Intelligent Information Processing using TRLi

Themis Panayiotopoulos

University of Piraeus
Dept. of Informatics
80 Karaoli and Dimitriou Str.
18534 Piraeus, Greece
e-mail : themisp@iit.nrcps.ariadne-t.gr

Manolis Gergatsoulis

N.C.S.R "Demokritos"
Inst.of Informatics and Telecomm.
15310 Aghia Paraskevi
Athens, Greece
e-mail : manolis@iit.nrcps.ariadne-t.gr

Abstract

An expressively rich temporal reasoning system, called TRLi is presented in this paper. TRLi introduces and relates the concepts of temporal points, temporal instances and temporal intervals, and has deeply embedded in its semantics the notion of temporal uncertainty. TRLi is a temporal logic system with a syntax and proof procedure very similar to that of Prolog, and it can therefore be easily used as the basic temporal deductive component of Intelligent Information Systems. To exploit its expressiveness, we have selected problems from various domains, such as intelligent problem solving, temporal planning, simulation and temporal databases and provide their solution in TRLi.

1. Introduction

The parameter of time is very important in all aspects of every day life, and thus a very important aspect of Information Systems. Information Systems which must take seriously into account a changing world, must also incorporate the means to express the temporal aspects of data.

Artificial Intelligence (AI) community has investigated many temporal logics which can deal successfully with various types of temporal information [1],[2],[3],[4]. Intense research has resulted to the development of Temporal Logics [1],[5],[6],[7],[8] which have found application in many domains, such as, planning, temporal databases, verification of concurrent systems, VLSI design, etc. Many practical systems which implement Temporal Logics have also been reported. However, most of them are implementations of Modal temporal logics [3].

The syntax and semantics of the TRL language have been proposed elsewhere [9]. The notion of temporal references which is used in TRL has also been discussed

[10]. Recently, the Horn TRL system (HTRL) was also developed [11] which is a subset of TRL and contains a pre-processor which transforms a TRL temporal program to a Prolog program with additional calls to a Symbolic Constraint Solver. An application of the TRLi system on temporal planning has been presented in [12].

In this paper we first briefly present the architecture of the *TRLi* (<u>T</u>emporal <u>R</u>eference <u>L</u>anguage <u>i</u>nterpreter) system, and informally discuss its syntax, semantics and inference rules. We then present some issues concerning different domains such as intelligent problem solving, simulation and temporal databases, along with their solution with the *TRLi* system. Then we discuss some implementation issues and finally we present conclusions and future research.

2. The TRLi Temporal Reasoning System

The *TRLi* temporal reasoning system follows the syntax and semantics of the TRL language, and the inference rules of HTRL in order to provide a system with automated temporal deduction capabilities. In contradiction to HTRL, which contains a preprocessor to transform a TRL temporal program to a Prolog program, the *TRLi* system has been developed as a meta-level interpreter, handling the temporal world and the queries as input data, and providing the answers as output data. The meta-interpreter implements the temporal rules of inference in collaboration with a Symbolic Constraint Solver.

2.1. Syntax and Semantics: Informal presentation

TRLi assumes that the world is changing: some assertions hold at specific temporal points, or through temporal intervals, or they may hold at some temporal instance. *TRLi* uses a syntax very similar to that of Prolog. Temporal information is expressed through

temporal references which are labels of atoms (predicates). Predicates referenced in such a way are called *extended predicates* or *extended atoms*. There are three categories of predicates: classical predicates, extended predicates and temporal predicates. *Classical predicates* bear no temporal reference on them, i.e. these are predicates normally found in Prolog. Extended predicates are further subcategorised into *properties* and *events*. *Temporal predicates* must be interpreted as relations among their temporal arguments.

Properties and events have been long ago introduced by Allen [6], but they are considered here as objects of a temporal logic. Their main difference concerns their behaviour over temporal intervals, as it will be explained later on.

