ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

Querying and Updating Constraint Databases with Incomplete Information

Julie Wilks

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

ISBN: 3-540-76227-2

Querying and Updating Constraint Databases with Incomplete Information

Julie Wilks

School of Computing and Information Systems

University of Wolverhampton

Wolverhampton

WV1 1SB

j.p.wilks@wlv.ac.uk

Abstract

It is the aim of the research described in this paper is to extend the work to deductive databases by proposing a theory of indefinite deductive constraint databases (IDCDs). This model provides a useful and natural representation of partial and infinite information. As a result of basing our representation of incomplete information on constraints, we can easily extend our model in order to enable the representation of temporal and/or spatial data by defining the domain of variables over which constraints can be defined to either be the temporal or spatial domain or by using, for example, linear arithmetic constraints to represent these domains. We describe how the syntax of the deductive database model can be enhanced with global conditions on incompletely specified constants to naturally represent partial information and provide a full declarative, fixpoint and procedural semantics for the language defined. We discuss an as yet unexplored problem with respect to constraint databases - that of updating and maintaining a consistent set of data in the presence of global conditions on incomplete information.

1 Introduction

Non-traditional applications in domains such as planning and scheduling, decision support, spatial information systems, CAD/CAM, scientific databases, etc. require a number of functionalities not traditionally provided by standard database models. These include the need to represent temporal and/or spatial information, the existence of both complete and incomplete representations for data, the ability to retrieve both conditional and definite answers to queries in the face of incomplete information and the ability to represent, query and dynamically update complex interrelated data.

In this extended abstract, we present the main concepts behind the ongoing development of a prototype deductive database which aims to provide this functionality through an integration with constraint handling similar to that proposed by Koubarakis [9] for relational databases. We motivate this work using the following example.

Example 1 *Routing and scheduling* applications are concerned with collections of entities (eg locations, depots) requiring service by a fleet of vehicles. Real-world routing and scheduling applications involve large amounts of data and multi-user access and would therefore benefit from database support. Applications in this domain require the representation of spatio-temporal information, for example, routes can be represented as spatially embedded networks and the times at which vehicles service locations on the route needs to be stored, as well as basic attribute data about these vehicles and entities. Information about these values may be incompletely specified but still needs to be represented and maintained prior to the generation of a schedule. Indeed, even once a schedule has been generated incompletely specified information may still exist. For example, the actual times at which locations are scheduled may not be known at the outset, however, such

details as the latest time by which a location must be serviced, that location A must be serviced before location B, etc, is known.

In addition to the problem of how to model such information, such representations are not static. For example, new information about the possible times that locations can be serviced may become known at any time and existing information may be found to be incorrect and need to be deleted.

Analysis of this and similar applications raises several issues not adequately dealt with by existing general purpose systems, in particular, the presence of spatio-temporal data represented as continuous values, the presence of incompletely specified values, which may also be continuous, and interrelated (incomplete) specification of values, for example, that one event occurred before another.

There have been numerous attempts to extend existing data models in order to deal with this sort of incomplete information (see, for example, [1, 4, 8, 10]) however, these extensions either concentrate on discretely representable, independent, incomplete values or definite representations of continuous data. In common with other researchers, however, we believe that many applications, for example applications involving temporal or spatial reasoning, can benefit from the development of a database model which allows the representation of definite, indefinite, finite and infinite information in a single unifying framework and that this can best be achieved through an integration of formalisms developed in both Artificial Intelligence and databases. Such an integration will not only provide the expressive power required for such applications, but also the necessary functionalities for more general forms of querying, dynamic updates, concurrency, recovery mechanisms, etc. We see these as necessary functionalities given the large amounts of data which these applications are now generating, the dynamic nature of that data and the need for multi-user access to the data. The approach we take is based on an integration of constraint databases, deductive models for dealing with incomplete data at the level of value incompleteness” and formalisms for dealing with constraints.

Constraints have been extensively used to represent continuous data, such as temporal intervals or spatial objects, and have recently been integrated into database query languages. In [8] the Constraint Query Language (CQL) scheme was introduced which directly supports the representation of continuous data. In this scheme a “constraint tuple” is a finite representation of an infinite fact, such as an interval, a line or a region. A traditional data value is a special case of constraint tuple where the only constraint is equality.

In [9] the CQL scheme was extended for relational databases to allow the representation of global constraints on existentially quantified variables appearing in constraint tuples. This extension enables complex partial information such as described above to be represented.

It is the aim of the research described in this paper is to extend the work reported above to deductive databases by proposing a theory of indefinite deductive constraint databases (IDCDs). This model provides a useful and natural representation of partial and infinite information. As a result of basing our representation of incomplete information on constraints, we can easily extend our model in order to enable the representation of temporal and/or spatial data by defining the domain of variables over which constraints can be defined to either be the temporal or spatial domain or by using, for example, linear arithmetic constraints to represent these domains.

The contributions of this paper are as follows:

- We describe how the syntax of the deductive database model can be enhanced with global conditions on incompletely specified constants to naturally represent partial information and provide a full declarative, fixpoint and procedural semantics for the language defined.
- We present a deductive query language which can be used to reason with information represented in this model and address the issue of bottom-up evaluation in this framework.
- We discuss an as yet unexplored problem with respect to constraint databases - that of updating and maintaining a consistent set of data in the presence of global conditions on incomplete information.

This paper reports on work in progress and some of the ideas are presented through examples only.

2 Syntax and Semantic

Work on representing partial information in deductive databases describes their semantics as a set of databases, each one corresponding to a possible world but requiring more information to know which one which corresponds to the actual state of the world [2]. This imaginary scenario is unsuitable for direct implementation due to the prohibitive expense of storing multiple databases [13] particularly where the domain for any partial information is continuous. A more compact representation of the candidate databases is required for the sake of efficiency. Several formalisms currently exist, however, these are inadequate for representing the sorts of application domains in which we are interested.

In this section we lay the foundations for the implementation of a deductive constraint database with indefinite information, based on an integration of semantic concepts from deductive databases with null values [10, 4] constraint databases [9, 10] and constraint logic programming [5, 6, 7] by developing the model of indefinite deductive L -databases where L , the parameter, is a first order constraint language. In this model IC-constants represent incompletely specified data. Global conditions specify what is known about these IC-constants at assertion time.

2.1 Language syntax

The Indefinite Deductive L -Constraint Database ($IDC(L)$) model extends the concepts of definite deductive databases by allowing constraint terms known as C-constants in facts. These constants can have one of two interpretations.

