A Temporal-Abstraction Rule Language for Medical Databases

David Boaz[1], Mira Balaban[2], and Yuval Shahar[1]

Department of Information Systems Engineering,

Ben Gurion University, Beer Sheva 84105, Israel

[1]{dboaz, yshahar}@bgumail.bgu.ac.il, [2]mira@cs.bgu.ac.il

Abstract. Physicians and medical decision-support applications, such as for diagnosis, therapy, monitoring, quality assessment, and clinical research, reason about patients in terms of *abstract,* clinically meaningful concepts, typically over significant *time periods*. Clinical databases, however, store only *raw, time-stamped* data. Thus, there is a need to bridge this gap. We introduce the Temporal Abstraction Language (TAR) which enables specification of abstract relations involving raw data and abstract concepts, and supports query answering. We characterize TAR knowledge bases that guarantee finite answer sets and shortly explain why a complete bottom-up inference mechanism terminates. The TAR language was implemented as the inference component termed ALMA in the distributed mediation system IDAN, which integrates a set of clinical databases and medical knowledge bases. Initial experiments with ALMA and IDAN on a large oncology-patients dataset are highly encouraging.

1. Introduction: Temporal Abstraction and Deductive Databases

Many clinical domains require measurement and capture of numerous data of multiple types, often on electronic media. Making decisions in those domains requires reasoning about these data. Most stored data include a time stamp in which the particular datum was valid. Thus, it is desirable to automatically create *abstractions* of time-oriented data, and to be able to answer *queries* about such abstractions. These needs can be referred to as *temporal-abstraction* services. Providing these services would benefit both humans (e.g. physicians) and automated decision-support tools (e.g., clinical-guideline application, quality assessment of medical care, eligibility determination, exploration and visualization of time-oriented clinical data for patient-management and medical-research purposes, etc).

The main contribution of this paper is the proposal of a general language for temporal abstraction, investigating the properties of knowledge bases specified in that language, and presenting an implementation of a problem-solving module that answers queries about a set of time-oriented patient data. We define restrictions on such knowledge bases, in order to guaranty finiteness of answer sets. Under these restrictions, conventional bottom-up complete inference mechanisms terminate. The language is implemented as the inference component of the IDAN mediation architecture [1], and is used for visual exploration of a large set of medical records of patients monitored for several years after a bone marrow transplantation procedure. Initial results are highly encouraging.

1.1. Background

Many approaches had been proposed previously for providing temporal-abstraction services [2, 3, 4, 5, 6, 7].

One of the first in-depth ontologies for handling many aspects involved in the temporal-abstraction task is the ***knowledge-based temporal-abstraction*** (**KBTA**) ontology [8]. A problem solving method based on the KBTA *ontology*, the KBTA *method*, was implemented within the **RÉSUMÉ** system. The input of the KBTA method includes a set of time-stamped facts: primitive (raw-data) *parameters* (e.g., blood-glucose values), external *events* (e.g., insulin injections, a chemotherapy protocol), and, optionally, the user's *abstraction goals* (e.g., abstract the data in the context of "therapy of patients who have insulin-dependent diabetes"). The output includes a set of interval-based, context-specific parameters at the same or at a higher level of abstraction and their respective values (e.g., "a period of 5 weeks of grade III bone-marrow toxicity in the context of therapy with AZT"). (*Contexts* are *induced* by the existence of parameters, events, or abstraction goals).

The ***constraint-based pattern-specification language*** (**CAPSUL**) [9] is an extension of the KBTA ontology that describes its pattern-matching language. CAPSUL enables specification of *linear patterns* (a single occurrence of a set of phenomena, including other patterns, from which the pattern is composed, and constraints on that set) and p*eriodic patterns* (two or more repetitions of a phenomenon, and the constraints on these repetitions).

A useful framework for discussion and analysis of temporal-abstraction inference rules is a deductive database.