Temporal references, [9],[10],[11], are constructed by using *temporal constants*, *temporal variables*, *temporal compound terms* and the *temporal constructors* '<',`>',`[',`]'. The most general temporal reference is the *uncertain temporal interval*, notated as <[T1,T2],[T3,T4]>, where each of Ti is a temporal constant, a temporal variable, or a temporal compound term. It represents a temporal interval with uncertain start and end points. If a relation is true during an uncertain temporal interval <[t1,t2],[t3,t4]> then there exists a time point r between t1 and t2 and a time point s between t3 and t4, such that the relation is true during the temporal interval <r,s>. For example, the statement "John stayed at the beach from some time between 9.00 am to 11.00 am till 5.00 pm" can be encoded in *TRLi* as:

<[9.00am, 11.00 am], 5.00 pm> : stay(john,beach)

The meaning of such a statement is that there is some time point, say 10:30 am, at which John arrived at the beach and he stayed there till late in the afternoon (5:00 pm).

There are many forms of temporal references, the most important of which are: the *temporal point* T, the *temporal interval* <T1,T2>, and the *temporal instance* [T1,T2].

Intuitively, when a predicate (event or property), or assertion is true at a temporal point t, then it is only known to be true at the moment t. Examples of pieces of information containing temporal points are the following : "The bus is leaving at 6:00 am", "Peter was promoted to a manager at January the 1st of 1993", etc., and their representation in *TRLi* is[1] :

6:00 am : leaving(bus).
1/1/1993 : promoted(peter,manager).

If a predicate (event or property) or an assertion is true at a temporal instance [t1,t2], then it is known to be true at some time between t1 and t2. Examples of pieces of information containing temporal instances are: "Maria will arrive at some time between 6 to 7 o'clock in the afternoon", "...the system crashed at some time in the morning", etc., and their representation in *TRLi* is :

[6:00 pm,7:00 pm] : arrives(maria).
[8:00 am,12:00 am] : system_crash.

Events and properties are types of extended predicates which behave in a different way over temporal intervals. Properties are relations which preserve their characteristics in subintervals, i.e. when a property is true during a temporal interval it is also true during all subintervals of this interval. Properties are states, attributes, properties of objects, e.g. the relation 'spaceship' in the statement <1980,2100>:spaceship(apollo) should be considered as a property because the object 'apollo' has the property of being a 'spaceship' from the moment it was built to the moment it was destroyed, and for all the intermediate time points.

When a property is true over an interval it is necessarily true over every subinterval of this interval: "Roberta din't stop dancing all evening", "John's bank account contained at least $5,000 during the last 5 years". In *TRLi* these sentences are encoded as:

<8:00 am,12 am>:dances(roberta).
<1/1/1990,1/1/1995>:bank_account (john,X) :-
X ≥ $5000.

From these examples we may easily infer that "Roberta was dancing from 9:00 am to 11:00 am", and "John's bank account contained at least $5000 during 1992".

On the other hand, events are relations which do not necessarily preserve their characteristics inside larger temporal intervals. For example, the relation 'published_papers' in the statement <1980,2000> : published_papers(george,50) does not necessarily preserve its truth value during the temporal interval <1980,1990>, i.e. George has 20 papers published from 1980 to 1990.

When an event is true over an interval, then it is not necessarily true over all subintervals of this interval: "Joice has accumulated $10000 during the last two years", "During the last hour he has run 10 times around the square" In *TRLi* these sentences are encoded as:

<1/1/1993,1/1/1995>:accumulates(joice,$10000).
<8:00,9:00>:times_around(square,10).

It is obvious that these predicates do not hold over temporal intervals which are contained in the given intervals, e.g. "Joice has not accumulated $10000 during the last five weeks", etc.

Notice that events and properties are syntactically equivalent, i.e. they are expressed by *TRLi* in exactly the same way. However, they are semantically different, that is they have different interpretations and are handled by the meta-interpreter in a different way.

2.2. Deduction and Inference rules

For classical atoms, we retain SLD-resolution [13], i.e. *TRLi* behaves like Prolog. For extended atoms, however, we have defined additional inference rules. These rules have been defined in [11] in a mathematical style. In the following lines we will briefly present the most important of these rules in an informal way which captures their intuitive meaning. Inference rules are given in the form "*From A infer B*", which is equivalent to "*to prove B, prove A*".

2.2.1. Inference rules for properties. In the cases of the inference rules for properties which follow we temporal intervals, temporal instances and temporal points. We should also keep in mind that both the temporal references <t,t> and [t,t] are temporal points.