1. *Definite C-constants (DC-constants)* represent infinite (continuous) data, for example, temporal intervals or spatial attributes such as lines or regions. Bounds on these constant values are represented by local constraints. Using constraints in this way is the subject of most work on constraint databases and we only describe the details of how DC-constants are used and interpreted where necessary to highlight the semantics of $IDC(L)$.
2. *Indefinite C-constants (IC-constants)* represent values which are known to exist but which are imprecisely or incompletely specified. The partial specification of such constant values is represented by *global conditions*.

We distinguish C-constants from ordinary variables because of their different interpretation.

We assume familiarity with deductive and constraint databases and introduce the following extended definitions.

A *term* can be any one of the following: a variable, a constant, a DC-constant or an IC-constant. An *atom* takes the form $p(t_1, \dots, t_n)$ ($n \geq 1$) where p is a predicate, the t_i are terms and c is a (possibly empty) conjunction of constraints. A ground atom (or fact) is an atom which only contains constants.

An $IDC(L)$ clause takes the general form $(A_0 \leftarrow C \wedge A_1 \wedge \dots \wedge A_n, c)$ where each of the A_i is an atom, C is a conjunction of query constraints and c is a constraint expression. A_0 is the head of the clause, $C \wedge A_1 \wedge \dots \wedge A_n$ is called the body of the clause and c is called the domain specification of the clause. A goal is a program clause with no head. If there are no IC- or DC-constants in a clause then $c = \emptyset$.

We use the notational conventions of representing normal constants and predicate symbols as strings beginning with a lower case character and general variables as strings beginning with an upper case character. We require that atoms with the same predicate symbols are of the same arity, that is they have the same number of arguments. DC-constants are denoted by lowercase letters of the English alphabet (for example, x, y, z, t), possibly subscripted and IC-constants by letters from the Greek alphabet (for example, ω).

A *constraint expression* c is a conjunction of closed L -formulas (constraints) which place bounds on the values of an IC-constant ω_i (resp DC-constant x_i). The partially solved form of c is called the range of ω_i (resp x_i).

An atom A is an *infinite L-constraint fact* $(p(t_1, \dots, t_m), c)$ if and only if $\exists t_i \in p$ such that t_i is a DC-constant and all other terms in p are either normal constants or DC-constants. Each infinite L -constraint fact

p is associated with a conjunction of L-constraints $l(p) \in c$ over DC-constants appearing in p called the local condition of p .

An atom A is an *indefinite L-constraint fact* if and only if $\exists t_i \in p$ such that t_i is an IC-constant and all other terms in p are either normal constants or DC-constants or IC-constants. Each indefinite L-constraint fact p is associated with a boolean combination of L-constraints $g(p) \in c$ called the global condition of p . The collection of global conditions defined for predicates in D is known as the *global constraint store* G . It is assumed that G is always consistent.

An IDC(L) database consists of a finite set of clauses that satisfy the following conditions:

- Each infinite or indefinite constraint fact appearing in D is ground, that is, they contain no ordinary variables.
- If an IC-constant ω_i appears in two different clauses, then the range of ω_i is the same for both clauses.
- Each fact of DB is ground.
- Every variable occurring in the head of a rule must also occur in the body of the same rule.
- Each DC- and IC-constant occurring in a condition in the body of a rule must also occur in a predicate in the body of the same rule.

We also require all predicates with the same name to be of the same arity.

Conditions in the body of a rule cannot be used to manipulate the values of DC- and IC-constants stored in the database, unless they appear in an update clause. Rather, they are collected and solved during query evaluation to restrict the facts generated for a particular rule.

For the purposes of query evaluation and optimisation, the database D is partitioned into three sets: the extensional database D_B , the set of ground and constrained user atoms in D , the intensional database D_D , an IDC(L) program and the global constraint set GCS_D , the set of constraints on IC-constants appearing in DB . The predicates occurring in D_B and D_D are divided into two disjoint sets: the D_B -predicates which are all those occurring in D_B and the D_D -predicates which are all those occurring in D but not in D_B . We require that the head predicate of each rule in D be a D_D -predicate, D_B -predicates may occur in D_D but only in the body of clauses. The global constraint store (GCS) is also partitioned into two disjoint sets: the set of unary constraints UC_D and the set of binary constraints BC_D .

The following example highlights the syntax of IDC(L) where L is instantiated with the arithmetic constraint domain $\mathbf{Z}$. The example shows the use of IC-constants to represent indefinite information only.

Example 2 Let us consider a portion of a simple genealogy database. This database contains information about possible ancestors and the year of their birth represented by the atom

person(First, Last, DoB).

Years are integer values whose range is drawn from the integer arithmetic constraint domain denoted by $\mathbf{Z}$. Names of individuals are always fully known but we may not know, or may only have partial information about, dates of birth.

person(arthur, hayward, 1896).

This first ground atom is a normal deductive fact which states that the date of birth of Arthur Hayward is known to have been 1896. The following atoms contain indefinite information

person(abraham, hayward, ω_1).
person(alfred, hayward, ω_2).
person(alice, hayward, ω_3).

Conditions on the IC-constants appearing in the $\mathbf{Z}$ -constraint facts provide all currently known information about those values and are held in the *global constraint store* $g(\text{person})$ shown below:

$$g(person) : 1860 \leq \omega_1, \omega_1 \leq 1870, \omega_1 - 15 \geq \omega_2, \\ \omega_2 \leq 1845, \omega_3 \leq 1825, \omega_2 - \omega_3 \leq 20.$$

The global condition states that the date of birth of Abraham Hayward is known to have been sometime prior to 1870 and after 1860 and the date of birth of Alfred Hayward is assumed to be at least 15 years prior to that of Abraham and 20 years later than that of Alfred. Finally, Alice Hayward is known to have been born sometime prior to 1825 and 20 years before Alfred. We also define a rule that a person is a possible ancestor of another person if they have the same surname and were born at least 15 years apart.

$$\begin{aligned} possible_ancestor(First1, First2, Last) \leftarrow \\ person(First1, Last, X), \\ person(First2, Last, Y), \\ Y \leq X + 15. \end{aligned}$$

In this rule the query constraint $Y \leq X + 15$ is used to restrict the derivation of facts using the rule.

2.2 Language semantics

IDC(L) has a full declarative, procedural and fixpoint semantics based on extending existing database semantics with IC-constants and conditions and taking account of the indefinite nature of the data through the provision of both definite and possible model semantics. This extension is heavily based on approaches to representing nulls in deductive databases (see, for example, [10]).

The declarative semantics is based upon an extension of the Herbrand universe with IC-constants. This is used to generate a 'quasi' Herbrand Base with IC-constants. Information about IC-constants is derived from the solution universe of the database. This is the set of all interesting constraints achieved by projecting out each IC-constant from the global constraint set and provides a lower and upper bound on the value of each IC-constant (known as the solved form of its associated set of conditions). By attaching relevant constraints to predicates appearing in the Herbrand Base we achieve the generalised Herbrand Base with indefinite values. From this we generate two types of interpretations - definite interpretations, which represent facts true in every possible world of the database, and possible interpretations which represent values true in at least one possible world of the database. A minimal model semantics is generated from these models by taking their intersection.