A *deductive database* is a general approach for answering queries that are formulated as *rules* [10, 11]. *Extensional relations* in deductive database are equivalent to regular relations. In addition, a deductive database extends regular databases with rules that specify *intensional* relations (intensional relations are close to database views with recursion mechanism). Rules in a deductive database are more expressive than relational algebra, since they may be defined *recursively*. Relations in a database driven knowledge base must be finite. The finiteness property is called *safety*. Significant amount of researches [10, 12, 13] was devoted to the syntactic characterization of safety in deductive database, and to the study of complete computation mechanisms. Determining safety is undecidable for general deductive databases with function symbols. The study of safety concentrates on the characterization of classes of problems for which safety can be checked.

Methods for processing queries in deductive databases are partitioned into two classes: *top-down* and *bottom-up*. Bottom-up strategies start from the base relations and keep assembling them to produce derived relations, until they generate the query answer set. Top-down strategies start from a query, and keep reducing it by applying the rules to the derived predicates.

A **Datalog** database is a deductive database where functions are not permitted [11]. The domain of a Datalog database is finite, since the extensional predicates are finite (the number of tuples in the database relation is finite), and new terms can not be created (there are no function symbols). Therefore, there are simple algorithms for checking safety of Datalog rules, although they are not precise.

1.2 Requirements from Temporal-Abstraction Query Services

A necessary feature for temporal abstraction, other than the existence of the time dimension, is the existence of multiple abstraction levels in the domain, with mappings among them. Abstractions typically are *vertical*, derived from the values of one or more facts occurring at the same time, or *horizontal*, derived from facts occurring at different times. For example, "150 kgs" might be mapped to *heavy*, while two distinct "heavy" facts that held on Monday and Friday might be mapped into one "heavy" fact that holds during the interval from Monday to Friday.

Our goal in this research is to formalize and generalize the semantics of temporal abstraction. We present a temporal abstraction mechanism that subsumes CAPSUL's linear pattern, as well as other mechanisms types in the KBTA ontology and the RÉSUMÉ system.

A temporal-abstraction service should supply the following requirements:

1. Finite answer sets to user queries.
2. Tractable and complete inference mechanism.
3. The temporal dimension, such as the time-point, time-interval, and time-measure data-types, should be part of the language.
4. The rules used to answer queries must enable evaluation of *value*-oriented and *time*-oriented *functions*. For example, mapping the hemoglobin value 10 gr/dl to *moderately low* requires a value-classification function; creating a pregnancy context during the nine months after conception, as done by the KBTA methods context-forming mechanism [14] requires, among others, a time-oriented function.
5. The language should enable specification of recursive rules. For example, the KBTA method's interpolation mechanism [15], which concatenates two time intervals by bridging the gap between them, is recursive.
6. The time is the unique concrete domain supported by the language. That is, the language should be independent of any particular application domain, e.g., financial, meteorological or medical domains.

Note that relational algebra can not account for recursive reasoning. While Datalog does not allow functions and neither support the time dimension. General deductive databases allow function symbols but have to cope with termination problems. Thus, a specialized temporal-abstraction language and a corresponding knowledge-base structure are needed.

2. The Temporal-Abstraction Rules (TAR) Language

A TAR knowledge-base consists of *Rules* and *Facts,* and can be viewed as a subset of deductive databases. The following examples show how medical patterns are mapped into TAR rules:

Example 2.1: In patient with acute myocardial infarction (m.i.) the serum level of cardiac specific enzyme troponin increase 3 -12 hours after the onset of m.i., and return to base line over 5-14 days. Values (ng/ml) bellow 0.6 are *normal*, between 0.7 and 1.4 are *indeterminate* and above 1.5 are *abnormal*. The presence of this specific enzyme in serum permits accurate diagnosis of m.i. This pattern can be written in the TAR language as the following rule:

myocardial_necrosis(D, I, V) ⟵ *troponin(D, I_1, V_1)* | V_1>0.6, *ifn, vfn*

The consequence of the rule is *myocardial_necrosis(D, I, V)* (to the left of the arrow), and the rule condition is *troponin(D, I_1, V_1)* (to the right of the arrow). In addition, the rule has the constraint: V_1>0.6, *ifn*, is a *time function* that returns an appropriate interval relative to the examination date. *vfn* is a *value function* that returns an appropriate value according to the test result.

