

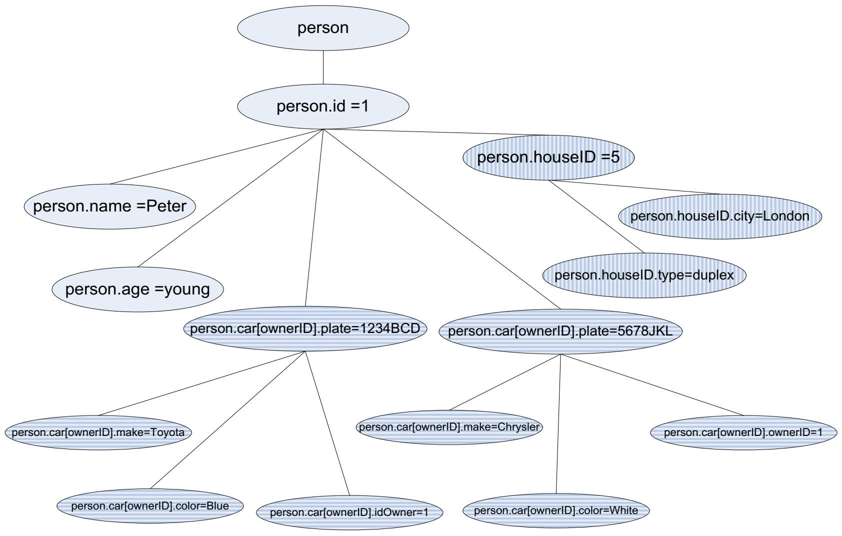
The key-based tree representation scheme represents all the attribute values within a tuple as children of its primary key.

As a consequence, the root node of the tree representing a tuple in the target relation will have, as its unique child, the value of the primary key of the tuple that the tree represents. The children of this primary key node will be the remaining attribute values in the tuple.

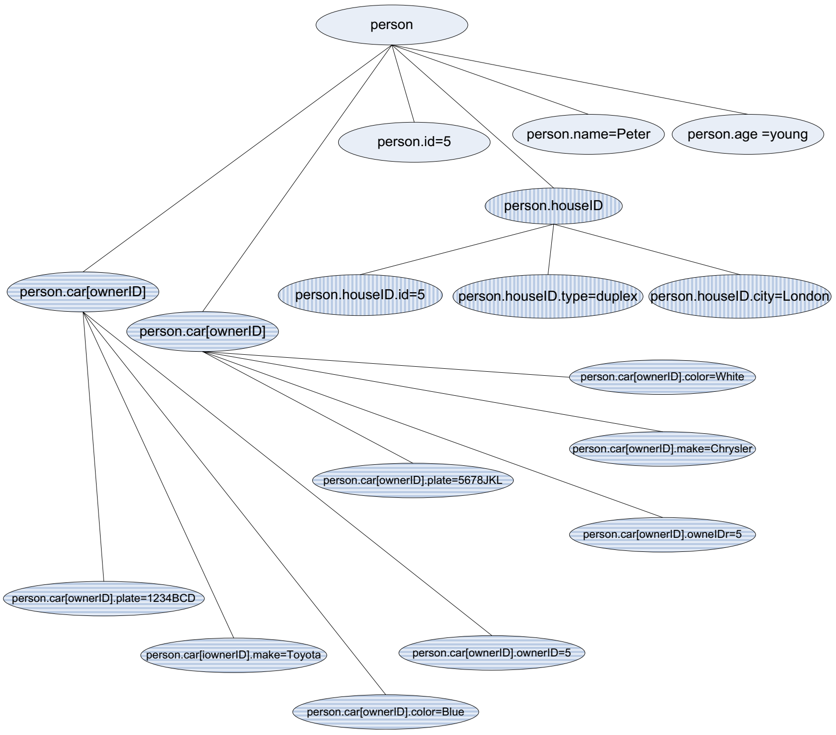
The tree is then built by exploring the relationships between the target relation and the others relations in the database. Let x be the target table, *person* in the example shown in Figure 2, and $x.K_x = k$ the node that represent the primary key, i.e. *person.id*. We can find two possible scenarios:

– When we have a one-to-one or many-to-one relationship between two relations, x and y, an attribute A of table x will be a foreign key pointing to the primary key of the table y. The attributes B of table y will be the children of $x.A = a$ in the tree and they will be represented as $x.A.B = b$. In our example, the relationship between Person and House is one-to-one and its representation is depicted in the Figure 3 *a)* using nodes with vertical lines in their background.