We begin by extending some definitions for definite constraint databases.

Definition 1 (Valuation) A valuation $\theta : C \rightarrow M$ is a mapping from $C \rightarrow M$ which maps an IC-constant C to an element of the constraint domain L , a DC-constant to a interval over the constraint domain and a constraint c to a closed L^* formula of the structure D . If X is a set of facts, each denoted by A , containing IC- and/or DC-constants, then

$$[X]_D = \{a | a \rightarrow c \in X, D \models c\}$$

Definition 2 (Solution) A constraint c is considered solvable if $\exists \theta : M \models \theta c$. That is, there is a valuation θ which satisfies the constraint c . θ is called a solution to c . A constrained atom is solvable, therefore, if c is solvable.

The constrained atom $(p(t_1, \dots, t_m), c)$ is an atom with distinct IC- and DC-constants as arguments and a conjunction of global and local constraints on these constants. Each constrained atom with DC-constants represents a possibly infinite set of facts which satisfy the local conditions on those constants.

Each constrained atom with IC-constants represents a "set of possible worlds" which could be obtained by assigning a single value to each constant which satisfies the global constraint(s) associated with it. That is, each IC-constant, together with its global constraint(s) is a finite representation of a possibly infinite set of disjunctive values.

The semantics of both IC-constants and DC-constants is defined by the set of their domain instances (see [] for the introduction of this notion in another setting).

Definition 3 (Domain instances) The (set of possible) domain instances $[(p(t_1, \dots, t_m); c)]$ of a constrained atom $(p(t_1, \dots, t_m); c)$ is defined as

$$[(p(t_1, \dots, t_m); c)] = \{p(t_1, \dots, t_m) | \theta \text{ is a solution of } c\}$$

Thus, the true (as yet unknown) value of an IC-constant is an instantiation to a single value from the set of possible domain instances for that constant. The true (known) value of a DC-constant, on the other hand, is the set of domain instances for that constant.

The set of possible worlds for a predicate $(p(t_1, \dots, t_m); c)$, therefore, consists of the set of possible domain instances obtained by replacing each IC-variable with each possible instantiation according to the global constraints. The meaning of $(p(t_1, \dots, t_m); c)$ is given by the constraint domain.

Definition 4 (Solution set) The solution set of a predicate $(p(t_1, \dots, t_m); c)$ is the set of possible worlds for that predicate obtained as described above.

Definition 5 Equivalence Let $\mathcal{A}$ be the set of constrained atoms appearing in a database D . We define a preorder on $\mathcal{A}$ as follows

$$(p(t_1, \dots, t_m); c) \leq (p(t_1, \dots, t_m); c) \text{ iff } [(p(t_1, \dots, t_m); c)] \subseteq [(p(t_1, \dots, t_m); c)].$$

Equivalence induced by on the set of atoms is denoted $\equiv$. That is, two constraint atoms are equivalent if their solution sets are equivalent.

We next extend the notion of substitution to constraint solving as in [7].

Definition 6 (Consistent) [7] Let c_1 and c_2 be sets of constraints, then c_1 and c_2 is consistent iff $D \models c_1 \wedge c_2$.

Definition 7 (Entailment) [7] Let c_1 and c_2 be two constraints, c_1 entails c_2 iff $D \models c_1 \rightarrow c_2$. For the purposes of this thesis the tests for consistency and entailment are always combined with a function *infer* which computes from c_1 and c_2 a new set of simplified constraints. The above definitions provide the basis for two alternative definitions of constrained substitution.

Definition 8 (Constrained substitution) A constrained substitution θ is a finite set of the form

$$\{t_1/v_1, \dots, t_m/v_m\},$$

where the following conditions hold

1. Each v_i is either a variable or an IC-constant, and $v_i \neq v_j$ for $i \neq j$.
2. Each v_i does not appear in the corresponding t_i .
3. If v_i is a variable, then t_i is an arbitrary term. If v_i is an IC-constant then t_i can only be a normal constant or another IC-constant.

If A_1 or A_2 contains IC-constants then $A_1 \theta A_2$ iff either

1. $\forall c \in \{A_1, A_2\}, D \models \exists c$, or
2. $\forall c_i \in \{A_1\}, \forall c_j \in \{A_2\}, i \neq j, D \models c_j \leftarrow c_i$.

The former case is known as a possible constrained substitution, denoted by θP , and the latter as a definite constrained substitution, denoted by θD .

2.2.1 Declarative Semantics

In this section we define a declarative semantics for IDC(L) by generalising the existing semantics for constraint databases based on ideas presented in work on value-based incomplete databases. In this semantics, rather than generating subdatabases, we represent alternative models of the world through constrained constants. We develop the semantics using the following example where L is the domain of integers.

Example 3 Let database $D =$

$$\begin{aligned}
 &p(9, \omega_1). \\
 &p(\omega_2, \omega_3). \\
 &p(\omega_4, 15). \\
 &p(W, Z) \leftarrow p(W, X), p(Y, Z), X < Z.
 \end{aligned}$$

$$\begin{aligned}
 g(p) : \\
 &5 \leq \omega_1, \omega_1 \leq 11, 5 \leq \omega_2, \omega_1 \leq \omega_2 - 6, \\
 &\omega_2 \leq 12, \omega_2 \leq \omega_3, 9 \leq \omega_3, \omega_3 \leq 12, \\
 &\omega_4 \leq \omega_3 + 2, 11 \leq \omega_4, \omega_4 \leq 14.
 \end{aligned}$$

We start by defining the declarative semantics without taking full consideration of the structure provided by the constraints. This enables us to show the meaning of the incomplete database in terms of the actual values of each possible world represented. This semantics is similar to that defined for definite constraint databases but results in a semantics which does not have a unique minimal model. We follow this initial definition by a semantics which takes full account of the constraint structure and generates a unique minimal model. This second semantics does not require us to deal with the full complexity of disjunctive logic programs and also enables us to provide a tractable semantics for infinite indefinite data.

Definition 9 (Generalised Herbrand Universe) The universe U_D of a database D with constraint constants is the set of all ground terms which can be formed from all constants and DC-constants appearing in D and the solution sets of all IC-constants appearing in indefinite facts of D , wrt equivalence.

Example 4 The universe of the set of clauses C in D is as follows:

$$U_D = \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$$

DC-constants are represented by intervals, for example, the DC-constant t_1 with the constraint $10 \leq t_1, t_1 \leq 15$ is represented by $\langle 10, 15 \rangle$. Partially specified DC-constants such as $\omega_1 \leq t_1, t_1 \leq \omega_1 5$ with the global constraint $11 \leq \omega_1, \omega_1 \leq 12$ is represented by the solution set $\{\langle 11, 15 \rangle, \langle 12, 15 \rangle\}$.