Example 2.2: Hemoglobin state is derived from hemoglobin measures. The function *hgbClass* maps *hemoglobin* values less than 9 into *low*, values between 9 and 16 into *normal*, and above 16 into *high*. Two close enough hemoglobin state facts can be concatenated into a single long fact. The function *interpolate* computes the new fact interval. This pattern can be written in the TAR language as the following pair of rules:

hemoglobin_state(D, I, V) ⟵ *hemoglobin(D, I_1, V_1) ifn, hgbClass*

hemoglobin_state(D, I, V) ⟵ *hemoglobin_state(D, I_1, V_1), hemoglobin_state(D, I_2, V2)* |
close_enogh(I1,I2), interpolate, vfn

Note, that the second rule is recursive.

2.1. TAR Language Syntax

The TAR language contains symbols of three types (for each there are constant and variable symbols; variables start with upper case letters): 1) *individual* symbols (e.g., *john*), 2) *time-interval* symbols (e.g., the day of 1/1/2000) and 3) *value* symbols (e.g., *2.3 cm, low, abnormal,* and *green*).

A fact is a TAR atomic formula specified using a predicate symbol with the signature *individual × time-interval × value*. For example, *height(john, 1/1/1990-1/1/1990, 152* cm*)*. That is, *atomic formulae* have the structure *p(d, i ,v)*, where *p* is a predicate symbol, *d* is a term of type individual, *i* is a term of type time-interval and *v* is a term of type value. A *fact* is a ground (without variables) atomic formula.

The language contains also a set of external evaluable *functions* and *constraints*: A *time-function* is a function symbol with the signature $(\textit{time-interval} \cup \textit{value})^n \rightarrow \textit{time-interval}$ (i.e. the function accepts time-intervals and values, and returns a time-interval). A *value-function* is a function symbol with the signature $(\textit{time-interval} \cup \textit{value})^n \rightarrow \textit{value}$. A *constraint* is an external n-ary predicate symbol that accepts time-intervals and values symbols as arguments, and evaluate to TRUE/FALSE by calling an external constraints package (e.g., *greater_than*, *during*, *monday* etc).

A *TAR rule* (to be defined from now on, simply as *rule*) is a statement of the form:

h(D, I, V) ⟵ *b_1(D, I_1, V_1),…,b_n(D, I_n, V_n)* | *{c_1,…,c_m}, ifn, vfn*

where *h(D, I, V), b_1(D, I_1, V_1),…,b_n(D, I_n, V_n)* are atomic formulae (note that the same individual appears in all atomic formulae, and $n \geq 1$), $c_1,\ldots,c_m$ are constraints, *ifn* is a time-function symbol, and *vfn* is a value-function symbol. *h(D, I, V)* is the *head* atom of the rule, *h* is the head predicate, each *b_i(D, I_i, V_i)* is a *base* atom of the rule, b_i is a base predicate in the rule, the conjunction of the bases is the *body* of the rule. Variables that appear in the head are *head variables*, and variables that appear in the body are *body variables,* denoted $\vec{B}$. The Interval and Value head variables are distinct from all body variables. A *rule instance* is a pair of substitutions (σ_h, σ_b), where σ_h is a substitution for the head variables, and σ_b a substitution for the body variables. A *ground rule instance* is a rule instance in which all body variables are ground.

2.2. Semantics of TAR

The Temporal Model: We assume a time line that can be identified with the integers. The model distinguishes among three temporal data-types. A *time-point* is identified with the integers, a *time-measure* that denotes sizes on the time-line and a *time-interval* that denotes a segment on the time line. For example, 3 o'clock is time-point, t*wo hours* is a time measure, and it can be from *3* to *5 o'clock* or from *8* to *10 o'clock*. A measure can be a positive, negative or zero.