– When we have a one-to-many relationship, an attribute B of table y is a foreign key that refers to the primary key of table x. Many tuples in y may point to the same tuple in x. In this case, for each tuple in y that points to the same tuple in x, we create a new child of the x primary key node with the name of both tables, x and y, the attribute B that points to our target table, and the primary key of y with its value k_y using the notation $x.y[B].K_y = k_y$. This node will have, as children nodes, the resmaining attributes in y using the notation $x.y[B].C = c$. In our example, the one-to-many relationship between Person and Car is shown in the nodes shaded with horizontal lines in Figure 3 *a)*.


person
person.id =1
person.houseID =5
person.name =Peter
person.houseID.city=London
person.houseID.type=duplex
person.age =young
person.car[ownerID].plate=1234BCD
person.car[ownerID].plate=5678JKL
person.car[ownerID].make=Toyota
person.car[ownerID].make=Chrysler
person.car[ownerID].ownerID=1
person.car[ownerID].color=Blue
person.car[ownerID].idOwner=1
person.car[ownerID].color=White

a) Key-based tree representation.


person
person.id=5
person.name=Peter
person.age =young
person.houseID
person.car[ownerID]
person.houseID.id=5
person.houseID.type=duplex
person.houseID.city=London
person.car[ownerID]
person.car[ownerID].color=White
person.car[ownerID].make=Chrysler
person.car[ownerID].plate=5678JKL
person.car[ownerID].owneIDr=5
person.car[ownerID].plate=1234BCD
person.car[iownerID].make=Toyota
person.car[ownerID].ownerID=5
person.car[ownerID].color=Blue

b) Object-based tree representation.

Fig. 3. Tree-based representation alternatives for the multirelational database shown in Figure 2

3.2 Object-Based Tree Representation

The object-based representation uses intermediate nodes as roots of the subtrees derived from the data in each table. In this representation, all the attribute values within a tuple will be in the same tree level.

If we have a single table, all the attribute values within the tuple (including the primary key) will be children of the root node in the tree representing the tuple.

– The case of one-to-one and many-to-one relationships is now addressed by adding the attributes of y as children of the node $x.A$ using the notation $x.A.B = b$. The nodes shaded with vertical lines illustrate this in Figure 3*b)*.
– When another table y has a foreign key B that refers to the target table x, a new node is built for each tuple in y that points to the same tuple in x. This node is labeled with the name of both tables and the attribute B involved in the relation, i.e. $x.y[B]$. Its children are all the attribute values of the tuple in y, i.e. $x.y[B].C = c$. The nodes with horizontal lines in Figure 3*b)* show an example of this kind of relationship.

It should be noted that the object-based tree representation generates trees with more nodes than the key-based one. However, the tree depth is typically lower in the object-based tree representation than in the key-based one.

3.3 Deriving Trees from a Multirelational Database

Once we have presented two alternative schemes for the tree-based representation of multirelational databases, we have to establish how we traverse the relationships between the relations in our database to build the tree. In particular, we have to consider if it is interesting to go back through using a relationship that we have already represented in the tree.

In Figure 3, when we have represented the information about Peter and his cars, it is not necessary to go back through the person-car relationship because we would again obtain the information we already have in the tree.

However, if the target table were car, we would first represent the information of the Toyota car. Next, we would traverse the car-person relationship to obtain the information about the owner of the car (Peter). Finally, we would go back through the person-car relationship to represent all the cars that Peter owns.

Therefore, if we go through a one-to-many relationship from the relation with single cardinality, it is not necessary to go back. However, if we start from the table with multiple cardinality, we can go back through the same relationship to obtain more information.