Definition 10 (Generalised Herbrand Base) The base B_D is the set of all ground definite facts which can be generated from the predicates in D and the universe U_D .

Example 5 The base of the set of clauses C is as follows:

$$B_D = \{p(9, 9), p(9, 10), p(9, 11), \dots, p(15, 15)\}$$

Definition 11 (Generalised Interpretation) An interpretation I is a subset of B_D containing all facts which are true under I . A fact $(p(t_1, \dots, t_m), c)$ is true under I iff $\forall (p(t_1, \dots, t_m), c), \exists [(p(t_1, \dots, t_m), c)] \in I$. That is, a ground fact which represents a possible domain instance for p appears in I . A rule $A_0 \leftarrow A_1 \wedge \dots \wedge A_n \wedge C_1 \wedge \dots \wedge C_m$ is true under I iff whenever $A_1\theta \wedge \dots \wedge A_n\theta \in I$ then it also holds that $A_0\theta \in I$.

Example 6 Possible interpretations for the example database are:

$$\begin{aligned}
 I_1 &= \{p(9, 10), p(10, 9), p(9, 9)\} \\
 I_2 &= \{p(9, 10), p(10, 10), p(9, 10)\} \\
 I_3 &= \{p(9, 10), p(10, 11), p(9, 11), p(13, 15)\}
 \end{aligned}$$

Definition 12 (Generalised Possible Model) An interpretation which is a possible solution to a clause C is called a Generalised Possible Model (GPMD) of D .

D has no unique minimal model. Instead, a minimal possible model is one for which no other model of D is strictly smaller.

A constraint atom A is true iff it is true in all minimal possible models. Ground facts of D , denoted $Ground(D)$, are exactly the ones which are true in all possible minimal models. That is if F is a ground fact, $D \models F$ iff F is true in every minimal model for D .

Each possible minimal model is a possible world for D according to the global conditions of D . However, generating each possible world would be intractable, particularly when the underlying domain is continuous or little (or no) information is available for the IC-constants appearing in the database.

In the following examples we show that by taking full account of the constraint structure it is unnecessary to generate each possible world explicitly in order to generate models for D , instead, we can continue to model domain instances using constraints. The advantage of this approach is that we obtain a unique minimal model for the database.

Definition 13 (Generalised Herbrand Universe with IC-constants) The Generalised Herbrand Universe with IC-constants UC_D is the set of all constants, DC-constants (with their associated intervals) and IC-constants appearing in D .

Example 7 The universe UC_D is represented as:

$$UC_D = \{9, 15, \omega_1, \omega_2, \omega_3, \omega_4\}$$

Definition 14 (Generalised Herbrand Base with IC-constants) The Generalised Herbrand Base with IC-constants $HBIC_D$ of D is the set of all atoms generated from the predicates in D and the universe UC_D . We note that when there is no IC-constant in D , $HBIC_D$ reduces to the (classical) Herbrand Base.

Example 8 A sample of the base $HBIC_D$ is as follows

$$HBIC_D = \{p(9, 15), p(9, \omega_1), p(9, \omega_2), p(9, \omega_3), \\ p(9, \omega_4), \dots, p(\omega_4, \omega_4)\}$$

The $HBIC_D$ is not adequate as a basis to describe the semantics of $IDC(L)$ databases as it contains no information about the incompleteness of IC-constants. We also require that the range of each predicate be associated with the $HBIC_D$. We call this the solution universe of D [10].

Definition 15 (Solution Universe) The solution universe s of a predicate p is the set of constraints which define its range. The solution universe S of a database D is therefore

$$S = \{c | c \in \text{consistent}(g(p))\}.$$

Example 9 The solution universe of the predicate $p(\omega_2, \omega_3)$ is

$$s = \{5 \leq \omega_2, \omega_1 \leq \omega_2 - 6, \omega_2 \leq 12, 12 \leq \omega_3, \\ 9 \leq \omega_3, \omega_3 \leq 12, \omega_4 \leq \omega_3 + 2, 11 \leq \omega_3, \\ \omega_4 \leq 14.5 \leq \omega_2, \omega_2 \leq 12, \omega_2 \leq \omega_3\}$$

This being the set of constraints necessary to define its range.

Definition 16 (Generalised Herbrand Base with Indefinite Values) Let D be a database, $HBIC_D$ be its Herbrand Base with IC-constants and S its solution universe. The Generalised Herbrand Base with Indefinite Values for database D , HBI_D is the set of ordered pairs $(p(t_1, \dots, t_m), s)$ where $p(t_1, \dots, t_m) \in HBIC_D$ and $s \in S$. We call elements of HBI_D extended atoms.

Example 10 The extended atom for $p(\omega_2, \omega_3)$ is

$$(p(\omega_2, \omega_3), 5 \leq \omega_2, \omega_1 \leq \omega_2 - 6, \omega_2 \leq 12, \\ \omega_2 \leq \omega_3, 9 \leq \omega_3, \omega_3 \leq 12, \omega_4 \leq \omega_3 + 2, \\ 11 \leq \omega_3, \omega_4 \leq 14.5 \leq \omega_2, \omega_2 \leq 12, \omega_2 \leq \omega_3).$$

A more compact representation is utilised through the remainder of this paper by separating out the binary constraints which relate to more than one predicate.

Example 11 The Generalised Herbrand Base with Indefinite values for the database D consists of the following extended atoms

$$(p(9, \omega_1), 5 \leq \omega_1, \omega_1 \leq 11, \omega_2 \leq \omega_3) \\ (p(\omega_2, \omega_3), 6 \leq \omega_2, \omega_2 \leq 11, 9 \leq \omega_3, \omega_3 \leq 12) \\ (p(\omega_4, 15), 11 \leq \omega_4, \omega_4 \leq 14)$$

along with the globally defined binary constraints.

A program with indefinite values can be viewed as a compact means of representing a set of possible worlds. For example, if D had the following atoms $(p_1(\omega_1), 5 \leq \omega_1, \omega_1 \leq 11)$, $(p_2(\omega_2), 3 \leq \omega_2, \omega_2 \leq 4)$, and $(p_3(5), \emptyset)$ this means that either 1 or 2 is in predicate p_1 , either 3 or 4 is in predicate p_2 and 5 is in predicate p_3 , we can view D as the representing four possible worlds: $D_1 = \{p_1(1), p_2(3), p_3(5)\}$, $D_2 = \{p_1(2), p_2(3), p_3(5)\}$, $D_3 = \{p_1(1), p_2(4), p_3(5)\}$, $D_4 = \{p_1(2), p_2(4), p_3(5)\}$. These can be obtained by instantiating each indefinite value ω_i with a constant from the constraint domain which satisfies any binary conditions. Each possible world can therefore be identified with a particular substitution [10] but it is not necessary to carry out this substitution in order to represent the semantics of the database.