[A1] From <u1,u2>:p infer <l1,l2>:p,
    where the temporal interval <u1,u2> contains the temporal interval<l1,l2>.
Example: From *<8:00 am,12 am> : dances(roberta)*
                infer *<9:00 am,11 am> : dances(roberta)*
Explanation: If Roberta was dancing from 8 to 12 in the evening, then she was also dancing from 9 to 11 in the evening.

[A2] From <u1,u2>:p infer [l1,l2]:p,
    where the temporal instance [l1,l2] contains at least on point of the temporal interval <u1,u2>.
Example: From *<8:00 am,12 am> : dances(roberta)*
                infer *[7:00 am,9 am] : dances(roberta)*
Explanation: If Roberta was dancing from 8 to 12 in the evening, then she was also dancing at some time between 7 and 9 in the evening.

[A3] From [u1,u2]:p infer [l1,l2]:p,
    where the temporal instance [l1,l2] contains the temporal instance [u1,u2].
Example: From *[8:00 pm,9 pm] : eats(paul)*
                infer *[7:00 am,10 am] : eats(paul)*
Explanation: If Paul was eating at some time between 8 to 9 in the morning, then he was also eating at some time between 7 to 10 in the mo rning.

[A4] From <u1,u2>:p and <u3,u4>:p infer <u0,u5>:p,
    where the temporal interval <u1,u2> overlaps with
the temporal interval <u3,u4>, and when concatenated they produce <u0,u5>.
Example:
From *<1/1/90,1/1/95> : bank_account (a1,john,$10000),*
and *<1/1/93,1/1/98> : bank_account (a1,john,$10000),*
infer *<1/1/90,1/1/98> : bank_account (a1,john,$10000)*
Explanation: If John has the bank_account '*a1*' with $10000 from 1/1/90 to 1/1/95, and the same bank_account form 1/1/93 to 1/1/98 then his '*a1*' bank account contains $10000 from 1/1/90 to 1/1/98.

[A5] From [u1,u2]:p infer [l1,l2]:p or [l3,l4]:p, where the temporal instance [u1,u2] is the concatenation of the temporal instances [l1,l2] and [l3,l4].
Example :
From *[monday,saturday] : leaves(john),*
infer *[monday,wednesday] : leaves(john)*
or *[tuesday,saturday] : leaves(john)*
Explanation: John will leave at some time during the coming week, i.e. at some time between Monday and Saturday. Therefore he will either leave at some time between Monday and Wednesday or at some time between Tuesday and Saturday.

2.2.2. Inference rules for events. For events the inference rules [A1] and [A4], do not hold. Rule [A2] holds only partially as presented in [B1] and [A3] holds exactly in the same way as presented in [B2].

[B1] From <u1,u2>:p infer [l1,l2]:p,
    where the temporal instance [l1,l2] contains the temporal interval <u1,u2>.
Example:
From *<1/1/93,1/1/95>:accumulates(peter,$10000)*
*in*fer *[1/1/90,1/1/96] : accumulates(peter,$10000)*
Explanation: If peter has accumulated $10000 during the time period 1/1/93-1/1/95, then he has accumulated $10000 at some time between 1/1/90-and 1/1/96.

[B2] From [u1,u2]:p infer [l1,l2]:p,
    where the temporal instance [l1,l2] contains the temporal instance [u1,u2].
Example:
From *[10:00 pm, 12:00 pm]:times_around(square,10)*
infer *[8:00 pm, 13:00 pm]:times_around (square,10)*
Explanation: If one has been running 10 times around the square at some time between 10:00 pm and 12:00 pm, then one has been running 10 times around the square at some time between 8:00 pm and 13:00 pm.

The inference rules for events and properties which we have presented refer to simple forms of temporal references, i.e. temporal intervals, temporal instances and temporal points. The simplicity of the presentation clarifies their usefulness and importance in temporal reasoning. More general inference rules containing uncertain temporal intervals, which have a more complex structure <[t1,t2],[t3,t4]>, have been presented in [11]. The inference rules presented here are special cases of those in [11].