4 Identifying Frequent Patterns in Multirelational Databases

The use of a tree-based representation for multirelational databases lets us apply tree mining techniques to identify the frequent patterns that are present in the multirelational database. Several algorithms have been devised to identify tree patterns, including TreeMiner [7], SLEUTH [8], and POTMiner [9].

Different kinds of subtrees can be defined depending on the way we define the matching function between the pattern and the tree it derives from [10] (see Figure 4):

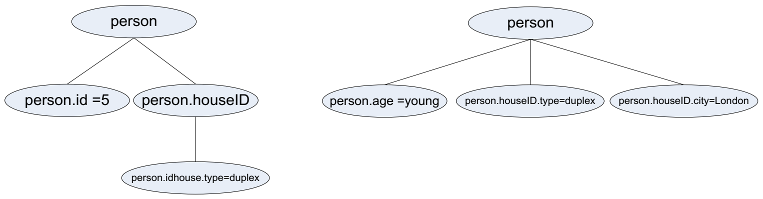

person
person.id =5
person.houseID
person.idhouse.type=duplex
person
person.age =young
person.houseID.type=duplex
person.houseID.city=London

Fig. 4. An induced subtree *(left)* and an embedded subtree *(right)* from the tree shown in Figure 3 *(b)*

- A **bottom-up subtree** T' of T (with root v) can be obtained by taking one vertex v from T with all its descendants and their corresponding edges.
- An **induced subtree** T' can be obtained from a tree T by repeatedly removing leaf nodes from a bottom-up subtree of T.
- An **embedded subtree** T' can be obtained from a tree T by repeatedly removing nodes, provided that ancestor relationships among the vertices of T are not broken.

4.1 Induced and Embedded Patterns

Induced and embedded patterns give us different information about the multirelational database.

Induced patterns preserve the relationships among the nodes in the tree-based representation of the multirelational database. Induced patterns describe the database in great detail, in the sense that the patterns preserve the structure of the original trees in the database. However, identifying large patterns is often necessary to obtain useful information from the multirelational database. Unfortunately, this might involve a heavy computational effort.

If we use embedded patterns, some of the relationships among the nodes in the original trees are not preserved. In other words, we could obtain the same pattern from different tree-structures in the database. On the other hand, embedded patterns are typically smaller than the induced patterns required to represent the same information.

For example, Figure 5 shows some patterns identified from the object-based tree representation of the multirelational database in Figure 2. The induced pattern shown on the left tells us that some people in our database have a Toyota *and* a white car, while the induced pattern on the right of the Figure tells us that people in our database have a Toyota *that* is also white. In the case of the embedded pattern shown in the same Figure illustrates that some people

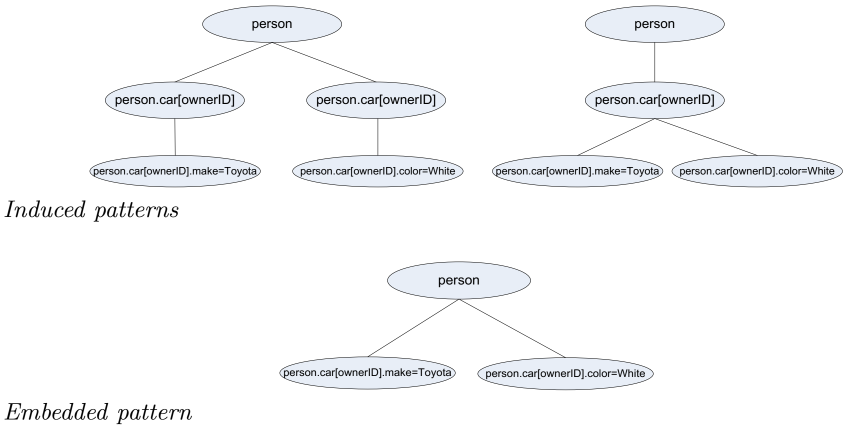

person
person.car[ownerID]
person.car[ownerID]
person.car[ownerID].make=Toyota
person.car[ownerID].color=White
person
person.car[ownerID]
person.car[ownerID].make=Toyota
person.car[ownerID].color=White
Induced patterns
person
person.car[ownerID].make=Toyota
person.car[ownerID].color=White
Embedded pattern

Fig. 5. Embedded and induced patterns identified in the object-based tree representation of the multirelational database in Figure 2

have a Toyota car and a white car, but we do not know if it is the same car (a white Toyota) of they own two cars (a Toyota and a white car).