Definition 17 (Extended Interpretation) An extended interpretation I is a subset of the HBI_D containing all the extended atoms which are true under I . An extended fact $(p(t_1, \dots, t_m), c) \in D$ is true under I iff $p(t_1, \dots, t_m, c) \in I$. A rule $R = A_0 \leftarrow A_1 \wedge \dots \wedge A_n \wedge C_1 \wedge \dots \wedge C_o$ is true under I iff whenever $A_1\theta_\delta \wedge \dots \wedge A_n\theta_\delta$ are true in I , $(A_i\theta_\delta = p(t_1, \dots, t_m), c)$ it also holds that $A_0\theta_\delta$ is true in I , where $\delta \in \{D, P\}$. If $\delta = D$ then I is called a definite extended interpretation otherwise, if $\delta = P$, I is called a possible extended interpretation.

Example 12 Three extended interpretations of the database D are:

$$I_1 = \{ (p(9, \omega_1), 5 \leq \omega_1, \omega_1 \leq 11), \\ (p(9, \omega_3), 6 \leq \omega_2, \omega_2 \leq 12), \\ (p(\omega_4, 15), 11 \leq \omega_4, \omega_4 \leq 14) \}$$

$$I_2 = \{ (p(9, \omega_1), 5 \leq \omega_1, \omega_1 \leq 11), \\ (p(\omega_2, \omega_3), 6 \leq \omega_2, \omega_2 \leq 12, 9 \leq \omega_3, \omega_3 \leq 12), \\ (p(\omega_4, 15), 11 \leq \omega_4, \omega_4 \leq 14) \}$$

$$I_3 = \{ (p(9, \omega_1), 5 \leq \omega_1, \omega_1 \leq 11), \\ (p(\omega_2, \omega_3), 6 \leq \omega_2, \omega_2 \leq 12, 9 \leq \omega_3, \omega_3 \leq 12), \\ (p(9, \omega_3), 9 \leq \omega_2, \omega_2 \leq 12), \\ (p(\omega_4, 15), 11 \leq \omega_4, \omega_4 \leq 14), \\ (p(9, 15), \\ (p(\omega_2, 15), 6 \leq \omega_2, \omega_2 \leq 12) \}$$

where I_1 and I_3 are possible interpretations and I_2 is a definite interpretation. As a consequence of allowing two types of interpretations, two types of model can be derived for a clause c or set of clauses s .

Definition 18 (Possible model) I is a possible model MP_D for a clause c (or set of clauses s) iff for each fact F in I , $S \models_P F$ (that is, F satisfies S).

Definition 19 (Definite model) I is a definite model MD_D for a clause c or set of clauses s iff for each fact F in I , $D \models_D F$ (that is, S entails F).

Definite (resp. possible) models have a unique minimal model MMD_D (resp. MMP_D) which is the intersection of all the definite (resp. possible) models of D .

The minimal model therefore consists of two parts, the data part and the constraint part, and defines the declarative meaning of a constraint database D . That is, it contains all constrained atoms which are definite (resp. possible) consequences of D . For any atom A , $D \models_P A$ ($D \models_D A$) if A is a member of minimal model MMD_D (MMP_D).

Example 13 I_1 is a possible model of D , I_2 is a definite model of D and I_3 is not a model.

2.2.2 Fixpoint Semantics

Using either of the declarative approaches to generate the model theoretic semantics of indefinite constraint databases would be inefficient as it would require generating all models of D and then finding the intersection.

However, $IDC(L)$ databases have a fixed point semantics which generates extended atoms. The fixpoint semantics of $IDC(L)$ starts from an interpretation containing the base extended atoms of D . A function T_D (resp. T_P) is applied to an interpretation I to obtain the interpretation which is its immediate consequence. The immediate consequence is generated by applying the rules in D to an interpretation I .

Example 14 The base interpretation for database D is as follows

$$I_0 = \{ (p(9, \omega_1), 5 \leq \omega_1, \omega_1 \leq 11), \\ (p(\omega_2, \omega_3), 6 \leq \omega_2, \omega_2 \leq 12, 9 \leq \omega_3, \omega_3 \leq 12), \\ (p(\omega_4, 15), 11 \leq \omega_4, \omega_4 \leq 14). \}$$

If T_P is applied to the base interpretation then the following interpretation is generated

$$I_I^P = I_0 \cup \{ (p(9, \omega_3), 9 \leq \omega_3, \omega_3 \leq 12), \\ p(9, 15), \\ (p(\omega_2, 15), 6 \leq \omega_2, \omega_2 \leq 12) \}$$

Application of the operator T_D produces

$$I_I^P = I_0 \cup \{ p(9, 15), (p(\omega_2, 15), 6 \leq \omega_2, \\ \omega_2 \leq 12) \}$$

The fixpoint function is repeatedly applied to each successive interpretation until the fixpoint is reached. In this example only one application is required.

2.2.3 Procedural semantics

Before we can define a procedural semantics for $IDC(L)$ we need to extend the definitions of unification and resolution for definite deductive databases.

Definition 20 (Unifiers for extended atoms) A set of constrained atoms $T = A_1, \dots, A_n$ is unifiable if there is a constrained substitution θ such that $A_i\theta = A_j\theta$ for $1 \leq i, j \leq n$ and $[\theta]_A \neq \emptyset$. θ is called a unifier of T . A unifier θ of T is an mgu iff for each unifier θ' of T , there is a substitution μ such that $\theta' = \theta\mu$.

Depending on which form of constrained substitution is used we say that a unifier θ is either a definite unifier θ_D or a possible unifier θ_P .

Example 15 Let $A_1 = p(Y, \omega_1)$ and $A_2 = p(a, X)$ and let $c_1 = \{6 \leq \omega_1, \omega_1 \leq 9\}$ and $c_2 = \{X = 8\}$. $\{A_1, A_2\}$ is definitely unifiable since $\theta_D = \{Ya, \omega_1 X\}$ is a definite unifier. If $c_2 = \{5 \leq X, X \leq 10\}$ then θ_D is not a definite unifier but is a possible unifier θ_{θ_P} since c_1 is consistent with c_2 .

