STUDIA UNIV. BABEŞ-BOLYAI, INFORMATICA, VOLUME II, NUMBER 1, 1997

INCOMPLETE RELATIONAL DATABASES AS CONSTRAINT LOGIC PROGRAMMING

DOINA TATAR SORANA CAMPAN

Abstract. The purpose of this paper is to illustrate how constraint logic programming (CLP) can be used to reduce the incomplete information in a database for which some fixed types of restrictons are given. An operational semantics of this process is defined in terms of the semantics of logic programming.

1. Introduction

In [5] the author believes that "CLP is one of the most promising and stimulating areas in computer scence". The confluence between CLP and databases is part of a general trend by which different fields are explored in order to profit from their common properties. The integration of logic programming and databases extends the frontiers of the management for complex data instead of simple data. In [4] is described how is possible to build systems of coupling logic programming to databases, providing efficient data access. These systems (as for example QUINTUS-PROLOG)
contain an interface that is capable of recognizing the database predicates and treating them in a special way. Our paper presents by small examples some posibilities of treatement of incomplete databases by logic programming when the constraints are added. Our tool is the language Turbo Prolog. The presence of the relational operators {<,<=,>=,>,=,<>} make from the language Turbo Prolog a language "like" constraint logic language.

2. Incomplete databases

For a given database we assume that we have more informers (persons, statistic papers, etc.). As the information comes from different sources, there are some major problems that can appear: inconsistency and incompleteness. We will approach the second problem, considering that the first one has been solved.

Having multiple informers means every given information is correct, although it might be incomplete. One approach for reducing incompleteness is to use constraints referring to the databases.

For the databases we choose the relational model for representing the information. So there is a given number of attributes, and a given number of tuples. A special attribute will be the time index. In this way, what the databases represents in fact is the image of a smaller databases, placed in different time moments. For the "small"

Received by the editors: May 7, 1997.
*1991 Mathematics Subject Classification.*68N17, 68P15.
1991 CR Categories and Descriptors. F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic - *logic programming.*

databases we keep more states of the databases. In this way we have the evolution of data. This way of representation with constraints that are giving the evolution of the database (for example the monotonity of an attribute) allows modelling the general behaviour of the database.

Time	*Attribute A_1*	*Attribute A_2*	. . .	*Attribute A_n*
t_0	a_{11}	a_{12}	. . .	a_{1n}
t_0	a_{21}	a_{22}	. . .	a_{2n}
. . .	. . .	. . .	. . .	. . .
t_0	a_{i1}	a_{i2}	. . .	a_{in}
t_1	$a_{i+1,1}$	$a_{i+1,2}$	. . .	$a_{i+1,n}$
. . .	. . .	. . .	. . .	. . .
t_p	a_{m1}	a_{m2}	. . .	a_{mn}

Table 1. Model for relational databases with the time attribute

In the previous representation, the following notations were used:

- t_k (with $k = 0...p$) - the time index for a tuple. All the tuples having the same time index consist the image of the database in one moment.
- a_{ij} (with $i = 1...m, j = 1...n$) - the *j*-th attribute of the *i*-th tuple.

All these notations will be used later.

The number of tuples in time moment t_i can be different from the number of tuples in time momnet t_j $(i \neq j)$ as the database is dynamic.

In a database, the data can be classified in three types, with the mention that by data we understand a *given attribute* of a *given tuple*, there are *unknown data* - there is an infinite set of possible values for the data; *incomplete data* - there is some information about the data that restricts the infinit set of possible values (to a set with the cardinality greater than one); *known data* - there is an exact value for the data.

Some relations between different type of data can be shown. These will be discussed in the section dedicated to constraints.

For representing the content of a database, the following notations will be used for different type of data: *?n* -- for an unknown data, where *n* is an integer or a letter; $n_{11}-n_{1,i1}, \dots, n_{k1}-n_{k,ik}$ - for incomplete data, where n_{pq} are integers. For example [3-7,10-12] is the representation for {3,4,5,6,7,10,11,12}; *p* - for a known data, where *p* is an integer.

3. Constraints

A constraint is a sentence that shows a behaviour of the database. There are many types of constraints, which we divide into two classes : the BASE constraints and the GENERAL constraints.