4.2 Key-Based and Object-Based Patterns

The key-based and the object-based tree representation schemes for multirelational databases also provide us different kinds of patterns.

Since intermediate primary key nodes from the target relation are not frequent in the tree database and induced patterns preserve all the nodes as in the original tree, no induced patterns starting at, or including, a primary key node from the target relation will be identified using the key-based representation scheme. However, it is possible to identify induced patterns starting at other key nodes because they may be frequent in the database.

When we use the object-based representation, induced patterns can be obtained with information about the target table. Therefore, the object-based representation is our only choice if we are interested in patterns that preserve the original structure of the trees in the database, i.e. induced patterns.

On the contrary, using the object-based representation to discover embedded patterns is useful only if we are interested in patterns like the one shown in Figure 6. That pattern indicates that Londoners with two cars are frequent, without any references to the car features. It should be noted that this kind of pattern cannot be identified using the key-based representation, since they always involve attribute values. In the object-based representation, however, the presence of intermediate nodes increases the number of identified patterns. Therefore, we should only resort to the object-based representation when we are interested in patterns like the one in Figure 6. Otherwise, the key-based representation provides faster results.

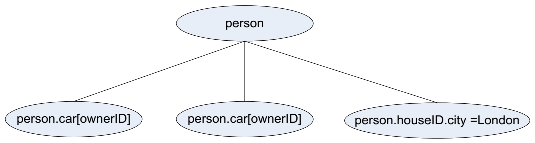


Fig. 6. An embedded subtree from the object-based tree in Figure 3(b)

5 Experimental Results

Once we have described our two tree representation schemes for multirelational databases, we discuss the experimental results we have obtained using both representation schemes. We have used the POTMiner algorithm [9] to identify induced and embedded subtrees from the trees representing the mutagenesis multirelational database, which can be downloaded from:
http://www.ews.uiuc.edu/∼xyin1/files/crossmine.html.

	Induced Subtrees					
	Support = 20%		Support = 10%		Support = 5%	
Representation	**Patterns**	**Time(ms)**	**Patterns**	**Time(ms)**	**Patterns**	**Time(ms)**
Key-based	29	3040	29	3096	63	4315
Object-Based	4308	314378	6891	531263	14363	797126

	Embedded Subtrees					
	Support = 20%		Support = 10%		Support = 5%	
Representation	**Patterns**	**Time(ms)**	**Patterns**	**Time(ms)**	**Patterns**	**Time(ms)**
Key-based	4886	131185	8159	190484	16950	280758
Object-Based	19862	1783031	31143	1749505	63915	3034299

Fig. 7. Number of patterns and POTMiner execution time corresponding to the identification of induced and embedded patterns in the mutagenesis multirelational database using different minimum support thresholds

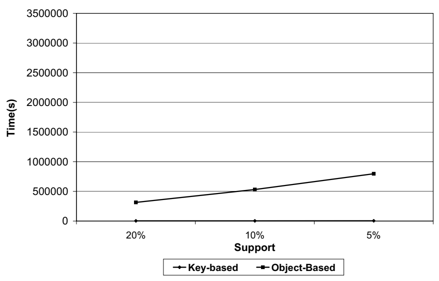
In our experiments, we have identified induced and embedded patterns including up to four nodes, i.e. MaxSize=4.

Figure 7 shows the number of discovered patterns and POTMiner execution time using different minimum support thresholds for the mutagenesis dataset, for both the key-based and the object-based tree representation schemes.

The number of discovered patterns using the object-based tree representation is larger than the number of identified patterns using the key-based representation. This is mainly due to the use of intermediate nodes, which are usually frequent, to represent objects.