3. Intelligent information processing

In the following paragraphs we will provide some interesting examples taken from various domains, in order to exhibit the expressive power of the *TRLi* system. The *TRLi* system is expressively rich, for many reasons:

- It can represent temporal points, temporal instances, temporal intervals, temporal periodic data, and temporal duration, in a straightforward and natural manner.
- It can reason about temporal relations using a Symbolic Constraint Solver.
- It is able to produce temporal deductions, which are chains of temporal inferences (in the form described in a previous section) and classical inferences, i.e. Prolog's SLD-resolution [13].

3.1. Intelligent Problem Solving

The workshop murder mystery is an interesting problem solving example which needs some help from a temporal logic in order to be expressed declaratively and to be solved efficiently. The workshop murder mystery has been previously solved [14] by the Annotated Constraint Logic Programming (ACLP) Temporal System, but the solution is based on Constraint Logic Programming (CLP) over finite domain variables. Moreover, the ACLP system cannot yet be considered to be a Temporal Logic but an application of CLP to temporal reasoning.

The *workshop murder mystery* concerns a workshop and a murder. In an afternoon session, after the coffee break, there are four talks, from 3:26 pm to 5:05 pm (*temporal intervals*). However, the last talk never took place, as the last of the speakers (Dr Roussau) was found dead at 5:35. The experts have decided that a murder must have taken place an hour to an hour and a half earlier (*relative temporal instance*). There are three suspects: Dr Kosta, Dr Roussau and Dr Dean, which present their alibis. Dr Kosta took the last shuttle to the airport to board the plane at 5:05 pm (*temporal point*). The shuttle leaves every half an hour (*recurrent*, *periodic data*) and it takes 50 minutes (*temporal duration*) to get to the

airport. Dr Dean was copying during the second talk: it took him 5 minutes to get to his room and find a copy of 30 slides in his room, another 5 minutes to get to the copy room, 15 minutes to copy the slides, and another 5 minutes to get back to the lecture hall, totally 30 minutes. Dr Roussau has been found murdered and the way he was murdered excludes a suicide. We must reason about the murder of Dr Roussau.

Dr Kosta's alibi holds, as he took the 3:30 shuttle to arrive at the airport at 4:20, to board on the 5:05 plane. Dr Dean's alibi does not hold because a talk takes 25 minutes while it took him 30 minutes to copy. Therefore he is the only true suspect.

The workshop murder mystery can be expressed in *TRLi* by defining a temporal world containing temporal rules and assertions. The assertions of the temporal world (*TempWorld*) are the data of *TRLi* which is able to make temporal (and classical) deductions on given queries. The *TempWorld* is as follows :

```
:-      properties([alibi/1,on_shuttle/1]).
:-      events([talk/3,found_dead/1,board_plane/1,
coffee_break/0, murdered/1, shuttle/2, copying/1]).
<3.00pm,3.25pm>:coffee_break.
<3.26pm,3.50pm>:talk(1,prof_Zadeh,fuzzy_deductions)
.
<3.51pm,4.15pm>:talk(2,dr_Wu,temporal_systems).
<4.16pm,4.40pm>:talk(3,dr_Dean,rule_based_systems)
.
5.35pm:found_dead(dr_Roussau).
[S-90 min,S-60 min ]:murdered(X) :-
        S:found_dead(X).
5.05pm:board_plane(dr_Kosta).
<T1,T1+50>:on_shuttle(X)          :-
        T1:shuttle(0.00pm,8.00pm),
        T2:board_plane(X),
        T1+50min<T2, T2<T1+80 min.
T:shuttle(T,S) :- T £S.
T:shuttle(A,W) :-
        A £W, B is A+30min, T:shuttle(B,W).
<T,T+30>:copying(dr_Dean) :-
        <A,W>:talk(2,X,Y), A £T, T£W,
        <B,C> : talk(_,dr_Dean,_), T+30£B.
<T1,T2>:alibi(X) :-
        (<T1,T2>:copying(X);
        <T1,T2>:talk(J,X,Y);
        <T1,T2>:on_shuttle(X)).
```