Language of Constraints: In the TAR language facts can be constrained by temporal or value constraints. Temporal constraints are built-in, and value constraints depend on the application domain. Constraints can be combined using logical connectives and possibly by cardinality based connectives like *at-least-N*.

The language temporal built-in constraints are: *Calendar constraints* over a single time-point. e.g., *wednesday(P1)* holds if P1 happened on Wednesday. *Point-* and *Measure-constraints* enables comparison of time-points and time-measures. Following Allen time-interval algebra [16], we define *interval-constraints* as predicates equivalent to the 13 basic binary relations between intervals.

TAR language is domain independent. It can be applied on various value types, e.g., numbers, strings, images, structures etc. For each value type, the user adds external value constraints that are invoked (evaluated) on query

processing. It is the user responsibility to apply the correct constraints on the appropriate value types.

A TAR semantic structure is a pair $J= (D, \cdot^J)$ of a domain D. The Domain is partitioned into three parts: Individual, Time-Interval and Value.

- *Individual* is a finite non-empty domain of individuals (in the medical domain patients are individuals), e.g., the person whose name is John.
- *Time-Interval* is an infinite domain of time intervals. An interval is a segment on a discrete time line, e.g., the segment of time from 1/1/2000 00:00 to 04:00.
- *Value* is a finite or infinite domain of values. Different value domains are possible in different application domains, e.g., *182 centimeters* and *high-fever*.

The time-interval and value symbols in the language are identified with the entities in the Time-Interval and the various Value domains. That is, the collection of symbols is the semantic domain for these types.

A time-function symbol denotes a function that accepts time-intervals and values, and returns an interval. Similarly, a value function symbol denotes a function with the same argument types that return a value. A relation symbol denotes I*ndividual × Time-Interval × Value.* For example: the meaning of the relation *height* is a finite set of triplets: {*(john, 1/1/1990-1/1/1990, 152* cm*), (marry, 1/1/1990-1/1/1990, 160 cm), (john, 1/1/2000-1/1/2000, 185 cm)…}*. Constraint symbols denote external constraint predicates. The constraints are partitioned into *temporal constraints* which are built-in, and *value constraints* which are evaluated using external, packages. A set of constraints must be evaluable for every ground instance of a rule.

Temporal-Abstraction Property: Given the rule $r=h(D, I, V) \leftarrow b_1(D, I_1, V_1),\ldots,b_n(D, I_n, V_n) \mid \{c_1,\ldots,c_m\}$, *ifn, vfn*, and an interpretation J, we say that the rule is true in J, written $J \models r$, if for every ground instance (σ_h, σ_b) of r, where σ_h is empty, the following holds:

If:

1. *All body atoms are satisfied in J, i.e.*, $J \models (b_1(D, I_1, V_1),\ldots,b_n(D, I_n, V_n))\sigma b$
2. *All constraints evaluate to TRUE, i.e.,* $(c_i)\sigma b$*=TRUE for every* $1 \le i \le m$

Then:

1. *The time variable, I, and the value variable, V, in the head are substituted to the values obtained by the time function, ifn, and value function, vfn, i.e., define* $\sigma h'$ *to be* $\{I=i, V=v\}$ *such that* $iJ=ifn((\vec{B}\sigma b)J)$ *and* $vJ=vfn((\vec{B}\sigma b)J)$
2. *The rule head holds in J, i.e.,* $J \models (h(D, I, V)\sigma_h'$

This condition is called the *Temporal-Abstraction Property* of *r* with respect to interpretation *J*.

An interpretation *J* is a *model* of a knowledge-base if every rule in it fulfills the temporal-abstraction property with respect to *J*. An inference mechanism for TAR can be either query driven i.e., find whether the query holds, in every model of the knowledge base, or can compute the intensional relations. We say that an inference mechanism is *complete* if it can compute the intensional relations.

