STUDIES IN LOGIC, GRAMMAR AND RHETORIC 11 (24) 2007

Dariusz Surowik
University of Białystok

A FEW REMARKS ON QUERYING LISTS, TREES AND DAGS A TEMPORAL-LOGIC APPROACH[1]

Abstract. This paper discusses various conceptions of query languages for database systems (object oriented), in which objects are lists, trees and directed acyclic graphs. I use temporal logic as a modelling tool for the query languages under consideration. In the case of query language regarding lists, I will discuss temporal logic constituting an extension of linear time temporal logic, whereas in the case of query language regarding trees and directed acyclic graphs, I will discuss temporal logics constituting extensions of the branching–time temporal logic, the so called computation tree logic (CTL).

Key words: temporal logic, data models, query language

1. Introduction

Lists, trees and graphs are among the most important data structures in informatics. Computer linguistics or text databases are typical fields of database applications where trees, for example, are useful as a data type. In these applications, trees can serve as a tool for modelling a description of integrated concepts (for example, as a result of syntactic analysis) or for representing document structure. Although these structures are very important as tools for modelling, they have hardly been used so far as data types in existing databases. Standard database systems offer only sets of suitable relations as a way of representing unit sets and simple data types, such as whole numbers or strings, for representing unit attributes.

The present paper discusses lists, trees and directed acyclic graphs, as well as formal tools offered by temporal logic for searching the aforemen-

[1] The research reported in this paper is part of the project entitled *Temporal Representation of Knowledge and Its Implementation in the Computer Systems of Medical Conducts* supported by the Polish Ministry of Science and Higher Education, grant no. 3 T11F 011 30.

ISBN 978–83–7431–158–8 ISSN 0860-150X

tioned data structures. Applying temporal logic allows one to easily construct queries, which would be difficult to express without taking a temporal context into account. For example, a temporal modality such as "always" can be used to express the dynamic utterance "The wages of the workers on the list never decrease". One can, of course, express the same in a sense in a 'static' way as, for example, "the workers' wage list is ordered according to wages". It seems, however, that the use of temporal context makes the utterance more interesting.

2. Querying lists

In order to correctly define the syntax of temporal logic used for formulating queries in structures which have a form of lists, we must specify what we mean by the notion of an *atomic formula*. The correct definition of this concept turns out to be problematic, however. The problem lies in the fact that the elements of a list may have a very complex structure. What an atomic formula actually is depends therefore on the type of data to which the list has been applied. Thus, the definition of an atomic formula should be different when the elements of the list are some numerical values and different when the list has sets or families of sets as its elements. An *atomic formula* is a predicate which can be directly evaluated by referring to an individual element of a list. I am aware of the fact that this is not a precise definition of the concept in question. An atomic formula is a syntactic notion and should be described in syntactic terms. Unfortunately, the only thing we can say at this point about an atomic formula from a syntactic point of view is that it does not comprise logical operators[2]. An example of an atomic formula is the formula $(\circ < 2)$. This formula is true if the value of the current element of the list is smaller than 2.

In the language $\boldsymbol{L}$ of our logic we have the connectives $\neg, \vee$ temporal operators S (Since), U (Until), $\circ$ (Next) and $\bullet$ (Previous), and the binder operator $\uparrow$.

Well formed formulas we may define as follows:

- Atomic formula is well formed formula,
- If α, β are well formed formulas, then so are $\neg\alpha$, $\uparrow\alpha$, $\circ\alpha$, $\bullet\alpha$, $\alpha \vee \beta$, $\alpha U \beta$, $\alpha S \beta$.

[2] In some temporal logic systems an atomic formula is simply a formula which does not contain temporal operators.

SYNTAX

Definition 1

A list is a string of n objects $L[1], L[2], \ldots, L[n]$, where $n > 0$.

The value of the i-th element of the list will be marked as $L[i]$, whereas the i-th element of the list will have the symbol $\langle L, i\rangle$. In order to refer to the current position on the list, I will use the symbol $\downarrow$.

The truthfulness of the formula is defined as the truthfulness at the i-th point of the list L.