It is interesting to see how periodic data are represented in *TRLi*. The definition of the *shuttle* predicate represents such information: Given the query *T:shuttle(2.30pm,4.00pm)* the system answers *T=2.30pm, 3.00pm, 3.30pm, 4.00 pm.*

4

The queries are provided to *TRLi* by the following Prolog program. At the meta-level, the Temporal World is first read, and then the system tries to find out the murderer, taking into account the time of the murder which is a temporal instance, the murdered person, the possible suspects, and the alibi of each suspect for the corresponding temporal interval. A suspect which does not have an alibi can be accused to be the murderer. Given the query *murder(X,Y)* the system answers *X=dr_Dean, Y=dr_Roussau.*

```
/*  murder(X,Y) ⌐ X murdered Y  */
murder(X,Y)  :-
      world(TempWorld),

      Prove_in_TRL(TempWorld,[T1,T2]:murdered(Y)),
      suspect(X),X\=Y,

      not(Prove_in_TRL(TempWorld,<T1,T2>:alibi(X)))
.
suspect(dr_Kosta).
suspect(dr_Roussau).
suspect(dr_Dean).
```

3.2. Simulation

Simulation is an appropriate application domain of a temporal logic, due to its heavy utilisation of temporal objects and temporal constraints. *TRLi* can be used for simulation of time-varying circumstances. A simple example which is often used [3] is the simulation of a computer's cpu state. This example is also presented here in the context of *TRLi*.

```
:-
properties([cpu/2,job_queue/1,initial_queue/1,job/1]).
:- events([jobs/1]).
1:initial_queue([(job1,1),(job2,2),(job3,2)]).
T:cpu(idle,0) :-
      T:job_queue([]).
T:cpu(X,N) :-
      T:job_queue([(X,N)|Rest]).
T:job_queue(X) :-
      T1:initial_queue(Jobs),
total_time(T1,Jobs,Total),
      between(T1,Total,T),
      T2 is T-T1, pending_jobs(T2,Jobs,X).
T:job(X) :-
      T:job_queue([(X,N)|Rest]).
T:jobs([L]) :-
      T:job(L).
<S,E>:jobs([L|Ls]) :-
      S:job(L), <S+1,E>:jobs(Ls).
```

The predicate *T:initial_queue(Jobs)* defines the list of jobs (along with their duration) waiting for execution at the starting point *T*. The predicate *T:cpu(X,N)* returns the job *X* executed in the cpu at time *T*, and the remaining cpu time *N* for the job to be finished. The predicate *T:job_queue(Jobs)* returns the remaining job queue (*Jobs*)at *T*. The predicate *T:job(X)* returns in *X* the first job in the queue at *T*. The predicate *<B,E>:jobs(Jobs)* returns in *Jobs* the list of jobs executed during the interval *<B,E>*.

The classical predicates *total_time*, *between*, and *pending_jobs* are defined as follows: The predicate *total_time(T1,Jobs,Total)* computes the total time (*Total*) needed for all jobs to complete their execution. The predicate *between(T1,Total,T)* returns in *T* the time points between *T1* and *Total*. The predicate *pending_jobs(T,Jobs,X)* returns in *X* the jobs which are still pending at *T*, given that all the jobs are stored in the *Jobs* variable. Notice that the classical predicates are not labeled by any temporal reference. They are handled by the system as normal Prolog predicates.

Given the above program we may ask several questions:

Query 1. List the jobs *X* which are executed during the interval *<1,3>* :
      Query : <1,3>:jobs(X).
       Ans :   X=[job1,job2,job2]

Query 2. List the jobs *X* which are executed at some time between 1 and 3 (temporal instance [1,3]):
      Query: [1,3]:jobs(X).
      Ans :   X=[job1], X=[job2], X=[job2], X=[job1,job2], X=[job1,job2,job2], X=[job2,job2].

Query 3. List the jobs *X* which are executed during the interval *<4,E>* :
      Query : <4,E>:jobs(X).
      Ans :   E=4; X=[job3], E=5; X=[job3,job3].

Of course, simulation is a much more complex procedure. *TRLi* can be used as a basis for developing a simulation system supported by a temporal logic.