Definition 21 (Definite (possible) SLD-(L)-derivation step) A definite (resp. possible) SLD(L)-derivation from the goal A on database D proceeds by selecting a clause $H \leftarrow A_1 \wedge \dots \wedge A_n \wedge C_1 \wedge \dots \wedge C_l$ in D such that $\theta_\delta, \delta \in \{D, P\}$, is the mgu of A and H . An SLD(L)-derivation of a goal A is a finite or infinite sequence $G_0 = G, G_1, \dots, G_n$ of goals, a sequence $C_1 \wedge \dots \wedge C_l$ of variants of clauses of D and a sequence $\theta_\delta, \dots, \theta_\delta$ of mgus, such that each G_{i+1} is derived from G_i and C_{i+1} using $\theta_{\delta_{i+1}}$.

Definition 22 (SLD(L)-refutation) An SLD(L)-refutation of $P \cup \{A\}$ is a finite SLD(L)-derivation of $P \cup \{A\}$ where the last element is a goal of the form (c) where c is the answer constraint of the derivation. All other finite SLD(L)-derivations are finitely failed.

3 Answering queries in $IDC(L)$

$IDC(L)$ extends existing query optimisations and evaluation algorithms for constraint databases to take account of the possible incompleteness of data. In particular, $IDC(L)$ allows three types of queries: definite queries denoted by the symbol Γ preceding a goal, possible queries denoted by the symbol Δ and conditional queries represented by unannotated goals.

Definite queries have a “truth in all possible worlds” semantics. Under this semantics a fact “satisfies” a query if every assignment of fully specified values to IC-constants appearing in the facts, consistent with the global constraint set, satisfies any conditions appearing in the query. To compute the answer set to this query requires checking whether the global conditions for that fact entail the query conditions. This corresponds to the definite model of the database.

Possible queries have a “truth in at least one possible world” semantics. Under this semantics, a fact “satisfies” the query if at least one assignment of fully specified values to IC-constants in the fact, consistent with the global condition, satisfies the constraint in the deductive relation. To compute the answer set to this type of query we need to check the satisfiability of the conjunction of constraints present in the global constraint set and the deductive relation. This is equivalent to a possible model for the query.

Conditional queries also have a possible model semantics, however, answers to conditional queries are accompanied by the conditions under which they are true. This is computed by integrating any query conditions with the global conditions for each fact and then simplifying the resulting constraint set.

The above query types require that the rule application step in bottom-up evaluation is replaced by the following steps:

1. For every goal atom appearing in the body of the rule, choose a fact contained in the corresponding relation.
2. If the query is a definite query, check that any conditions associated with the fact entail any conditions appearing in the body of the rule else, check that the two sets of conditions are consistent.
3. If step 2 is satisfied, unify the constants appearing in the consistent facts with variables appearing in the head of the rule and use variable elimination techniques to project relevant constraints onto any IC-constants which appear in the newly generated fact.

Once the set of answers has been generated, if the query is conditional, query conditions are integrated with the resulting conditions on facts appearing in answers to the query and simplified.

Prior to evaluation, we utilise various extensions to rewriting techniques in order to reduce the number of facts generated during query evaluation. This includes

- Query independent constraint propagation which occurs at assertion time and is outlined in a subsequent subsection.
- Extended rule adornment which identifies constant and constraint bindings generated by a query.
- Query dependent constraint propagation and sideways information passing which uses the extended adornments to rewrite rules so that subgoals with bound arguments are evaluated before unbound subgoals, and rule and global condition are placed at the earliest point at which they can be solved.

4 Updating ICD(L) databases

The simple extensions to query evaluation outlined in the previous section rest on a number of assumptions:

- That the information which we have about IC-constants is finitely representable.
- That all interesting constraints on IC-constants are explicitly represented so that it is only necessary to collect the constraints relevant to a fact together for query evaluation to occur.
- All constraints stored in the database are consistent.

Existing approaches to incomplete constraint databases do not address the issue of how this state is to be achieved in a dynamic database. Our research has therefore been concerned to explore the update issue wrt ICD(L) databases. The problems raised can be characterised as follows:

- The presence of global conditions means that data is interrelated and that updates to one piece of data may have an effect on another.
- The existence of the global constraint store means that the issue of belief revision needs to be taken into account when discussing updates.
- Additional complexity is introduced if updates with conditions are allowed. These may require that an update is performed only to a subset of the possible worlds of the database.

These problems mean that the standard set of update operators (insert, delete and modify) are not adequate. Instead we extend the work of Abitoule and Grahne and Winslett incomplete relational databases to develop the Dynamic IDC(L) (DIDC(L)) language.

4.1 The CDUL Language - Background

In general, the goal of updating a database D with a fact U is a database that contains all the information in D as well as the information needed to derive the new fact or to prove its falsity. The problem tackled here is slightly different and can be characterised as follows: Given a normal indefinite constraint database D consisting of an EDB, an IDB and a constraint program GCS, a partially solved database of D' , and an update U to apply to D , compute an update such that U' is a partially solved program wrt D . The method used to achieve U' is to find ways to identify those parts of the GCS affected by an update and leave unchanged those parts not affected. In IDC(L) this requires that when a new constraint is added to the GCS all interesting constraints which can be derived from it are also inserted into the GCS and, when a constraint is removed from the GCS, any constraints which have been derived from it are also removed. This approach means that the following categories of update can be distinguished.

4.1.1 Updates vs revisions of the database

Firstly, we distinguish between updates and revision of a database where an update is an action which changes the truth value of a fact in the extended database (EDB) of an IDC(L) database and a revision is an action which alters the information that is stored about a partially specified fact in the EDB. For example, suppose we wished to carry out the action

$$U = \text{“Insert } person(frank, hayward, \omega), 1840 \leq \omega \text{ where } person(arthur, hayward, \omega)\text{”}.$$

Assuming there is some fact which can be substituted for the where part of the clause, then if

$$person(frank, hayward, \omega)$$

is not true wrt the EDB of D prior to the insert and is true afterwards, then U is an update. On the other hand, if

$$person(frank, hayward, \omega)$$

is already true, then U represents a revision of an already known fact. In order to avoid confusion, in the remainder of this paper we call the act of changing the contents of a database a dynamic action where such an action may be either an update or a revision.

4.2 Passive vs active updates and revisions

Secondly, we distinguish between passive and active actions. A passive action changes the truth value or revises the known information about a single explicitly listed fact. An active action, on the other hand, causes the update or revision of other, existing atoms. It should be noted that in this thesis we do not consider active actions resulting in the update of other atoms, however, this category of actions would allow us to model actions on rules and is considered as one element of future research. Instead, we only consider active actions which result in the revision of another atom. For example, in the action

$U = \text{"Insert } person(frank, hayward, \omega), 1840 \leq \omega \text{ where } person(arthur, hayward, _)"$

only the fact $person(frank, hayward, \omega)$ was affected. However, the action

$U = \text{"Insert } person(frank, hayward, \omega), \omega < \omega_1 \text{ where } person(arthur, hayward, \omega_1)"$

not only affects the fact $person(frank, hayward, \omega)$, but also the fact $person(arthur, hayward, \omega_1)$ as the introduction of the constraint $\omega \leq \omega_1$ infers an additional constraint $\omega_1 \leq 1840$. We normally require that active updates or revisions only result in addition of data about other atoms if that data is consistent with what is already known. However, we recognise that the data could be inconsistent yet still represent the true state of the world. This situation is equivalent to modification in standard databases and is handled in a similar fashion. That is, we would require that the existing information is first deleted before the update/revision is carried out. This could, of course, be done automatically.