Definition 2

For any list L, for any $1 \leq i \leq n$ and for any formula α holds:

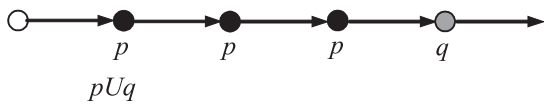
1) $\langle L, i\rangle \models \alpha$ $\equiv$ α is an atomic formula and evaluates to true for $L[i]$,
2) $\langle L, i\rangle \models \neg\alpha$ $\equiv \langle L, i\rangle \not\models \alpha$,
3) $\langle L, i\rangle \models (\alpha \vee \beta)$ $\equiv \langle L, i\rangle \models \alpha$ or $\langle L, i\rangle \models \beta$,
4) $\langle L, i\rangle \models \circ\alpha$ $\equiv$ if $1 \leq i < n$, then $\langle L, i+1\rangle \models \alpha$,
5) $\langle L, i\rangle \models (\alpha\, U \beta)$ $\equiv \exists_{j \geq i}$ such that $\langle L, j\rangle \models \beta$ and $\forall_k$ (if $i \leq k < j$, then $\langle L, k\rangle \models \alpha$),
6) $\langle L, i\rangle \models \bullet\alpha$ $\equiv$ if $1 < i \leq n$, then $\langle L, i-1\rangle \models \alpha$,
7) $\langle L, i\rangle \models (\alpha\, S \beta)$ $\equiv \exists_{j \leq i}$ such that $\langle L, j\rangle \models \beta$ and $\forall_k$ (if $j < k \leq i$, then $\langle L, k\rangle \models \alpha$),
8) $\langle L, i\rangle \models \uparrow x\alpha$ $\equiv \langle L, i\rangle \models \alpha^x_{L[i]}$.

Operators $\circ$, $\bullet$ are the operators of the next and the previous respectively. The notation $\langle L, i\rangle \models \circ\alpha$ is understood as follows: *a formula α is fulfilled for the $i+1$-st element of the list.* The graphic interpretation of the notation $\langle L, 1\rangle \models \circ p$ is shown below.

In the language of the logic under consideration, the sentence *the follower of the current element of the list is negative* can be written as $\circ(\downarrow < 0)$.

Operators U and S are known operators *until* and *since*. The formula $\alpha\, U \beta$ is understood as follows: *formula β is true at a certain moment in the future, let us call this moment j, whereas formula α is true from the*

current moment till the moment j. The graphic interpretation of the notation $\langle L, 2\rangle \models (p\,U q)$ is shown below.



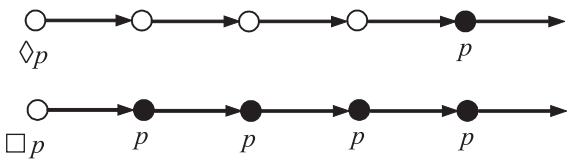
The notation $L \models \alpha$ will be used as an abbreviation for the notation $\langle L, 1\rangle \models \alpha$. If $L \models \alpha$ obtains, we will say that list L is the model for formula α. Let us note that operators $\circ, \bullet$ are weak operators, i.e., formula $\circ\alpha$, $(\bullet\alpha)$ is true in the last (first) element of the list, regardless of the form of formula α. This is the so called empty fulfilment, since, as follows on from the definition of fulfilment respectively for operators $\circ, \bullet$ in the first (last) element of the list the predecessor of the corresponding implication is false.

With the help of the previously defined operators, we can introduce additional specific temporal operators.

Definition 3

1) $\bar{\circ}\alpha \equiv \neg\circ\neg\alpha$ (strong next)
2) $\bar{\bullet}\alpha \equiv \neg\bullet\neg\alpha$ (strong previous)
3) $\Diamond\alpha \equiv true\ U\alpha$ (eventually in the future)
4) $\blacklozenge\alpha \equiv true\ S\alpha$ (eventually in the past)
5) $\Box\alpha \equiv \neg\Diamond\neg\alpha$ (always in the future)
6) $\blacksquare\alpha \equiv \neg\blacklozenge\neg\alpha$ (always in the past)

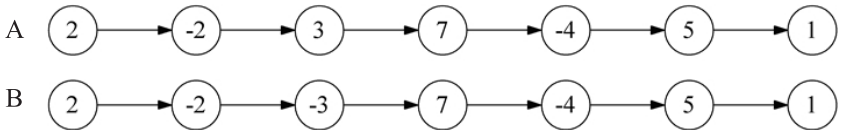
Examples of the graphic interpretations of the chosen operators are presented below.



Formula $\bar{\circ}\alpha$ is true at the i-th point of list L if the i-th point of the list is not the last point and α is fulfilled at point $i+1$.