3.1. *Base Constraints*

Definition 3.1. [3, 7] A *base constraint* gives the behaviour of precisely given attributes of given tuples. It can have one of the following forms:

a) $ref_1\ num_1\ relop$

b) $ref_1\ [num_1,\ num_2\ relop$

c) $ref_1\ ref_2\ relop$

Here ref_1 and ref_2 are simple algebric expressions with a single reference to an attribute (so there are two references or less in a constraint); *relop* is a relational operator; num_1 and num_2 are numbers and $[num_1, num_2]$ stands for an interval of integers.

Let us consider the database from the Table 2, and the following constraints B_1 (given in postfixed form):

$a_{21}\ 3\ =$
$a_{12}\ [5,10]\ =$
$a_{33}\ a_{34}\ >$
$a_{44}\ [20,70]\ \#$
$a_{12}\ a_{33}\ =$
$a_{21}\ a_{52}\ =$

where we used the notations from Table 2.

Time	A_1	A_2	A_3	A_4
0	5	?1	4	13
0	?2	4	9	8
0	6	45	?1	6
1	7	2	23	23
1	3	?2	5	5

(a)

Time	A_1	A_2	A_3	A_4
0	5	[7-10]	4	13
0	3	4	9	8
0	6	45	[7-10]	6
1	7	2	23	[0-19,71-100]
1	3	3	5	9

(b)

Table 2. (a) Database B_1; (b) Reduced form of database B_1

The last two constraints have special meaning. Although they are unknown (so we know nothing about them), beetwen two unknown data ,some relations can be described (equality, unequalities). The last two restrictions presented above are describing such relations.

Although good for representing base constraints, the notation used above is not useful when trying to present the **general** constraints. This fact force us to introduce another notation for referring to the attribute of a tuple - *oo.aa* (instead of a_{ij}). Here *oo* is the index tuple (equal with *i-1*) and *aa* is the attribute index (equal with *j-1*). The tuple index (*oo*) is considered related to the entire database (the time moment does not matter).

Before a **base** constraint the letter 'b' appears. Using this form for referring to an attribute, the above constraints will have the following form:

```
b $01.00 3 =
b $00.01 [5,10] =
b $02.02 $02.03 >
b $03.03 [20,70] #
b $00.01 $02.02 =
b $01.00 $04.01 =
```

After applying the constraints to the database, that will change as shown in Table 2.

3.2. General Constraints

The name of this type of constraints is **general**, because unlike **base** constraints, with a single constraint we give the behaviour of more tuples (eventualy all the tuples in the database).

With the help of general constraints we define a frame for the database, frame in which we force the database to fit. These restrictions can give the behaviour of the database in certain time moments, or its evolution in time.

The reference to an attribute for these types of constraints is a bit different from that in the case of **base** constraints. It is like $*tt.oo.aa* - where *tt* is the time index, *oo* and *aa* having the same meaning as in the case of base constraints. The difference is that the tuple index (*oo*) is considered related to the time moment specified, and not to the entire database.

The general form of this type of constraint is [7, 3]

$$LogExp\ (AlgExp_1(ref_{11},\ldots,ref_{1,i1}),\ldots,\ AlgExp_n(ref_{n1},\ldots,ref_{n,in}))$$

where *LogExp* is a logical expresion, $AlgExp_k$ are algebrical expresions, ref_{ij} are references to attributes.

The **general** constraints can be local, global, key, temporal or set constraints. From one type of constraint to another some more restrictions are added to the general form, or this form is changed a bit.

Local Constraints. A local constraint gives separate behaviour of the database inside specified time moments. To be clear we give the following two constraints, in which the first is considered to be base constraint (see 'b' before the constraint) and the second is a local constraint (letter 'l' before it):

b $00.02 5 < (constraint B_1)

l $00.00.02 5 < (constraint L_1)

The first restriction says that the attribute a_{13} has a value less then 5. In this way it is decribed the behaviour of a certain attribute from a certain tuple ("the third attribute of the first tuple should be less then 5").

The second constraint means that starting from time moment 0 (the first moment) the third attribute of each tuple (inside a time moment) should have a value less then 5. This way a general behaviour of the database is described, starting from a certain moment.