2.3. Well Defined Knowledge-Bases

In a TAR knowledge base we are interested in having safe (finite) intensional relations that can be effectively computed by a complete inference mechanism. Such a TAR knowledge base is termed *well defined*. Clearly, the existence of a complete terminating inference mechanism implies safety. The common approach for computing intensional relations is bottom-up evaluation of the rules [10, 11]. In this approach one computes the knowledge-base least fixpoint (which is known to exists and be computable). Bottom-up evaluation repeatedly applies the rules to the facts, in order to create new facts; each application is called a *round*. If a round provides no new facts, the least fixpoint is reached (finite in this case), and the algorithm stops. Termination is guarantied [12] if: each round is finite, and the number of rounds is finite.

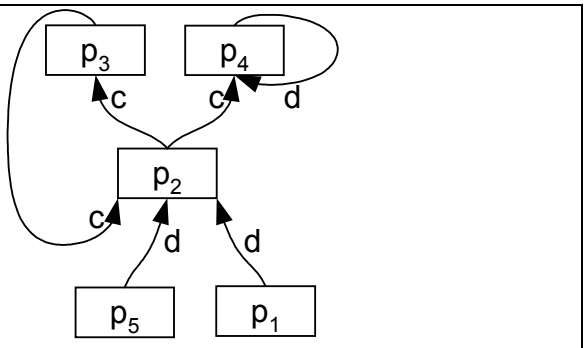
Claim 2.1: TAR rules guaranty finite rounds, i.e., in each round, a finite number of facts are added (note that in the general case, if a rule has an infinite number of instances, this is not necessarily true).

Proof: (shortened) The claim results form the fact that all base relations are finite, and the head variables are limited by the functions in the rule body.□

In order to guaranty termination we still have to make sure that the number of rounds in bottom-up evaluation is finite. For that purpose, we impose restrictions on the structure of knowledge bases. These restrictions consist of characterizations of two function types: *converging* and *diverging* and the definition of *diverging dependency graph* that is derived from the rules. A similar restriction was used in [13] for guarantying finiteness of answers to queries over sequence databases.

A *Converging function* is a function that does not produce new terms. e.g.: *maximum, member* and *substring* are converging functions. For instance, *maximum (2,4,1)* is *4*, a term that already exist in the knowledge base. On the other hand, a *Diverging function* might extend the domain.

For instance, the + function with the parameters (1, 2) creates a new term *3,* sequence concatenation is also a diverging function. The d*iverging dependency graph* of a TAR knowledge-base is a directed-graph, where *nodes* are labeled by the predicates. There is a *converging arc* from *b* to *h*, labeled "*c*", if there is a rule, *r*, with *h* as the head predicate, *b* as a base predicate and the time and value functions of *r* are converging; and there is a *diverging arc* from *b* to *h*, labeled "d", if at least one of the rule functions is diverging. A knowledge-base has a *stratified-diversion* structure if it does not contain cycles with a diverging arc. Fig. 1 shows an example for a knowledge base that is not stratified diversion.



r1: p2(V) ⟵p1(V1)| V= V1*2

r2: p2(V) ⟵p3(V3), p5(V5)| max(2*V5, V3)

r3: p3(V) ⟵p2(V2)| V= V2

r4: p4(V) ⟵p2(V2)| V= V2

r5: p4(V) ⟵p4(V4)| V= s(V4)

Fig. 1. An example of a knowledge base and its diverging dependency-graph (for the sake of simplicity, we omit the time variables, the constraints and the time function of the rules). *p1* and *p5* are finite, since they are extensional predicates. The value functions used in r1, r2, and r5 are diverging. *p2* and *p3* are finite because they are not participating in a cycle with a diverging arc. *p4* is not finite because it participates in a cycle with a diverging arc (p4→p4). The knowledge base does not have a stratified-diversion structure, and hence it is not well defined.

Claim 2.2: A TAR knowledge base with a stratified-diversion structure is well defined.

Proof: (shortened) Each round that applies diverging functions produces new terms. Therefore recursive application includes only converging functions that do not add new terms, while non recursive applications might be applied only finite number of times diverging functions that might add new terms. All in all, the amount of new terms in finite.□

2.6. The ALMA System and its Query Evaluation Strategy

We have implemented a system that applies the semantics of a TAR knowledge base to any particular time-oriented database, called *ALMA*[1]. ALMA processes TAR queries using a focused bottom-up strategy [10].