Example 1

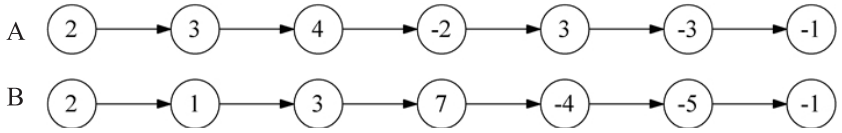
a) The sentence *The list does not contain two neighbouring negative elements* can be written symbolically as: $G((\downarrow < 0) \Rightarrow \circ(\downarrow \geq 0))$



The model for the formula under consideration is list A; while list B is not the model for this formula due to the value of the second and third element of the list. In the language of first order classical logic the above formula can be written as follows:

$$\neg\exists_{1\leq i<n}\,(L[i] < 0 \wedge L[i+1] < 0)$$

b) The expression $(\downarrow > 0) \wedge \bar{\circ}((\downarrow > 0)\, U \Box(\downarrow < 0))$ is a formal notation of the statement that
The first element of the list is positive. The following elements are also positive until a certain element is reached whose value is negative. All the elements following the element that has negative value, also have negative value. (We can thus state that the change in the value sign of the elements of the list has occurred only once).



List A is not a model for this formula, because the change of the sign of the elements of the list occurs twice. List B, on the other hand, constitutes the model for the formula under consideration since the first element of the list is positive and the change in the sign of the elements of the list occurs only once. In the language of first order classical logic the formula under consideration can be written as follows:

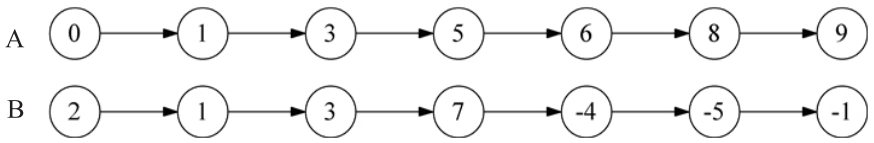
$$L[1] > 0 \wedge \exists_{j>1}\,(\forall_{1\leq i<j}\, L[i] > 0 \wedge \forall_{i>j}\, L[i] < 0)$$

So far we have considered expressions in which individual list values are compared to some constant arbitrarily chosen value. However, in the case of a query of the type *Is the list monotonously growing?*, we have to compare a number of values of the elements of the list to one another. To this end we shall use the operator $\uparrow$. By formula $\alpha^{x}_{L[i]}$ we understand a formula obtained from formula α by replacing all free occurrences of variable x in formula α with the value $L[i]$. According to definition 2, we should understand formula $\uparrow x\alpha$ in the same way.

Example 2

a) Formula $\Box(\uparrow x \circ(\downarrow > x))$ is a formal notation of the statement that *the list is monotonously growing*. (In the language of first order classical logic the above formula can be written as follows: $\forall_{1 \leq i < n}\ (L[i+1] > L[i])$).

b) Formula $\Box(\uparrow x \neg \circ \Diamond(\downarrow = x))$ is understood as: *the list does not contain two identical elements*. $(\neg(\exists_{1 \leq i \leq n}\ \exists_{1 \leq j \leq n}\ (i \neq j \wedge L[i] = L[j]))$ in the language of first order classical logic).



List A is a model for the formula $\Box(\uparrow x \circ(\downarrow > x))$, while both list A and list B are examples of models for the formula $(\Box(\uparrow x \neg \circ \Diamond(\downarrow = x))$.

3. Querying trees

Let us consider an unordered tree $T = (V, E, \lambda)$. V is a set of nodes of tree T, E is a set of edges of tree T, while λ is a function assigning to each node v of tree T a value from a certain domain D and thus $\lambda : V \rightarrow D$. Analogously to lists, $T[v]$ denotes a value in node v of tree T, while by $\langle T, v \rangle$ we understand node v of tree T.

In order to analyse lists with the help of temporal logic, it is sufficient to adopt notions of temporal logic regarding linear time (PLTL). In linear time there is only one future (just like in list type structures) and each element can have only one follower. It is different in the case of tree-like structures, whose elements can have several followers. The future in such structures can branch. Hence, for tree analysis one should use notions created for the purpose of temporal logic of branching time. A temporal logic which, after small modifications, can be used to construct the language of queries for tree-like structures is CTL* logic. The main idea of CTL* logic was introducing the so called path operators E, A. Formula $E\alpha$ should be understood as *There is a path starting at the current node such that formula α is fulfilled in this path*. Formula $A\alpha$, on the other hand, should be understood in the following way: *In all the paths starting at the current node formula α is fulfilled*.