By modifying the time index, the moment starting from which the constraint should be satisfyed is changed.

As we already mentioned, there can be more references in a constraint, but for a clearer understanding we will use only two at the beginning. If the references have different time indexes, then the behaviour of the database inside a time moment is modelled related to another time moment. For example, the constraint:

l $00.00.00 $01.00.01 < (constraint L_1)

says that starting from the second time moment (01) the value of the second attribute should be greater that the values existed for the first attribute in the previous time moment.

For a better understanding we consider the database L_1 from Table 3 (a). By applying the constraint L_1, and consider as existing (and applied) constraints that give the equality between different unknown data, the reduced form of database L_1 is presented in Table 3 (b).

Time	A_1	A_2	A_3
0	1	5	3
0	4	7	?2
0	2	7	2
1	?1	6	3
1	2	8	23

(a)

Time	A_1	A_2	A_3
0	[0-5]	5	3
0	4	7	[0-5]
0	[0-5]	7	2
1	[0-5]	6	3
1	2	8	23

(b)

Table 3. (a) Database L_1; (b) Reduced form of database L_1

If there are two references with the same index both for time and ojbect, then the relation that the constraint describes, is beetwen the specified attributes of the same tuple.

If the object index is different (but the time index is still the same), then the constraint refers to all possible combination of different tuples inside a time moment.

The functional dependencies are also part of local constraits. They are working inside time moments.

As known, with a functional dependency we can find out the value of an attribute according to the value of other attributes. There can be $A_2 = f(A_1, A_3)$, where A_1, A_2, and A_3 are attributes.

If we know the form of function f then we can construct the constraint. For instance, if $f(x,y)=x+y$, the constraint will be like

1 $00.00.01 $00.00.00 $00.00.03 + = (constraint L_3) ex *P6*

If the form of the function is not known then we can still model the functional dependency in the following form

1 $00.00.00 $00.01.00 = *$00.00.02 $00.01.02* =
& $00.00.00 $00.01.00 = _ (constraint L_2)

where _ stands for implication.

The constraint says if the first and the third attributes of two tuples are equal, then the second attributes will also be equal.

After applying constraint L_4 to the input from Table 4 (a) we obtain the reduced form shown in Table 4 (b).

Time	A_1	A_2	A_3	A_4
0	2	?1	3	6
0	2	7	3	9
1	6	?2	2	8
1	?3	16	3	4
1	6	10	2	2

(a)

Time	A_1	A_2	A_3	A_4
0	2	7	3	6
0	2	7	3	9
1	6	10	2	8
1	?3	16	3	4
1	6	10	2	2

(b)

Table 4. (a) Database L_2; (b) Reduced form of database L_2

Global Constraints. A global constraint describes the global behaviour of the database no matter the time moment. The database is considered entirely (all time moments together).

The time index has no semnification here. The most important help of global constraints is the possibility to implement functional dependencies that are valid no matter of the time moment.

Let us consider a global constraint that has the same form as L_2. Notice that in front of the constraint the letter `g` appears.

g $00.00.00 $00.01.00 = $00.00.02 $00.01.02 =
& $00.00.00 $00.01.00 = _(constraint G_1)

Consider also the databases from Table 5. After applying the constraint G_1 we obtain a reduced form of the database:

Here the functional dependency was applied between tuples placed in different time moments (the first and the forth tuple).

Time	A_1	A_2	A_3	A_4
0	2	?1	3	6
0	2	7	3	9
1	6	?2	2	8
1	2	?3	3	4
1	6	10	2	2

(a)

Time	A_1	A_2	A_3	A_4
0	2	7	3	6
0	2	7	3	9
1	6	10	2	8
1	2	7	3	4
1	6	10	2	2

(b)

Table 5. (a) Database G_1; (b) Reduced form of database G_1

One can look other constraints presented in the local constraints paragraph as global constraints. The way they will work is similar to the way they work as local constraint, it`s like we had the big database in one time moment instead of consisting of several smaller databases.