ALMA uses a set of optimizations, such as caching intermediate computation results, reordering of the rule body base predicates according to their number of tuples, evaluation of constraints as early as possible, in order to cut irrelevant branches, a special time-sorted data-structure, and other task-specific means.

ALMA is a component of IDAN distributed temporal-abstraction mediation architecture. IDAN includes a set of distributed data-sources, distributed knowledge-sources, as well as ALMA and a controller that serves as the interface to client applications (see [17] for details). When IDAN Controller receives a query, it prepares a set of potentially relevant facts, and a set of relevant rules (from the data and knowledge sources), then it passes the query with the facts and rules to ALMA. ALMA processes the query, and the appropriate result is returned to IDAN controller.

ALMA was implemented in Sicstus Prolog, benefiting of Prolog built-in unification, backtracking, and "objects" (a library for logic object oriented programming) features. Rules and facts are stored only in the main memory. For each distinct user (client) a new ALMA object is created, hence the facts different ALMA objects are not mixed. Each ALMA object remains in memory for a predefined amount of time (few minutes), and then it is collected. Caching the ALMA objects might consume time in the client next queries.

ALMA was successfully applied to a database of more than a thousand patients followed for up to four years after a bone marrow transplantation procedure, to support applications such as visual interactive exploration of time-oriented clinical data [17].

3. Discussion

TAR mechanism tries to formalize previous knowledge embedded in the KBTA. TAR is a more general language for reasoning over time. The KBTA Ontology is a more specific language that fulfills common abstraction needs. The KBTA method's mechanisms (without periodic patterns) can be mapped into the TAR language without losing their intuitive underlying original semantics. Hence,

[1] Alma is an Aramaic word meaning "hence". It is typically used as part of a logical argument.

we can say that the TAR language subsumes the KBTA ontology. Moreover, we have gained new insights by applying the TAR safety analysis on the KBTA mechanisms that were mapped into TAR rules.

As the KBTA ontology is more task-specific, a system based on it might perform better than the single, generic algorithm that interprets the TAR language. Knowledge acquisition in the KBTA framework is easier, since we know which templates the user expects, and they have clear, task-specific semantics and corresponding structure, while in the TAR language the rules are generic. A good symbiosis between the two languages is to interact with the user using the KBTA ontology (except for complex patterns, where we might use the more expressive TAR language), to translate the knowledge into rules, and to answer queries using ALMA. In the IDAN project, we have in fact adopted this solution with initial encouraging results.

The RÉSUMÉ implementation was responsible only for the generation of abstractions (i.e., temporal reasoning). The generated facts (output) were transformed into an intermediate storage format (e.g., a file, or a database). In order to answer queries there was a need to analyze this output. The benefit of this approach is that the reasoning can be done in a batch process, and queries are quickly answered. The main drawback of this approach that the query is not a part of the reasoning input, hence, the reasoning process can not be focused, as is the case in ALMA.

As ALMA does not store its facts in a persistent fashion (only for a short time, but under the assumption that during that time no changes were made in the extensional facts), Alma does not need a Truth Maintenance System (TMS), such as exists in RÉSUMÉ, which caters for modification of facts and propagation of effects. In the case of updates, ALMA will reprocess the data from scratch. Thus, ALMA does not operate in incremental fashion, as might be more appropriate in a context such as intensive care (where data arrive continuously and abstractions need to be updated rather then recomputed).

In order to be domain independent, ALMA intentionally puts value constraints in external packages. The rather general structure of such constraints enables ALMA to process not only simple value types, as number and symbols, but also complex data types, as lists or other structures. For example: medication administrations are often represented as complex frames such as <*dose*, *preparation*, *rout*…>. The ability to use additional domain-specific packages greatly extends the potential benefits of using ALMA.