3.3. Temporal Databases

Issues on temporal databases have been examined in great depth during the last few years [15],[16],[17]. Many problems however, are still under research [18],[19]. Moreover, temporal logics have been applied for various issues concerning temporal databases [18],[20].

In the following paragraphs we will only provide some issues on using *TRLi* as a temporal database supervisor. More specifically, we will not cope with the problem of supervising a temporal database for insertions and deletions in order to conserve its consistency and/or completeness [18], but only with some of the issues concerning intelligent retrieval of temporal data.

The following example is similar to an extended example found in [21]. Assume that the domain of interest consists of three tables, i.e. a salary table, a department table, and a manager table, as follows :

1. Salary Table

Start	End	Name	Salary	Tref:salary(Name,Salary)
6	44	Peter	15 K	*<6,44>:salary(peter,15K)*
45	49	Peter	20 K	*<45,49>:salary(peter,20K)*
50	55	Peter	25 K	*<50,55>:salary(peter,25K)*
0	15	John	20 K	*<0,15>:salary(john,20K)*
36	46	John	30 K	*<36,46>:salary(john,30K)*
66	80	Mary	25 K	*<66,80>:salary(mary,25K)*
26	80	Paul	23 K	*<26,80>:salary(paul,23K)*
0	39	Helen	25 K	*<0,39>:salary(helen,25K)*
45	80	Helen	25 K	*<45,80>:salary(helen,25K)*

2. Department Table

Start	End	Name	Dept	Tref:dept(Name,Dept)
6	39	Peter	Shoes	*<6,39>:dept(peter,shoes)*
40	55	Peter	Hardware	*<40,55>:dept(peter,hardw)*
0	15	John	Clothing	*<0,15>:dept(john,clothing)*
36	46	John	Credit	*<36,46>:dept(john,credit)*
66	80	Mary	Credit	*<66,80>:dept(mary,credit)*
26	80	Paul	Shoes	*<26,80>:dept(paul,shoes)*
0	39	Helen	Toys	*<0,39>:dept(helen,toys)*
45	80	Helen	Toys	*<45,80>:dept(helen,toys)*

3. Manager Table

Start	End	Mangr	Dept	Tref:manager(Mangr,Dept)
6	39	Peter	Shoes	*<6,39>:manager(peter,shoes)*
40	44	Paul	Shoes	*<40,44>:manager(paul,shoes)*
36	42	John	Credit	*<36,42>:manager(john,credit)*
66	80	Mary	Credit	*<66,80>:manager(mary,credit)*

Using this approach and some syntactic sugar of standard Prolog, it is very easy to retrieve answers of complex queries:

Query 1. List the names N and their salaries S of all employees in the department D (e.g. Shoes) when M (e.g. Peter) was a manager.
query1(M,D,N,S) :- <B,E>:manager(M,D),
[B,E]:dept(N,D),   [B,E]:salary(N,S).
?-query1(M,D,N,S):

Ans Manager	Department	Employee	Salary
peter	*shoes*	*peter*	*15*
peter	*shoes*	*paul*	*23*
paul	*shoes*	*paul*	*23*
john	*credit*	*john*	*30*
mary	*credit*	*mary*	*25*

Query 2. Compare the salaries of N1 (e.g. Peter) and N2 (e.g. John), when they were both employed by the company.
query2(N1,S1,N2,S2):-
<B,E>:dept(N1,D1), <B,E>:dept(N2,D2),
N1 ¹N2,
[B,E]:salary(N1,S1), [B,E]:salary(N2,S2).
?-query2(peter,S1,N2,S2):

Ans Employee1	Salary1	Employee2	Salary2
peter	*15K*	*john*	*20K*
peter	*15K*	*john*	*30K*
peter	*15K*	*paul*	*23K*
peter	*15K*	*helen*	*25K*
peter	*20K*	*john*	*30K*
peter	*20K*	*paul*	*23K*
peter	*20K*	*helen*	*25K*
peter	*25K*	*paul*	*23K*
peter	*25K*	*helen*	*25K*

If we also required that N1 and N2 were both working at the same department while employed by the company, then we would ask the following query :
query2'(N1,S1,N2,S2):-
<B,E>:dept(N1,D), <B,E>:dept(N2,D),
N1 ¹N2,
[B,E]:salary(N1,S1), [B,E]:salary(N2,S2).
?-query2'(N1,S1,N2,S2):