Key Constraints. A key constraint gives the combination of attributes that are the key for the tuples. In the most simple (and frequent) case the key is formed by only one attribute. Still, the general form is the one presented previously (see General Constraints) on which some restrictions are added. The new form is

$$LogExp\ (ref_{11}\ ref_{12} =, \ldots, ref_{i1}\ ref_{i2} =, \ldots, ref_{n1}\ ref_{n2} =)$$

where

- *LogExp* accepts as operators *and* (&) and *or* (||);
- ref_{i1} and ref_{i2} (for $i=1, \ldots, n$) refer the same attribute of different objects;
- $ref_{11} \ldots ref_{n1}$ have the same object index;
- $ref_{12} \ldots ref_{n2}$ have the same object index, different from the previous one.

The following constraint is a key constraint (letter `k` appears in front of it):

k $00.00.01 $00.01.01 = (constraint K_1)

It says that the second attribute is the key attribute.

The key constraint is actually a strong form of local functional dependencies. They say that if beetwen two tuples the combination of key attributes is identical, then all the attributes should be equal. Which concludes the fact that these constraints give the identification of tuples. Two tuples that have the same key are equal (in fact is the same tuple). According to the standard terminology of key in databases.

The constraint K_1 is saying that the second attribute is the key attribute. That means that if inside a time moment there are two tuples with the same value for the second attribute, then all the other attributes have to have the same values (an intersection will be made between the existing values).

The fact that the key constraints act locally (inside a time moment) is very naturally. The key attributes identify the tuple. But in time, some attributes of a tuple change their values. For example, if we had a database that keeps the personnel of an interprise, then the attribute "age" will change its value from one time moment to another. That means that in different time moment it is natural to have different values for that attribute, although we are talking about the same person.

So a key constraint has two effects. One is that it sets the key, and the second is that it makes the reduction. The setting of the key is important, as other types of constraints (temporal and set constraints) use the identification of tuples made by key constraints.

Temporal Constraints. A temporal constraint gives the evolution of particular tuples, and the way the database evoluates in time.

For instance the monotony of an attribute can be modelled with such kind of constraints. The temporal constraints use the identifications of tuples made by key constraints.

The temporal constraint (notice letter 't' before it):

t \$00.00.01 \$01.00.01 < (constraint T_1)

says that the second attribute increases its value in time. This means that for tuples (placed in consecutive time moments) that have the same identification, the value of the second attribute has to be increased.

Time	A_1	A_2	A_3
0	2	?1	67
0	4	?2	78
1	4	6	45
1	2	4	56

(a)

Time	A_1	A_2	A_3
0	2	[0-3]	67
0	4	[0-5]	78
1	4	6	45
1	2	4	56

(b)

Table 6. (a) Database T_1; (b) Reduced form of database T_1

After applying the constraint T_1 (Table 6 (a)) (and considering that a key constraint that sets the first attribute as the key attribute has already been applied) the reduced form presented in Table 6 (b) will be obtained.

Set Constraints. A set constraint has two parts: a normal (local) constraint, and a set of numbers that give possible cardinalities of tuples that within one time moment will satisfy the first part of the constraint.

So the general form of the constraint is modified as the following (letter 's' appears in the beginning of set constraints):

$$s\ LogExp\ (AlgExp_1(ref_{11}, \ldots, ref_{1,i1}), \ldots, AlgExp_n(ref_{n1}, \ldots, ref_{n,in}))\ (n_1, \ldots, n_k) =$$

where $(n_1, \ldots, n_k)$ is the set of possible cardinalities, n_j (with $j = 1, \ldots, k$) are integers and all the references have the same time index and object index. The object index has to be the same, otherwise the set constraints are loosing their sense, because different references would cause choosing different tuples and as in all possible combinations they would appear, proper counting would not be possible. The time index has to be the same (as we refer in this first part of the constraint to the same tuple), and the constraint will be considered starting from the moment specified by this index.

What such a constraint says is that there are n_1 or n_2 or ... or n_k tuples that within one time moment satisfy the first part of the constraint.

The following set constraint:

$$s\ \$00.00.00\ 2 = (0,3)\ (\text{constraints } S_1)$$

says that there are 3 or there are no tuples that within one time moment have the first attribute with a value of 2.