4.3 Definite vs possible world actions

A further distinction is made between definite and possible updates. This distinction arises because $IDC(L)$ has two semantics - a definite world semantics which generates interpretations of clauses based on whether they are entailed by EDB facts and a possible world semantics which generates interpretations for clauses based on their consistency with the EDB. Either of these semantics can be used to check the pre-condition(s) of the update.

4.4 Every world and some world actions

In this section we distinguish between actions which are performed on every world represented by a constraint and those which are performed on only a subset of possible worlds. For example, suppose $U = \text{"Insert } person(frank, hayward, \omega), 1840 \leq \omega \text{ where } person(arthur, hayward, X), X(1850)"$. If a possible world semantics is used to evaluate the precondition of this update then the body is true even though it is possible for Arthur to have been born after 1850. The head of the update is therefore only true in those possible worlds where Arthur was born before 1850, but is false in those possible worlds where he was born after 1850. Using the definite world semantics for the precondition would result in the head of the update being definitely true in all possible worlds.

4.5 The Update Language - Syntax

The primitive manipulations definable on a constraint database program are general insertion, general deletion, positive revision and negative revision of (constraint) atoms. These are denoted by the symbols $\{+, -, ++, --\}$ respectively. For the sake of readability, in this chapter we will refer to ground constraint atoms of the form $p(X) \leftarrow c$ using the form $(p(X), c)$ or, where the only constraints are equality, as $p(v)$.

Definition 23 A basic action U is of the form

$$\Delta\alpha(p_0(\overline{X}), c) \leftarrow p_1(\overline{X}_1), \dots, p_n(\overline{X}_n), C_{m+1}, \dots, C_l.$$

where Δ denotes either possible or definite, α is an action operator, the head of the action consists of $p_0(\overline{X})$ (also denoted by A), a predicate to be inserted or deleted, and c , a constraint expression to be inserted or deleted. Either $p_0(\overline{X})$ or c may be empty, but not both. The body of the action

$$p_1(\overline{X}_1), \dots, p_n(\overline{X}_n), C_{m+1}, \dots, C_l. \text{ (also denoted by } B)$$

is a pre-condition which consists of

$$p_1(\overline{X}_1), \dots, p_n(\overline{X}_n),$$

called the predicate pre-condition of U , and

$$C_{m+1}, \dots, C_l,$$

called the constraint pre-condition of U . If both c and C are empty then the action is equivalent to a standard (relational) update.

Definition 24 Each update to an IDC database must satisfy the following conditions:

1. The head constraint expression c is a conjunction of constraints of the form $t_1 \theta t_2$ where at least one t_i is an IC-constant appearing in p_0 . The second t_i may be any of the following:
 - an IC-constant appearing in p_0 ,
 - a variable appearing in $p_1(\bar{X}_1), \dots, p_0(\bar{X}_n)$, or
 - a constant from the constraint domain denoted by L .
2. Each C_i is a constraint $t_1 \theta t_2$ where at least one t_i is a variable appearing in one of the EDB predicates, the other t_i may be any of the following:
 - a variable appearing in $p_1(\bar{X}_1), \dots, p_0(\bar{X}_n)$, or
 - a constant from the constraint domain denoted by L .

Basic actions are combined declaratively to allow actions in sequences to express actions involving more than one atom.

Definition 25 Let $\{U_1; \dots; U_n\} \{n \leq 1\}$ be a set of basic actions. Then $U_1; \dots; U_n$ is an action operation. The execution of the action operation $U_1; \dots; U_n$ corresponds to executing the basic action U_1 first; then U_2 is executed, and so on, until the execution of the basic action U_n .

Example 16 The following example actions highlight the syntax of CDUL(L). Each example is followed by an informal description of its semantics. In the examples, each ω_i represents an IC-constant from the constraint domain dePCL [9].

$$+(person(harold, wilks, \omega), 1840 \leq \omega) \leftarrow (person(alfred, mcombs, \omega)).$$

$$+(person(robert, mcombs, \omega), 1840 \leq \omega, \omega \leq \omega_2) \leftarrow (person(alfred, hayward, \omega_3)).$$

The first update is a passive insert which results in a database where

$$(person(harold, wilks, \omega), 1840 \leq \omega)$$

is definitely true assuming that prior to the update

$$person(alfred, hayward, \omega)$$

is either definitely or possibly true wrt the EDB (depending on whether the update is a definite or possible update) and

$$(person(harold, wilks, \omega), 1840 \leq \omega)$$

is definitely false. This is achieved by adding the data part of the update to the EDB and generating the solution universe for the IC-constant. The second update is an active insert, the result of which is a database where

$$(person(robert, mcombs, \omega), 1840 \leq \omega, \omega \leq \omega_2)$$

is definitely true and is achieved by inserting $person(robert, mcombs, \omega_1)$ into the EDB and merging the associated solution universe with that for ω_3 . This insertion will only succeed if the new constraint is consistent with the existing constraints in the EDB and results in the addition of new constraints on the range of ω_3 (e.g. $1840 \leq \omega_3$).

$$++ (1830 \leq X) \leftarrow \\ \text{person}(\text{alice}, \text{mccombe}, X).$$

$$++ (X \leq \omega_2, 1830 \leq X) \leftarrow \\ \text{person}(\text{alice}, \text{mccombe}, X), \\ \text{person}(\text{robert}, \text{mccomb}, \omega_1).$$

The first revision is passive and provides additional information on the IC-constant which is substituted for the variable X . The result, assuming a substitution $\text{person}(\text{alice}, \text{mccombe}, \omega_3)$ exists for the body predicate, is to add the constraint $1830 \leq \omega_3$ to the solution universe for the substituted fact (if it is consistent) and to propagate any constraints which can be derived from the existing global conditions and the new constraint. The second revision is active and, again assuming $\text{person}(\text{alice}, \text{mccombe}, X)$ and $\text{person}(\text{robert}, \text{mccomb}, \omega_1)$ are true wrt the EDB, is to add the conditions $\omega_3 \leq \omega_1, 1830 \leq \omega_3$ to the global conditions of the two pre-condition predicates, as long as the conditions being inserted are consistent with the conditions in the store.