Ans Employee1	Salary1	Employee2	Salary2
peter	*15K*	*paul*	*23K*

Query 3. List the names N of employees in the department D while the department did not have a manager.
query3(N,D) :- <B,E>:dept(N,D),
not (<S,F>:manager(M,D),<S,F> contains <B,E>).
?-query3(D,N):

Ans Employee	Department
peter	*hardware*
john	*clothing*
john	*credit*
paul	*shoes*
helen	*toys*

Query 4. List the names N1 of employees in the department D1 while the department D2 did not have a manager.

query3(D1,D2,N1) :- <B,E>:dept(N1,D1),
     not (<S,F>:manager(M,D2),<S,F> contains
<B,E>).

Similar answers exist for query 4. Notice that, the predicates *salary*, *dept*, *manager* are treated as properties. The infix predicate *<T1,T2> contains <S1,S2>*, is a temporal predicate which becomes true when the temporal interval *<T1,T2>* contains the temporal interval *<S1,S2>*, i.e. when $T1 \geq S1$ and $S2 \leq T2$.

In the case of large databases we do not have to retrieve the elements of the entire table into memory before *TRLi* starts working. It is very easy to upgrade *TRLi*'s control mechanism so that every time it (re)tries a new goal, it brings to its memory the (next) available entry of the given table. This is easy, with conventional databases, because the *TRLi* system works at the meta-level and we can therefore appropriately change its rule selection mechanism with only a few modifications. For relational databases we need an appropriate interface, on the development of which we are currently working.

4. Implementation issues

The presented temporal reasoning system has been implemented as a goal driven, depth first, Prolog meta-interpreter. The meta-interpreter first selects as input the temporal knowledge as a set of temporal assertions containing temporal rules and facts. It then also selects the query and processes it by trying to prove it, in a way similar to that of Prolog.

Each time a new goal is under observation, its type is obtained and the appropriate inference rules are triggered. The meta-interpreter considers four different types of predicates: *events*, *properties*, *classical predicates* and *temporal predicates*.

Events and properties are handled through the corresponding inference rules. These rules have been implemented as an equivalent set of temporal constraints. These constraints are provided as input to a symbolic constraint solver which checks for their consistency and solves them whenever possible.

Classical predicates are handled as normal Prolog predicates, while temporal predicates are relations among temporal arguments. These predicates are also implemented as sets of temporal constraints.

5. Conclusions

In conventional information systems temporal points and temporal intervals have already been implicitly used, as they store in their databases time dependent information. Such information is either dependent on a temporal point, or on a temporal interval. They must however explicitly provide, through procedures, all the appropriate control mechanism to handle the corresponding time dependent data.

When the need of temporal uncertainty comes to reality, due to incompleteness or unavailability of information, conventional information systems are unable to perform any computations. They just ignore such cases.

We have developed a temporal reasoning system, called *TRLi*, which is based on the semantics of a previously proposed temporal logic called TRL. *TRLi* is a practical tool, as it handles time as a first order component, but it is also expressive enough to represent some kind of temporal uncertainty about the future and the past. Programming in *TRLi* resembles with Logic Programming. In fact TRLi is an extension of Prolog. Therefore, it is also possible to incorporate pure classical predicates (without any temporal references) into a *TRLi* program.

The presented architecture of *TRLi* along with the examples of its use not only show its expressive power, but also indicate that it can be used as a key temporal deduction component of an Intelligent Information System taking care of the representation of temporal knowledge and the reasoning task by making time dependent inferences.

The *TRLi* system can be used as a system in which one can develop advanced temporal, logic oriented programs and applications that can serve in many ways :

- **Stand alone modules** which provide an automatic temporal control mechanism, and are **coupled** to existing information systems.
- **Intelligent software supervisors** which are called by information systems in order to automatically take decisions and supervise temporal database operations.
- **Temporal deductive mechanisms** encoded into intelligent information systems, for intelligent problem solving, simulation, temporal planning, scheduling, etc...

We are currently working on practical and large examples concerning planning, scheduling, simulation, temporal deductive databases, verification of computer architectures, etc.