After applying the constraint S_1 - Table 7 (a) as there are already two tuples that have the value of 2 for the first attribute (in time moment 0), this value will be excluded from the domain of possible values for the first attribute of the fourth tuple. Inside the second time moment the unknown data ?1 and ?2 will be set to the value of 2 as there have to be 3 or zero tuples with the value of 2 for the first attribute, and there already exists one with such value (so the cardinality zero is excluded). The reduced form of the database is presented in Table 7 (b).

Time	A_1	A_2	A_3
0	31	10	8
0	2	4	56
0	33	40	10
0	[0-10]	16	8
0	2	7	9
0	2	77	9
1	2	7	9
1	?1	64	39
1	?2	90	60

(a)

Time	A_1	A_2	A_3
0	31	10	8
0	2	4	56
0	33	40	10
0	[0-2,4-10]	16	8
0	2	7	9
0	2	77	9
1	2	7	9
1	2	64	39
1	2	90	60

(b)

Table 7. (a) Database S_1; (b) Reduced form of database S_1

4. Using constraint logic programming

CLP has a long tradition in computer science. It attempts to preserve the advantages of logic programming while removing their limitation, bz using constraint solving beside unification as an operational scheme.

The language considered here is essentially that of first-order predicate. Let:

- *P* be a set of predicate symbols. When these symbols correspond to a database we will call them *database predicates*;
- *C* be a set of constant symbols;
- *V* be a set of variable symbols.

An atom over $C \cup V$ is of the form

$$p(u_1, \ldots, u_n),\ n \geq 0,$$

where $p \in P$ with arity n, and each u_j is an element of $C \cup V$ or *lists* from these elements. Let us remark that we work with a simplified version of Prolog, caracterised by the absence of function symbols (as in Datalog).

If the arguments u_j are not interesting in a particular context, then we will denote an atom simply by p.

4.1. Base constraints

A database correspond to a *database predicate* $p \in P$, every tuple in a database is an argument of p which type is *list*. Every *ref* in section 1 is an element of a such list. If a tuple contains an unknown element, then the corresponding list contains a variable, on the corresponding position. If a tuple contains an element with a value interval, then the corresponding list contains a variable and the body of clause contains a constraint of the form b as below.

Let us remark that the only reason for considering the *database predicates* with only one argument, (of type list), and not with the number of arguments equal with the length of the tuples [4] is that of an easier querying of the programs.

Definition 4.1. The constraints corresponding to *base* constraints in section 1 are of the forms:

a) X <relop> num1,
b) X >= num1, X <=num2,
c) X <relop> Y.

where X and Y are variables, elements of the *lists*, and *relop* and *num1*, *num2* are as in section 1.

Definition 4.2. A constraint logic program P of the first type is a sequence of Horn clauses of the form :

$$p \leftarrow q_1, \ldots, q_n$$

where p is an atomic formula and $q_1, \ldots, q_n$ are either atomic formulas in first-order logic or constraints, the comma is the logic operation "and", and the sign $\leftarrow$ is "if" or reverse of the logical implication. A clause (a fact) can have an empty body.

We refer to the left (p) and right-hand side ($q_1, \ldots, q_n$) of a clause as its head and its body. A clause is logically interpreted as the universal closure of the implication $q_1 \wedge \ldots \wedge q_n \rightarrow p$.

The predicates that are in a head of a clause with a nonempty body, form intensional database (IDB) and the others form the extensional database (EDB).A predicate db that corresponds to an incomplete database DB or a predicate which correspund to a {non-base} constraint is allways in IDB.

Definition 4.3. A goal G consists of a conjunction of atomic formulas, $r_1, \ldots, r_t$ and is denoted by