A temporal logic which was to serve the purpose of tree analysis was proposed by Peter Becker [1]. Because it is a construct very similar to the known

CTL* logic, the author of the system in question uses the symbol CTL^*_{DB} for his logic.

Just as in the case of lists, this logic assumes that there is a set of atomic formulae. In CTL^*_{DB} logic, however, we deal with two types of formulae, i.e. *point formulae* and *path formulae*. Point formulae are connected to nodes and their truthfulness or falsehood is verified with reference to a given node. Path formulae, on the other hand, are connected to paths and their truthfulness or falsehood is verified in relation to a given path.

The syntax of CTL^*_{DB} is as follows:

Definition 4

a) State formulas
   1. Each atomic formula is a state formula,
   2. If α and β are state formulas, then $\alpha \vee \beta$, $\neg\alpha$ are state formulas,
   3. If α is a patch formula, then $E\alpha$, $A\alpha$ are state formulas.

b) Patch formulas
   1. Each state formula also is a patch formula,
   2. If α and β are patch formulas, then $\alpha \vee \beta$, $\neg\alpha$ are patch formulas,
   3. If α and β are patch formulas, then so are: $\alpha U \beta$, $\alpha S \beta$, $\circ\alpha$, $\bullet\alpha$.

c) The set of state formulas generated by the above rules forms the language of CTL^*_{DB} logic.

Before presenting the semantics of CTL^*_{DB} logic, let us define the notion of maximal path for a given node.

Definition 5

Let v be a node of tree T. Path $p = (v, v_1, v_2, \ldots, v_{r_p})$ of tree T is called the *maximal path for node* v if and only if v_{r_p} is a leaf.

The definition of truthfulness for CTL^*_{DB} logic is the following:

Definition 6

a) STATE FORMULAS
   For each tree T, for each node v and for each state formulas α, β

   1) $\langle T, v\rangle \models \alpha \quad \equiv$ α evaluates to *true* for $T[v]$,
   2) $\langle T, v\rangle \models \neg\alpha \quad \equiv \langle T, v\rangle \not\models \alpha$,
   3) $\langle T, v\rangle \models (\alpha \vee \beta) \quad \equiv \langle T, v\rangle \models \alpha$ or $\langle T, v\rangle \models \beta$,
   4) $\langle T, v\rangle \models E\alpha \quad \equiv \exists_{p=(v,v_1,v_2,\ldots,v_{r_p})}\ p$ is maximal path, such that $\langle T, p, 1\rangle \models \alpha$,
   5) $\langle T, v\rangle \models A\alpha \quad \equiv \forall_{p=(v,v_1,v_2,\ldots,v_{r_p})}\ p$ is maximal path, such that $\langle T, p, 1\rangle \models \alpha$.

b) PATH FORMULAS

For each tree T, for each maximal path $p = (v, v_1, v_2, \ldots, v_{r_p})$, for each $1 \leq i \leq r$ and for each path formulas α, β holds:

1) $\langle T, p, i\rangle \models \alpha \equiv \langle T, v_i\rangle \models \alpha$,
2) $\langle T, p, i\rangle \models \neg\alpha \equiv \langle T, p, i\rangle \not\models \alpha$,
3) $\langle T, p, i\rangle \models (\alpha \vee \beta) \equiv \langle T, p, i\rangle \models \alpha$ or $\langle T, p, i\rangle \models \beta$,
4) $\langle T, p, i\rangle \models \circ\alpha \equiv$ if $1 \leq i < r$, then $\langle T, p, i+1\rangle \models \alpha$,
5) $\langle T, p, i\rangle \models (\alpha\, U \beta) \equiv \exists_{j \geq i}$ such that $\langle T, p, j\rangle \models \beta$ and $\forall_k$ (if $i \leq k < j$, then $\langle T, p, k\rangle \models \alpha$),
6) $\langle T, p, i\rangle \models \bullet\alpha \equiv$ if $1 < i \leq r$, then $\langle T, p, i-1\rangle \models \alpha$,
7) $\langle T, p, i\rangle \models (\alpha\, S \beta) \equiv \exists_{j \leq i}$ such that $\langle T, p, j\rangle \models \beta$ and $\forall_k$ (if $j < k \leq i$, then $\langle T, p, k\rangle \models \alpha$),