References

[1] Galton A (1987) Temporal Logic and Computer Science: An overview, in *Temporal Logics and their applications*, (Galton A, Ed), pp. 1-52, Academic Press, London.

[2] Maiocchi R and Pernici B (1991) Temporal Data Management Systems: A Comparative View. *IEEE Trans. on Knowledge and Data Engineering*, Vol.3, No.4, 504-523, December 1991.

[3] Orgun M A and Ma W (1994) An overview of temporal and modal logic programming. In *Temporal Logic. First International Conference (ICTL'94)*, Lecture Notes in Computer Science No 827, (GABBAY D M and OHLBACH H J, Eds.), pp.445-479, Springer Verlag, Bohn.

[4] Vila L (1994) A survey on temporal reasoning in artificial intelligence. *AI Communications*, 7(1), 4-28.

[5] McDermott D (1982) A Temporal Logic for Reasoning about Processes and Plans. *Cognitive Science*, 6, pp.101-155.

[6] Allen J F, (1984) Towards a general theory of action and time. *Artificial Intelligence*, vol 23, pp.123-154.

[7] Ostroff J S (1989) *Temporal Logic for real-time systems*. Research Studies Press Ltd., John Wiley & Sons Inc., England, 1989.

[8] Jixin M and Knight B (1994) A General Temporal Theory. *The Computer Journal*, Vol.37,No.2,pp.114-123.

[9] Panayiotopoulos T and Spyropoulos C D (1994) TRL: a formal language for temporal references. In *Temporal Logic. Proceedings of the ICTL Workshop*, (OHLBACH H J, Eds.), pp.99-109, MPI-I-94-230, Bohn, Germany, July 1994.

[10] Frantzi K T, Panayiotopoulos T and Spyropoulos C D (1994) Extending Allen's relations for uncertain time points and uncertain intervals. Presented at the *7th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'94)*, Dublin, Ireland, September 1994.

[11] Panayiotopoulos T and Gergatsoulis M (1995) A Prolog like temporal reasoning system. Presented at the *13th IASTED International Conference on Applied Informatics*, Innsbruck, Austria, February 1995 (Proceedings forthcoming).

[12] Marinagi C C, Panayiotopoulos T, Spyropoulos C D (1995) Planning through the TRLi temporal reasoning system. Presented at the *10th International Conference on Applications of Artificial Intelligence in Engineering (AIENG 95),* Italy, July 1995.

[13] Lloyd J W (1987) *Foundations of Logic Programming*. Springer-Verlag, 1989.

[14] Fruehwirth T (1994) Annotated Constraint Logic Programming applied to Temporal Reasoning. Presented at the *6th International Conference on Programming Language Implementation and Logic Programming (PLILP'94)*, Madrid, Spain, September 1994.

[15] Snodgrass R, Ahn I (1986) Temporal Databases. In *IEEE Computer*, 19 (9), pp.35-42, September 1986.

[16] Snodgrass R (1993) An Overview of TQUEL. In *Temporal Databases: Theory, Design and Implementation*, pp.141-182, Benjamin/Cummings, 1993.

[17] Kokkotos S, Spyropoulos C D (1994) A framework for developing temporal databases. In *Proceedings of the DEXA'94 International Conference*, Lecture Notes in Computer Science No 856, pp.236-245, Springer-Verlag, Berlin.

[18] Bohlen M and Marti R (1994) On the Completeness of Temporal Database Query Languages. In *Temporal Logic. First International Conference (ICTL'94)*, Lecture Notes in Computer Science No 827, (Gabbay D M and Ohlbach H J, Eds.), pp.283-300, Springer Verlag, Bohn.

[19] Kokkotos S, Ioannidis E V, Panayiotopoulos T, Spyropoulos C D (1994) On the issue of valid time(s) in temporal databases. Submitted for publication, September 1994.

[20] Tanabe K and Suzuki A (1987) The application of Tense Logic to Database Behavioral Description. *Systems and Computers in Japan*, Vol.18, No.1, pp.33-40.

[21] Gadia S K (1988) A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on DataBase Systems*, vol. 13, No 4, pp.418-448, December.