$$\leftarrow r_1, \ldots, r_t.$$

Let us consider the CLP program for the database *B1*, from section 1. The *database predicat* name (here *b1*) corresponds to the batabase name, as in all the following examples:

```
domains
        lista=integer*
predicates
        b1(lista)
        c1(integer)
        c2(integer)
        c3(integer)
        member(integer,lista)
        append(lista,lista,lista)
        e(integer)
        lista_int(lista,integer)
clauses
        b1([0,5,X,4,13]):-c1(X).
        b1([0,X,4,9,8]):-c2(X).
        b1([0,6,45,X,6]):-c1(X).
        b1([1,7,2,23,X]):-c3(X).
        b1([1,3,X,5,9]):-c2(X).
        c1(X):-e(X),X>=0,X<=10,X>6.
        c2(X):-e(X),X=3.
        c3(X):-e(X),X<20,X>=0.
        c3(X):-e(X),X>70,X<=100.
        e(X):-lista_int(Y,100),member(X,Y).
        lista_int([0],0):-!.
        lista_int(Y,N):-M=N-1,lista_int(Z,M), append(Z,[N],Y).
        member(X,[X|_]).
        member(X,[_|T]):-member(X,T).
        append([],X,X).
        append([H|T],Y,[H|U]):-append(T,Y,U).
/* for the goal <----b1(X) we obtain all the 60 tuples in the reduced database B1 */
```

The *base* constraints of type a) and b) as in section 1 are done by the constraints of type a) and b) as above definition, and the constraints of type c) in section 1 are done by imposing the same body for the clauses. Here the atomic formula *c1(X)* in the first and the third clause corresponds to the {\bf base } constraint $a_{12}\ a_{33}$ = in the section 1.2. Analogously for the atomic formula *c2(X)* and the *base* constraint $a_{21}\ a_{52}$ =. The atomic formula *e(X)* corresponds to the implicit constraint that all the integers are less than 100. Let us remark that in [6], the author considers an similar way of enumerating integers.

4.2. General constraints

As is presented in section 1, a general constraint describes a constraint about the behaviour of the whole database. As the arguments of a *database* are lists, some *general* constraints can be still expressed by the CLP programs with a single argument of a database predicate representing a tuple of the database. The reson is that the type *list* takes over some informations like these about time in *local* constraints. Let us consider the database L_1, section 1.2. The corresponding CLP program is the following:

```
domains
        lista=integer*
predicates
        l1(lista)
        c1(integer)
        c2(integer)
        member(integer,lista)
        append(lista,lista,lista)
        e(integer)
        lista_int(lista,integer)
clauses
        l1([0,X,5,3]):-c1(X).
        l1([0,4,7,X]):-c2(X).
        l1([0,X,7,2]):-c2(X).
        l1([1,X,6,3]):-c1(X).
        l1([1,2,8,23]).
        c1(X):-e(X),X>=0,X<6.
        c2(X):-e(X),X>=0,X<6.
        e(X):-lista_int(Y,100),member(X,Y).
        lista_int([0],0):-!.
        lista_int(Y,N):-M=N-1,lista_int(Z,M), append(Z,[N],Y).
        member(X,[X|_]).
        member(X,[_|T]):-member(X,T).
        append([],X,X).
        append([H|T],Y,[H|U]):-append(T,Y,U).
        /* the clauses for c1(X) and c2(X) are the same because the minimal second attribut in the
           second time moment is 6 */
        /* for the goal l1(X) are obtaining the 25 tuples of the database L1 */
```

The method that will leave to express better *general* constraints is to consider the set of tuples as an entity, which can be entierely manipulate. Accordingly, a new predicat is needed, which forms a database from the tuples (the predicate *formlist*). The tuples are linked by this predicate and the general restrictions, about the whole database, can be expressed.

Let us consider the CLP program for the database L_2. The *database* predicat name is l_2. The constraint L_2 means that, for each two lists, if the second and the fourth elements are equal, than the third elements are also equal.

```
domains
        lista=integer*
        llista=lista*
predicates
        formlist(lista,lista,lista,lista,lista,llista)
        l2(llista)
        member(integer,lista,integer)
        member(lista,llista,integer)
clauses
        l2([L1,L2,L3,L4,L5]):-L1=[0,2,X1,3,6], L2=[0,2,7,3,9],
                L3=[1,6,X2,2,8], L4=[1,X3,16,3,4],
                L5=[1,6,10,2,2], formlist(L1,L2,L3,L4,L5,L),
                member(U,L,P1), member(V,L,P2),
                P1<>P2, member(Z,U,2), member(Z,V,2),
                member(W,U,4), member(W,V,4),
                member(X,U,3), member(X,V,3).
        member(H,[H|_],1).
        member(E,[_|T],N):-member(E,T,M),N=M+1.
        formlist(L1,L2,L3,L4,L5,[L1,L2,L3,L4,L5]).
```