The usage of deductive database approach contributes to the understanding of termination mechanism. In the future, we intend to relax the current characterizations of well defined knowledge bases. We intend also to optimize the query processing, mainly in order to be able to process in the same query several sets of patient records. For example, analyzing the set of all the patients who have had a bone-marrow transplantation in a particular hospital. In addition, we intend to define additional types of rules for *repeating* patterns.

The initial experience of using ALMA within the IDAN architecture leads us to believe that a well defined inference mechanism, such as represented by the TAR language, has the benefit of maintaining the safety, effective computability, of query processing. A domain independent expressive language for querying time-oriented clinical data, whose evaluation is efficient, has multiple implications for automated support to clinical care and medical research, both of which currently focus on chronic patients.

References

1. Boaz, D. and Y. Shahar: IDAN: A Distributed Temporal-Abstraction Mediator for Medical Databases. Proceedings of the 9th Conference on Artificial Intelligence in Medicine—Europe (2003), Protaras, Cyprus.
2. Kohane, I. S.: Medical Reasoning in Medical Expert Systems. In Salamon, R., et al., (eds.), Proceeding of Fifth Conference on Medical Informatics (MEDINFO-86), (1986) 170-174.
3. Russ, T.A.: Using hindsight in medical decision making. Proc. Symposium on Computer Applications in Medical Care, New York NY: IEEE Computer Society Press (1989) 38–44.
4. Kahn, M. G.: Combining physiologic models and symbolic methods to interpret time-varying patient data Methods of Information in Medicine 30 (1991) 167–178.
5. Larizza, C., A. Moglia, and M. Stefanelli: M-HTP: A system for monitoring heart-transplant patients, Artificial Intelligence in Medicine **4**(2) (1992) 111–126.
6. Haimowitz, I. J. and Kohane, I.S.: Managing temporal worlds for medical trend diagnosis. Artificial Intelligence in Medicine, Vol.8, No.3 (1996) 299–321.
7. Keravnou, E. T.: Temporal diagnostic reasoning based on time-objects. Artificial Intelligence in Medicine, Vol.8, (1996) 235–265.
8. Shahar, Y.: A Framework for knowledge-based temporal abstraction in clinical domains. Artificial intelligence 90(1-2) (1997) 79-133.
9. Chakravarty, S. and Y. Shahar: CAPSUL: A Constraint-Based Specification of Repeating Patterns in

Time-Oriented Data. Annals of mathematics and artificial intelligence (AMAI); Vol. 30 (2000) 3-22.

10. Bancilhon, F. and R. Ramakrishnan: An amateur's introduction to recursive query-processing strategies. ACM SIGMOD Intl. Conf. on Management of data, (1986) 16-52.
11. Ullman, J. D.: Principles of database and knowledge-base systems. (1988) volume1 chapter 3.
12. Krishnamurthy, R., R. Ramakrishnan, and O. Shmueli: A Framework for Testing Safety and Effective Computability of Extended Datalog. Proc. Sixth ACM Symposium on Principles of Database systems, (1988) 154-163.
13. Mecca, G. and A. J. Bonner: Sequences, Datalog and Transducers. In Fourteenth ACM SIGMOD Intern. Symposium on Principles of Database Systems (PODS'95), San Jose, California, (1995) 23-35.
14. Shahar, Y.: Dynamic Temporal Interpretation Contexts for Temporal Abstraction. Annals of Mathematics and Artificial Intelligence. 22(1-2) (1998) 159-92.
15. Shahar, Y.: Knowledge-based Temporal Interpolation. Journal of Experimental and Theoretical Artificial Intelligence. 11 (1999) 123-144
16. Allen, J. F.: Maintaining knowledge about temporal intervals. CACM 26 (11) (1983) 832-843.
17. Shahar Y., D. Goren-Bar, M. Galperin-Aizenberg, D. Boaz, and G. Tahan: KNAVE-II: A Distributed Architecture for Interactive Visualization and Intelligent Exploration of Time-Oriented Clinical Data. Intelligent Data Analysis in Medicine and Pharmacology—Europe (2003), Protaras, Cyprus.