It should also be noted that the number of induced patterns obtained from the key-based representation of the database is very small. This is due to the use of primary keys as internal nodes within the trees.

Figure 8 shows a comparison of the time required to identify induced and embedded patterns using both the key-based and the object-based representation schemes.



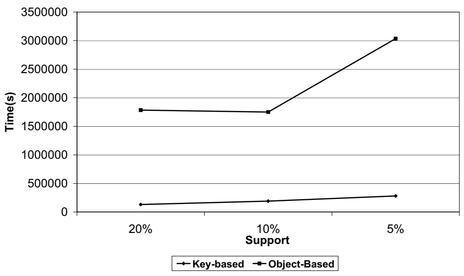


Fig. 8. POTMiner execution times when identifying induced patterns (left) and embedded patterns (right) in the mutagenesis database

Our algorithm execution time is proportional to the number of considered patterns. The execution time for discovering induced patterns is lower than the time needed to discover embedded patterns because a lower number of induced patterns are identified. Likewise, the object-based representation requires more execution time because a greater number of patterns are considered.

Albeit not shown in the Figure, it should also be noted that our algorithm is also linear with respect to the number of trees in the database.

6 Conclusions

This paper proposes a new approach to mine multirelational databases. Our approach is based on trees and we have designed two alternative tree representation schemes for multirelational databases. The main idea behind both of them is to build a tree representing each tuple in the target table by following the existing foreign keys that connect tables in the multirelational database.

The key-based representation scheme uses primary keys as intermediate nodes in the trees representing each tuple in the target relation. In contrast, the object-based representation scheme uses generic references as intermediate nodes to include new tuples in the trees.

We have identified frequent patterns in the trees representing the multirelational databased and we have studied the differences that result from identifying induced or embedded patterns in the key-based or in the object-based representation scheme. These frequent patterns can be used to extract association rules from the multirelational database, build multirelational classification models, or develop multirelational clustering techniques.

Our experiments with an actual database show that our approach is feasible in practise. The discovery of induced patterns combined with the object-based representation scheme is often enough to describe a multirelational database in great detail. Embedded patterns, when used with the key-based representation scheme, let us reach data that is farther from the target table, although they might not preserve the structure of the original database trees.

Acknowledgements

Work partially supported by research project TIN2006-07262.

References

1. Tung, A.K.H., Lu, H., Han, J., Feng, L.: Efficient mining of intertransaction association rules. IEEE Transaction on Knowlegde and Data Engeneering 15(1), 43–56 (2003)
2. Lee, A.J.T., Wang, C.S.: An efficient algorithm for mining frequent intertransaction patterns. Inf. Sci. 177(17), 3453–3476 (2007)
3. Džeroski, S.: Multi-relational data mining: An introduction. SIGKDD Explorations Newsletter 5(1), 1–16 (2003)
4. Yin, X., Han, J., Yang, J., Yu, P.S.: CrossMine: efficient classification across multiple database relations. In: International Conference on Data Engineering, pp. 399–410 (2004)
5. Yin, X., Han, J., Yu, P.S.: Cross-relational clustering with user's guidance. In: Knowledge Discovery and Data Mining, pp. 344–353 (2005)
6. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide, 2nd edn. The Addison-Wesley Object Technology Series. Addison-Wesley Professional, Reading (2005)
7. Zaki, M.J.: Efficiently mining frequent trees in a forest: Algorithms and applications. IEEE Transactions on Knowledge and Data Engineering 17(8), 1021–1035 (2005)
8. Zaki, M.J.: Efficiently mining frequent embedded unordered trees. Fundamenta Informaticae 66(1-2), 33–52 (2005)
9. Jimenez, A., Berzal, F., Cubero, J.C.: Mining induced and embedded subtrees in ordered, unordered, and partially-ordered trees. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) Foundations of Intelligent Systems. LNCS (LNAI), vol. 4994, pp. 111–120. Springer, Heidelberg (2008)
10. Chi, Y., Muntz, R.R., Nijssen, S., Kok, J.N.: Frequent subtree mining - an overview. Fundamenta Informaticae 66(1-2), 161–198 (2005)