Example 3

In the language of CTL^*_{DB} logic, the formulation:

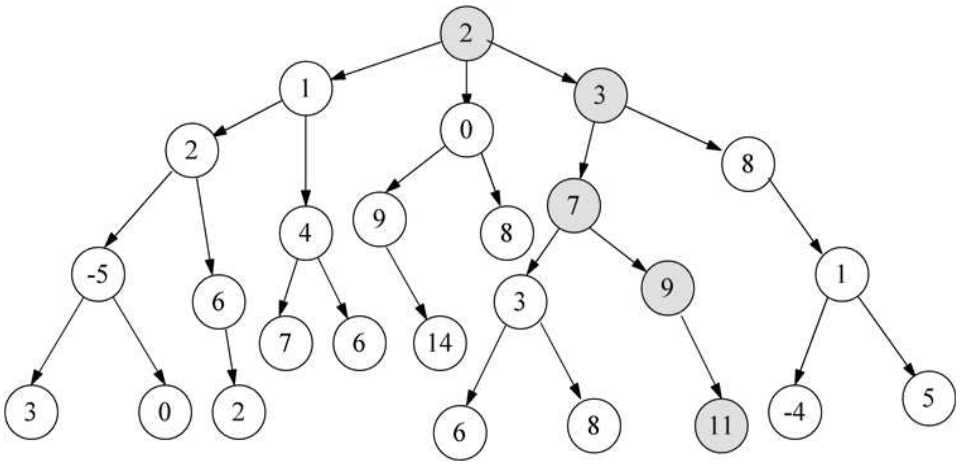
a) *There is a monotonously growing path* can be written symbolically as:

$$E\,(\Box\,(\uparrow x \circ (\downarrow > x))).$$

In the language of the first order logic the formula under consideration can be written as follows:

$$\exists_{p=(v_1, v_2, \ldots, v_{r_p}) \in T}\ \forall_{1 \leq i < r}\ (T[p, v_i] < T[p, v_{i+1}]).$$

An example of a tree-like structure, which is a model of the formula under consideration, is presented in the drawing below.



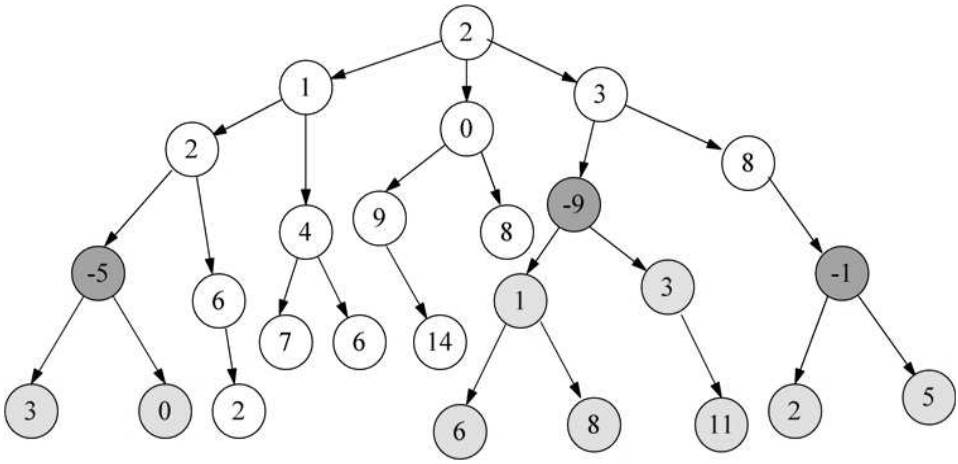
b) The statement that *Every node with a negative value has only non-negative children* can be written symbolically in the formula

$$A\,(\Box\,(\downarrow < 0) \rightarrow A(\circ(\downarrow \geq 0)))).$$

In the language of the first order logic the formula under consideration can be written as follows:

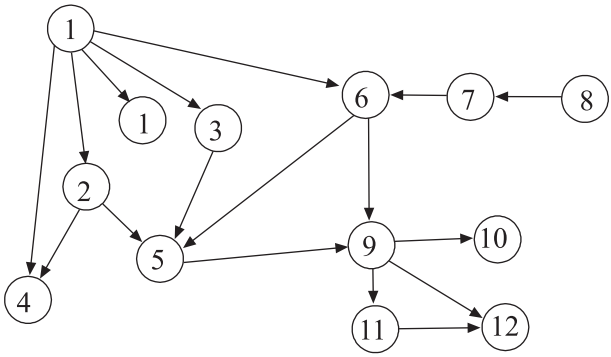
$$\forall_{p=(v_1,v_2,\ldots,v_r)\in T}\ ((\exists_{1\leq i<r}\ T[p,v_i] < 0) \Rightarrow$$
$$\Rightarrow \forall_{p'=(v_{1'},v_{2'},\ldots,v_{r'})\in T}\ ((\exists_{1\leq j<r'}\ v_i = v_j (\in p')) \Rightarrow (T[p', v_{j+1}] \geq 0))).$$

An example of a tree-like structure, which is a model of the formula under consideration, is presented in the drawing below:



4. Querying directed acyclic graphs

Analogical constructions can be created for the purpose of graph analysis. If a graph is coherent and undirected, it is equal to a tree, hence, in order to analyse such a graph we can use the tools discussed in the previous part of the paper. In the case of *directed graphs* (i.e. structures of the form $G = \langle V, A\rangle$ where: V is a set of vertices, A is a set of ordered pairs of various nodes from set V, called *directed edges*, or *arcs*: $A = \{(u, v), u, v \in V\}$), we will consider only *acyclic graphs*, i.e., directed graphs that do not have cycles. An example of such a graph is presented below.



Considering only this type of graph guarantees that there is a finite number of maximal paths. Unfortunately, contrary to trees, where the number of maximal paths from a given node to any leaf was limited by the number of nodes of a given tree, in graphs, the number of such paths grows exponentially. For this reason in the construction of the language of queries for directed acyclic graphs, one can use only the notions employed in CTL logic. It is only point formulae that are considered in this logic. A temporal logic which was to serve the purpose of directed acyclic graph analysis was proposed by Peter Becker [1], who marks the logic in question as CTL_{DAG}.

Unfortunately there are several problems connected with effective model verification for formulae of CTL_{DAG}. If, for example, G is an acyclic digraph with n nodes, then model verification for formula $\Phi\alpha$ of CTL_{DAG} logic (where Φ is any operator of this logic) may be performed by testing formula α for at least n nodes.

5. Conclusion

This paper discussed selected constructions of temporal logic systems, which allow forming queries for the purpose of analyzing lists, trees or acyclic digraphs. The constructions of these systems were based on CTL [3] logic or on CTL* logic. The use of temporal logic formalism for the analysis of the aforementioned data structures has its justification in well developed formal tools connected to temporal logic. The formalism of temporal logic expresses intuitive notions such as 'always', 'possibly', 'from now on...' well. This formalism, however, has certain limitations, which is especially visible when one considers issues relating to graph analysis, but not only. Even in the case of simpler data structures, such as lists, not everything can be expressed by means of temporal logic. For example, it is impossible to

formulate a query regarding only those elements of a list which occupy even positions [6].

Moreover, as we showed, some of queries are more complicated in the language of the first order classical logic in compare to the formulation in the language of temporal logic. From the other hand, there are some queries, which are simpler in the language of the first order classical logic.

Bibliography

[1] Becker P., *A temporal Logic based approach fro querying lists, trees and dags in databases*, vol. 978 of Lecture Notes in Computer Science, pp. 293–302, Springer, 1995.

[2] Cattel R. G. G., editor, *The object database standard*, ODMG-93. Morgan Kaufmann, 1994.

[3] Emerson E. A., *Temporal and modal logic*, Handbook of theoretical computer science, Elsevier, 1990.

[4] Richardson J., *Supporting lists in a data model (a timely approach),* Proceedings of the 18$^{\text{th}}$ VLDB Conference, 1992.

[5] Subramanian B., Zdonik B., Leung T. Vandenbergy S., *Ordered types in the AQUA Data Model*, Proc. 4$^{\text{th}}$ Intl. Workshop on Database Programming Languages, 1993.

[6] Wolper P., *Temporal logic can be more expressive*, Information and control, vol. 56, 1983.

[7] Yamamoto M, Tanabe Y, Takahashi K., Hagiya M., *Abstraction of Graph Transformation Systems by Temporal Logic and Its Verification.*