Finally, the program for the database T_1 is as the following. Remember that the temporal constraint T_1 says that for tuples in consecutive time moments, which have the same identification, the value of the second attribute is to be increased.

```
domains
```

```
        lista=integer*
        llista=lista*
predicates
        formlist(lista,lista,lista,lista,llista)
        t1(llista)
        member(integer,lista,integer)
        member(lista,llista,integer)
clauses
                /* t1 succeds with the goal:
                t1([[0,2,2,67],[0,4,4,78],[1,4,6,45],
                [1,2,4,56]]) and fails with the goal:
                t1([[0,2,8,67],[0,4,4,78],[1,4,6,45],
                [1,2,4,56]])  */
        t1([L1,L2,L3,L4]):-L1=[0,2,X1,67], L2=[0,4,X2,78], L3=[1,4,6,45],
                L4=[1,2,4,56], formlist(L1,L2,L3,L4,L),
                member(U,L,P1), member(V,L,P2), P1<P2,
                member(A1,U,2), member(A1,V,2),
                member(0,U,1), member(1,V,1),
                member(X1,U,3), member(A2V,V,3),
                X1<=A2V,X2<=A2V.
        member(H,[H|_],1).
        member(E,[_|T],N):-member(E,T,M),N=M+1.
        formlist(L1,L2,L3,L4,[L1,L2,L3,L4]).
```

5. Computation in CLP

A computation in a constraint logic program can be described as a goal-directed derivation procedure from the initial goal using the program clauses. A *computation state* [11] is defined as a pair $s=(g, \sigma)$ where g is the multiset of atoms and constraints to be solved, and σ is the set of constraints accumulated so far. (The empty set of constrains is Φ)

A transition from a computation state $s = (g, \sigma)$ to another, $s' = (g', \sigma')$, is defined as the following rewriting relation "$\Rightarrow$":

Definition 5.1.

$$s \Rightarrow s'$$

iff

- there exists an atom $a \in g$, selected by a computation rule and a clause, renamed to new variables:

$$h :- h_1, \ldots, h_m$$

  such that a and h have the same predicat symbol. Then,

$$g' = (g \setminus \{a\}) \cup \{h_1, \ldots, h_m\} \text{ and } \sigma' = \sigma \cup \{a=h\}$$

  Here the expression $a = h$ is an abreviation for the conjuction of equations between corresponding arguments of a and h, and $s' = (g', \sigma')$.

- there exists a constraint c in the goal part that can be satisfied with the constraint store. In this case, s' is obtained from s by removing this constraint from the goal part, and by adding this to the constraint store σ. Thus, s' = (g', σ'), where:

$$g' = g \setminus \{c\},\ \sigma' = \sigma \cup \{c\}$$

Let us observe that the constraint store σ is always consistent.

A *computation* is a sequence: $s_1 \Rightarrow s_2 \ldots \Rightarrow s_n$. If $\Rightarrow^*$ denotes the reflexive and transitive closure of the relation $\Rightarrow$, then the above computation can be denoted as $s_1 \Rightarrow^* s_n$.

Remark 5.2. The SLD refutation in logic programming requires that the atom in a goal $ g_i $, which is rewritten in a step by the computation rule, must be the leftmost atom.

Definition 5.3. A computation state $s_n = (g, \sigma)$ of a finit computation

$$s_1 \Rightarrow s_2 \ldots \Rightarrow s_n$$

is *terminal* if one of this conditions is fulfiled:

a) the goal part g is empty (and is denoted by ϕ), or else,

b) no computation state s' exists such that $s_n \Rightarrow s'$.

Definition 5.4. A finit computation is *successful* if the terminal state has an empty goal, and *fails* otherwise (case b). In this case we say that s_1 fails. If (ϕ, σ) is a terminal state of a succesful computation,any set X of variables, such that the constraints σ are fulfilled is an *answer constraint* [12].