$$-- (\text{person}(\text{robert}, \text{mccombe}, \omega_1)) \leftarrow \\ (\text{person}(\text{alfred}, \text{hayward}, \omega_2)).$$

This is a deletion. If the global condition (should there be one) for $\text{person}(\text{robert}, \text{mccombe}, \omega_2)$ does not contain any constraints involving IC-constants which do not appear in the data part of the constraint fact, the deletion is passive and the result, assuming both $\text{person}(\text{alfred}, \text{hayward}, \omega_2)$ and $\text{person}(\text{robert}, \text{mccombe}, \omega_1)$ are true wrt the EDB, is to delete $\text{person}(\text{robert}, \text{mccombe}, \omega_1)$ from the EDB and to delete the solution universe for ω_1 . On the other hand, if the global condition for $\text{person}(\text{robert}, \text{mccombe}, \omega_1)$ does contain constraints involving IC-constants which do not appear in the data part of the constraint fact, then the deletion is active and requires that not only is the data for the constraint fact $\text{person}(\text{robert}, \text{mccombe}, \omega_1)$ be deleted but any constraints which were derived from its insertion.

$$-- \omega_3 \leq \omega_1, 1825 \leq \omega_3) \leftarrow \\ \text{person}(\text{alice}, \text{mccombe}, \omega_3), \\ \text{person}(\text{robert}, \text{mccomb}, \omega_1).$$

This is an active negative revision. Assuming the predicates in the body and constraints in the head of the action are true prior to the action, this action results in the deletion of those constraints and any information which has been inferred from them through constraint propagation.

$$+(\text{person}(\text{robert}, \text{mccombe}, \omega_1) \leftarrow \\ (\text{person}(\text{alfred}, \text{hayward}, \omega_2), \omega_2 < 1862).$$

This last example highlights the every world and some world aspects of IDC(L). The effect of the update is to action to be performed only in the subset of possible worlds denoted by the rule condition. In the example this means that the inserted atom and associated conditions are only true in worlds where Robert McCombe was born after 1840.

5 Update algorithms for DIDC(L)

We are currently developing and analysing a number of algorithms with which to implement the above operators. These algorithms, in addition to performing consistency checking and insertion/deletion or modification of EDB atoms and global conditions, perform the additional function of query independent constraint modification which enables the maintenance of the solution universe of the database.

This has the added advantage of allowing standard database retrieval techniques to be used when retrieving facts from the EDB during query evaluation. The algorithms also enable the revision of the constraint store on deletion of atoms and/or global conditions.

6 Implementation of IDCD Query and Update Language

The IDC(R) system (Indefinite Deductive Constraint database system with constraints over Reals) is a prototype constraint database system which implements several of the concepts introduced in this abstract. The Dynamic IDC(R) database language is implemented using the ECLIPSE system. Constraint facts p are stored in the knowledge base as deductive relations p_data . IC-constants are represented by an internally generated unique alpha-numeric identifier beginning with the letters 'ic'. Each IC-variable appearing in a fact will have an associated fact p_cond which stores the lower and upper bounds generated from the conditions on that constant. This information is stored separately from the data itself as it is not possible to know beforehand how many attributes in a fact may contain IC-constants which could result in facts of the same predicate having different arities. A third fact g_p stores the unary and binary conditions on IC-constants. The solution universe is maintained through a fully connected constraint network and a constraint dependency graph which enables the insertion and deletion of constraints as described above.

The query evaluation and update algorithms are being developed using the Prolog language and Constraint Handling rules within the ECLIPSE system.

7 Related Work

The concepts of "truth in all possible worlds" and "truth in some possible worlds" have been discussed by several authors including [1, 4, 9, 12].

The use of nulls for representing incomplete information in deductive databases has been presented by We draw on several ideas presented in these papers for the development of IDC(L). Constraints have been used to represent incomplete information in both [9, 12]. 1994) Our work is an extension of [9] which only considers relational databases, and, unlike [12], we develop semantics and recursive query answering strategies for both possible and definite queries.

The query answering strategy presented is a simple extension of that proposed by [8] for definite constraint databases while the rewriting strategy is an extension of constraint pushing as developed by [11] which takes account of global constraints.

The issue of updates and query independent constraint propagation has not, to our knowledge, been explored elsewhere.

8 Conclusion and Future Work

As stated above, we are currently working on the query and update algorithms and a prototype implementation for dynamic IDC(L). We have yet to develop algorithms for complex updates.

One possible area for future work is to explore the complexity of updates and queries with various constraints domains. Although work does exist on this in relation to queries we have not seen any exploration of the update problem

9 Bibliography

1. S. Abitoul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," in *Proceedings of the ACM SIGMOD conference on the management of data*. ACM Press, 1987, pp. 34-48.
2. S. Abitoul and G. Grahne, "Update semantics for incomplete databases," in *Proceedings of VLDB'85*. 1985, pp. 1-12.
3. F. Dong and L. V. S. Lakshmanan, "Deductive databases with incomplete information," in *Proceedings of the Joint International Conference and Symposium on Logic Programming*. 1992, pp. 303-317.

4. T. Imielinski and W. Lipski, "Incomplete information in relational databases," *Journal of the ACM*, vol. 31, pp. 761-791, 1984.
5. J. Jaffar and J. L. Lassez, "Constraint logic programming," in *Proceedings of the 14th ACM Symposium on the Principles of Programming Languages*. 1987, .
6. J. Jaffar, J.-L. Lassez, and M. J. Maher, "A logic programming language scheme," in *Logic Programming: Functions, relations and equations* D. DeGroot and G. Lindstrom, Eds. : Prentice Hall, 1987, pp. 441-468.
7. J. Jaffar and M. J. Maher, "Constraint Logic Programming: A Survey," *J. Logic Prog.* vol. 19/20, pp. 503-581, 1994.
8. P. Kanellakis, G. M. Kuper, and P. Z. Revesz, "Constraint Query Languages," in *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 1990, pp. 299-313.
9. M. Koubarakis, *Foundations of temporal constraint databases* Ph.D. Dissertation, National Technical University of Athens, Athens, 1994.
10. Y. Liu, "Null values in definite programs," in *Proceedings of the North American Conference on Logic Programming* MIT Press, 1990, pp. 273-288.
11. D. Srivastava and R. Ramakrishnan, "Pushing constraint selections," *J. Logic Prog.* vol. 1, 1993.
12. D. Srivastava, R. Ramakrishnan, and P. Z. Revesz, "Constraint objects", *Proc. 2nd Workshop on Principles and Practice of Constraint Programming*. Orcas Island, WA, May 1994.
13. M. Winslett, "A model-theoretic approach to updating incomplete databases," , *ACM Transactions on Database Systems*, vol. 13(2), pp. 167-196, 1988.