The operational semantics of a database *DB* reprezented by the predicate *db*, can be defined as the set of answer constraints *X*, such that from $(db(X), \Phi)$ starts a successful computation. This set is denoted by *succes(db)*:

$$succes(db) = \{X | (db(X), \Phi) \Rightarrow^* (\phi, \sigma)\}$$

Let us remark that σ is any consistent set of constraints. In our examples from section 1, the *reduced form* of a database with the name *DB* is the set *succes(db)*. For each *X* a tuple of the database *DB* is obtained.

To formalise the way our programs are queried with some ground goals, let us introduce the following two sets:

$$ground^+(db) = \{d \mid (db(d), \Phi) \Rightarrow^* (\varphi, \Phi)\}$$

and

$$ground^-(db) = \{d \mid (db(d), \Phi) \text{ fails}\}$$

In our example, the goal with the answer "yes" was of the form *db(d)* with *d* from $ground^+ db(d)$ and the goal with the answer "no" was of the form *db(d)* with *d* from $ground^- db(d)$.

6. Conclusions

The mathematical properties of the incomplete databases are insufficientely treated from a point of view of theoretical fundation. In [13] a model-theoretical of nulls in a relational database based on modal logic is presented. The treatment of an incomplete database by the formalisme of CLP presents some evident advantages. One of this is formalisation of semantics in terms of logic programming semantics. Moreover, as the translation from a system of equations in relational algebra to a logic program and conversely is allways possible [4] the introduction of some new relational operators (which can express the peculiarityes of the incomplete databases) represents, we think, a tool for generating some sound constraint logic programs.

REFERENCES

[1] K.R. Apt, M.H. van Emden, *Contribution to the theory of logic programming* Journal of ACM, vol. 29 (1982), pp. 841-862.

[2] K.R. Apt, D. Pedreschi, *Studies in pure Prolog*, CWI Report CS-R9048, September 1990.

[3] S. Campan, *Incomplete databases. Reducing the incomplete information by constraints*, Babes-Bolyai Univ., Master Thesis, 1996. (in romanian)

[4] S. Ceri, G. Gottlob, L. Tanca, *Logic Programming and Databases*, Spriger-Verlag, Berlin, 1990.

[5] J. Cohen, *Constraint logic programming languages*, Communications of the ACM, vol. 33 (1990), pp 52-68.

[6] Colmerauer, *An introduction to Prolog III*, Communications of the ACM, vol. 33 (1990), pp. 69-90.

[7] P. Domonkos, *Representing databases under constraints*, Report LIA, Univ. de Savoie, 1994.

[8] M. Falaschi, G. Levi, M. Martelli, G. Palamidessi, *Declarative modelling of the operational behaviour of logic languages*, Report Univ. di Pisa, TR-10 (1980).

[9] H. Gaifman, H. Mairson, Y. Sagiv, M.Y. Vardi, *Undecidable Optimization Problems for Database Logic Programs*, Journal of ACM, (1993), pp. 683-714.

[10] A. van Gelder, K.A. Ross, J.S. Schlipf, *The Well-Founded Semantics for General Logic Programs*, Journal of ACM (1991), pp. 620--651.

[11] P. van Hentenryck, H. Simonis, M. Dincbas, *Constraint satisfaction using constraint logic programming*, Artificial Intelligence, vol. 58/1-3 (1992), pp. 113-161.

[12] J. Jaffar, M.J. Maher, *Constraint logic programming: a survey*, J. Logic programming, vol. 19-20 (1994), pp. 503-581.

[13] K.L.Kwast, *The incomplete database*, Proceedings of IJCAI (1991), pp. 897-902.

[14] J. Minker, *Perspective in deductive databases*, J. Logic Programming, vol. 5 (1988), pp. 33-61, J. Automated Reasoning, vol. 5 (1989), pp. 167-205.

[15] D. Tatar, *Term rewriting systems and completion theorems proving* Studia Univ. Mathematica, 1992, pp. 117-127.

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. Kogalniceanu 1, România.

E-mail address: dtatar@cs.ubbcluj.ro

Universite de Savoie, Laboratoire Intelligence Artificielle, 73376 Le Bourget du Lac, France.

E-mail address: sorana@lia.univ-savoie.fr